ALIBABA CLOUD

Alibaba Cloud

Container Service for Kubernetes

Best Practices

Document Version: 20220705

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Cluster

1.1. Select ECS instances to create the master and worker nodes of an ACK cluster

This topic describes how to select Elastic Compute Service (ECS) instances when you create the master and worker nodes of a Container Service for Kubernetes (ACK) cluster.

Considerations

When you create an ACK cluster, if you specify multiple small-sized ECS instances as the nodes of the cluster, the following issues may occur:

- The worker nodes that are created based on the small-sized ECS instances can provide only a small number of network resources.
- If one container consumes most of the resources that are provided by a small-sized ECS instance, the remaining resources of the instance are insufficient for specific operations and thus become idle. For example, the remaining resources cannot be used to create new containers or to restore failed containers. If multiple small-sized ECS instances are used in an ACK cluster, a large number of resources become idle.

We recommend that you use large-sized ECS instances to create the master and worker nodes. This type of ECS instance has the following benefits:

- Supports a high-bandwidth throughput. High-bandwidth applications can fully utilize the resources on the ECS instance.
- Increases the number of containers that can communicate with each other on the same ECS instance. This minimizes network latency.
- Optimizes the process to pull images. After an image is pulled to a large-sized ECS instance, the image can be used to create multiple containers on the ECS instance. To create these containers on a small-sized instance, more pull requests for the image must be made. If the nodes are scaled out in the ACK cluster to create the containers, it takes more time to create the ACK cluster. This also increases network latency.

Select ECS instances to create the master nodes

Multiple core components are deployed on a master node, such as etcd, kube-apiserver, and kube-controller-manager. These core components are critical to ensure cluster stability. When you create an ACK cluster, you must select the ECS instances that are suitable to create the master nodes. The specifications of the master nodes depend on the size of the ACK cluster. A larger ACK cluster requires the master nodes of higher specifications.

> **Note** The size of the ACK cluster is defined based on multiple factors, such as the number of nodes, the number of pods, the deployment frequency, and the number of requests that are sent to the cluster. In this topic, only the number of nodes is used to define the size of the ACK cluster.

The following table lists multiple common cluster sizes and the specifications of ECS instances that match the cluster sizes. You can select small-sized ECS instances in test environments. The listed specifications can be used to minimize the load on the master nodes.

Number of nodes in a cluster	Master node specification
1 to 5 nodes	4 vCPUs and 8 GiB of memory (We recommend that you do not use 2 vCPUs and 4 GiB of memory.)
6 to 20 nodes	4 vCPUs and 16 GiB of memory
21 to 100 nodes	8 vCPUs and 32 GiB of memory
100 to 200 nodes	16 vCPUs and 64 GiB of memory

Select ECS instances to create worker nodes

- You must select ECS instances with 4 or more vCPU cores and memory sizes must be greater than 8 GiB.
- Determine the specifications of worker nodes based on the total number of vCPUs in the ACK cluster and the maximum failure rate that is tolerated by the ACK cluster.

  For example, if the ACK cluster has 160 vCPUs and supports a maximum failure rate of 10%, we recommend that you select at least 10 ECS instances with 16 vCPUs for each instance. This ensures that at least 144 vCPUs can be consumed during peak hours. This limit is based on the following calculation: 160 vCPUs × 90% = 144 vCPUs. If the ACK cluster supports a maximum failure rate of 20%, we recommend that you select at least five ECS instances with 32 vCPUs for each instance. This ensures that at least 128 vCPUs can be consumed during peak hours. This limit is based on the following calculation: 160 vCPUs × 80% = 128 vCPUs. This way, if an ECS instance has faults, the remaining ECS instances can continue to serve your workloads and ensure high availability.
- Determine the vCPU-to-memory ratio. For applications that require high memory usage, such as Java applications, we recommend that you select the vCPU-to-memory ratio of 1:8.

Select ECS Bare Metal instances

We recommend that you use ECS Bare Metal instances in the following scenarios:

- The ACK cluster requires up to 1,000 vCPUs to serve your workloads. Each ECS Bare Metal instance has at least 96 vCPUs. You can use 10 or 11 ECS Bare Metal instances to create an ACK cluster.
- Shorten the time that is consumed to scale out a large number of containers. For example, traffic spikes may occur during popular e-commerce sales promotions. To process the traffic spikes, you can expand an ACK cluster based on ECS Bare Metal instances. You can use each ECS Bare Metal instance to add a large number of containers to the cluster.

ECS Bare Metal instances provide the following benefits for an ACK cluster:

- Ultra-high network performance. Remote direct memory access (RDMA) is supported. The Terway plug-in can be used to maximize hardware performance. This allows containers to reach each other across hosts with more than 9 Gbit/s of bandwidth.
- Optimal computing performance without jitter: ECS Bare Metal instances use chips that are developed by Alibaba Cloud to replace hypervisors. No resources are consumed to create or run virtual machines. The resource preemption issue is resolved.
- High security: ECS Bare Metal instances support encryption at the physical layer and encryption based

on Intel® Software Guard Extensions (Intel® SGX). This type of instance also provides trusted computing environments to support blockchain applications.

1.2. Recommended configurations for high reliability

This topic describes the recommended configurations for creating a Container Service for Kubernetes (ACK) cluster where applications can run stably and reliably.

Disk type and size

Disk type

- We recommend that you select an SSD.
- When you create an ACK cluster, select **Mount Data Disk** for worker nodes. The data disk is provided to */var/lib/docker* for storing local images in case the storage of the root disk is exhausted when the number of images increases. After a time period, the disk may contain images that are no longer needed. To clear these images, bring the machine offline, recreate a data disk, and then bring the machine online again.

Disk size

Docker images, system log, and application log are all stored in disks. Therefore, Kubernetes nodes require large disk space. When you create an ACK cluster, you must take into consideration the number of pods that you want to deploy on each node, the log size of each pod, the image size, the temporary data size, and the space reserved for the system.

We recommend that you reserve 8 GB of disk space for the operating system running on Elastic Compute Service (ECS) instances. The operating system occupies about 3 GB of disk space. The remaining space is used by Kubernetes resource objects.

Whether to create worker nodes when you create an ACK cluster

When you create an ACK cluster, you can set **Node Type** to the following values:

- **Pay-As-You-Go**: creates worker nodes when you create the ACK cluster.
- **Subscription**: does not create worker nodes when you create the ACK cluster. After the cluster is created, you can purchase ECS instances and add them to the cluster based on your requirements.

Network selection

- If you want to connect your cluster to external services such as ApsaraDB RDS for MySQL (RDS), you must use an existing virtual private cloud (VPC) instead of creating a VPC. VPCs are isolated from each other. You can create a vSwitch and attach the ECS instances of your cluster to the vSwitch. This facilitates cluster management.
- You can select one of the following network plug-ins when creating an ACK cluster: Terway and Flannel. For more information, see Work with Terway.
- Do not set a small CIDR block for the pod network. Otherwise, the number of supported nodes is limited. To set this parameter, consider the value that you want to specify for the **Pod Number for Node** parameter in the **Advanced Settings** section. For example, if you set the CIDR block of the pod network to X.X.X.X/16, it indicates that 256 × 256 IP addresses are assigned to your cluster. If the Pod Number for Node parameter is set to 128, the maximum number of nodes supported by your cluster is 512.

Use cross-zone deployment

Alibaba Cloud supports cross-region deployment and each region has one or more zones. Zones are isolated locations in a region. The power supply and network of each zone are independent. Using cross-zone deployment enables cross-zone disaster recovery, but increases network latency.

Claim resources of each pod

In an ACK cluster, excessive pods may be scheduled to one node. This overloads the node and the node can no longer provide services.

To avoid this issue, you can specify the requests parameter and the limits parameter when you deploy a pod in your cluster. This ensures that the pod is deployed on a node with sufficient resources. In the following example, the NGINX pod requires 1 CPU core and 1,024 MiB of memory. When the pod is running, the upper limit of resource usage is 2 CPU cores and 4,096 MiB of memory.

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    resources: # Resource claim
      requests:
        memory: "1024Mi"
        cpu: "1000m"
      limits:
        memory: "4096Mi"
        cpu: "2000m"
```

ACK uses a static resource scheduling method and calculates the remaining resources on each node in the following way: `Remaining resources = Total resources - Allocated resources`. The allocated resources are not equivalent to the resources that are used. When you manually run a resource-consuming program, ACK does not deduct the resources used by the program.

You must claim resources for all pods. For a pod that does not have resource claims, after it is scheduled to a node, the resources used by the pod are not deducted from the total resources of the node. As a result, excessive pods may be scheduled to this node.

Routine O&M

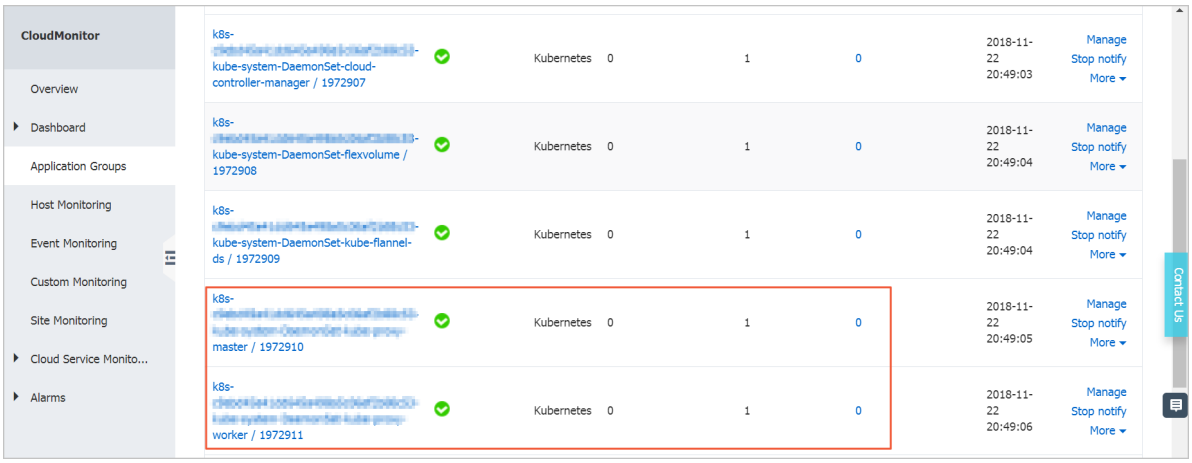
- Log

  When you create an ACK cluster, select **Enable Log Service**.

- Monitoring

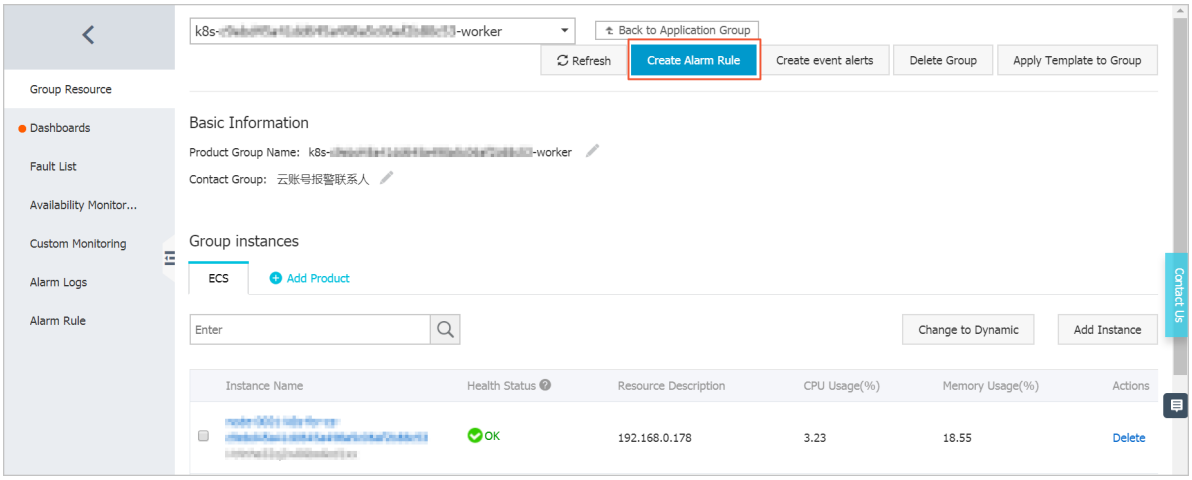
  ACK integrates the Cloud Monitor service of Alibaba Cloud. You can dynamically monitor your cluster by configuring node monitoring. You can add alert rules to help you locate the cause of unexpected high resource usage on nodes.

When you create an ACK cluster, two application groups are automatically created in the Cloud Monitor console: one for the master nodes of your cluster and the other for the worker nodes. You can add alert rules to these application groups. The rules apply to all nodes in the application groups. New nodes of the cluster are automatically classified into the application groups. You do not need to create alert rules for the new nodes again.



Therefore, you only need to configure alert rules for ECS instances.

> **Note**
>
> To monitor ECS instances, set alert rules for the **cpu**, **memory**, and **disk** resources in routine O&M. We recommend that you store */var/lib/docker* in a separate disk.



Waiting dependencies ready instead of terminating an application during startup

Some applications may have external dependencies. For example, an application may need to read data from a database or call the interface of another service. However, when the application starts, the database or the interface may be unready. In manual O&M, the application is terminated when the external dependencies are unready. This is known as failfast. This strategy is not suitable for ACK clusters. Most O&M activities in ACK are automated. You do not need to manually deploy an application, start the application on a selected node, or restart the application when it fails. Applications in ACK clusters are automatically restarted upon failures. You can also scale the number of pods through Horizontal Pod Autoscaler (HPA) to deal with increased loads.

For example, Application A is reliant on Application B and both applications run on the same node. The node restarts due to some reason. After the node is restarted, Application A is started while Application B is not. In this case, the dependency of Application A is unready. The traditional way terminates Application A in this case. After Application B is started, Application A must be manually started.

The best way for ACK is to check whether the dependencies are ready in a round robin manner and wait until all dependencies are ready instead of terminating the application. This can be achieved by using Init Container.

Configure the restart policy

Application processes running in a pod may be closed due to bugs in the code or excessive memory use. The pod fails when the processes are closed. You can set the restartPolicy parameter for the pod. This ensures that the pod can be automatically restarted upon failures.

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    restartPolicy: OnFailure #
```

Valid values of the restartPolicy parameter:

- *Always*: automatically restarts the pod in all cases.
- *OnFailure*: automatically restarts the pod upon failures (the state of the closed process is not 0).
- *Never*: never restarts the pod.

Configure liveness probes and readiness probes

A pod may be unable to provide services even if the pod is in the Running state. The processes in a running pod may be locked. Consequently, the pod cannot provide services. However, Kubernetes does not restart such a pod because the pod is still running. Therefore, you must configure liveness probes for all pods in a cluster. The probes check whether the pods are alive and can provide services. When a liveness probe detects exceptions in a pod, the pod is automatically restarted.

A readiness probe is used to determine whether a pod is ready to provide services. It takes some time for an application to initialize during startup. During the initialization, the pod where the application runs cannot provide services. A readiness probe is used to inform Ingresses or Services whether the pod is ready to receive network traffic. When pod errors are detected by readiness probes, Kubernetes stops forwarding network traffic to the pod.

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    livenessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 3
    readinessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
```

Run only one process in each container

Users who are new to the container service tend to use containers as virtual machines (VMs) and run multiple processes in one container. The processes include the monitoring process, logging process, sshd process, and the systemd. This causes the following two issues:

- It becomes complex to determine the resource usage of a pod. Implementing resource limit also becomes difficult.
- Running only one process in each container ensures that the container engine can detect process failures and restart the container upon each failure. If a container contains multiple processes, the container may be not affected when one of the processes is terminated. The external container engine is unaware of the terminated process and therefore does not perform any action on the container. However, the container may not be able to function as expected in this case.

ACK allows you to run multiple processes together. For example, if you want NGINX and php-fpm to communicate with each other by using a UNIX domain socket, you can use a pod that contains two containers. Then, put the UNIX domain socket into a volume shared by the two containers.

Eliminate single points of failure (SPOF)

If an application uses only one ECS instance, the application becomes unavailable during the time period in which the ECS instance is restarted upon a failure. The application also becomes unavailable when it is upgraded or when a new version is released. Therefore, we recommend that you do not directly run applications in pods. Instead, deploy applications by using Deployments or StatefulSets and use more than two pods for each application.

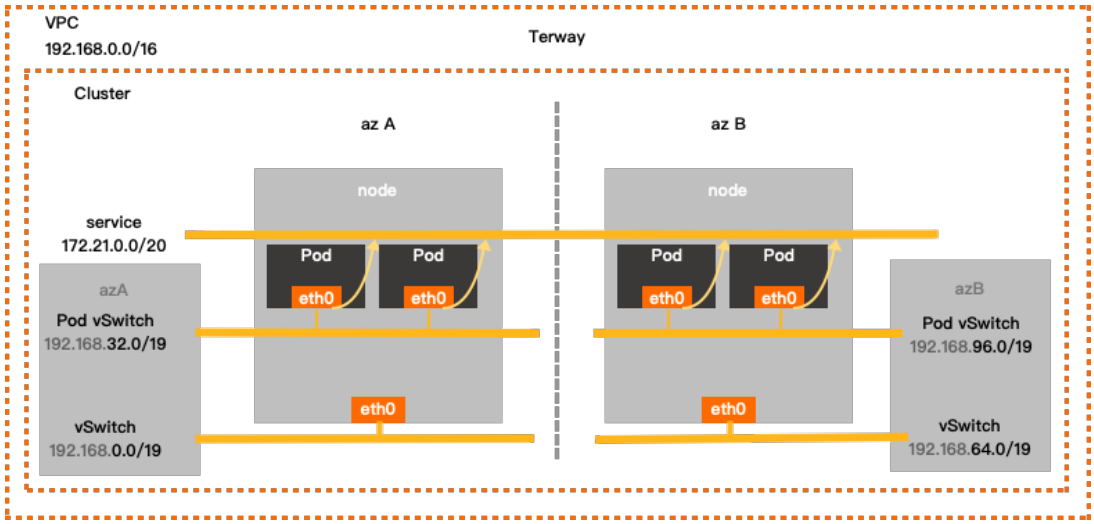
1.3. Plan CIDR blocks for an ACK cluster

When you create a Container Service for Kubernetes (ACK) cluster, you must specify a virtual private cloud (VPC), vSwitches, the CIDR block of pods, and the CIDR block of Services. Therefore, we recommend that you plan the IP address of each Elastic Compute Service (ECS) instance in the cluster, the CIDR block of pods, and the CIDR block of Services before you create an ACK cluster. This topic describes how to plan CIDR blocks for an ACK cluster deployed in a VPC and how each CIDR block is used.

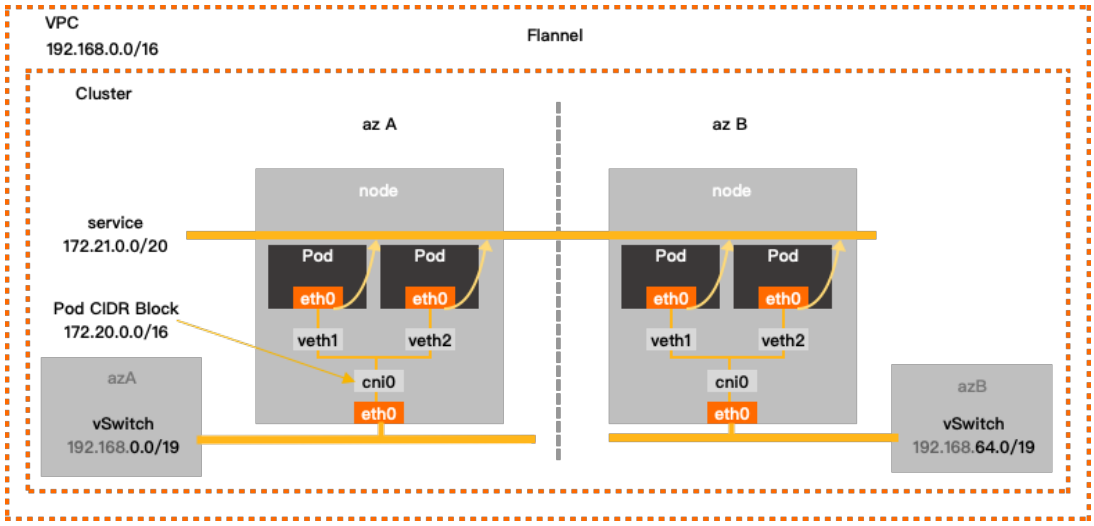
VPC-related CIDR blocks and cluster-related CIDR blocks

Before you create a VPC, you must plan the CIDR block of the VPC and the CIDR blocks of vSwitches in the VPC. Before you create an ACK cluster, you must plan the CIDR block of pods and the CIDR block of Services. ACK supports the Terway and Flannel plug-ins. The following figures show the network architectures of ACK cluster that uses Terway and Flannel.

Terway



Flannel



To install Terway or Flannel in an ACK cluster, you must specify CIDR blocks and other parameters as described in the following table.

Parameter	Terway	Flannel

Parameter	Terway	Flannel
VPC	When you create a VPC, you must select a CIDR block for the VPC. Valid values: 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16.	
VSwitch	The IP addresses of ECS instances are assigned from vSwitches. This allows nodes in a cluster to communicate with each other. The CIDR blocks that you specify when you create vSwitches in the VPC must be subsets of the VPC CIDR block. This means that the vSwitch CIDR blocks must fall within or the same as the VPC CIDR block. When you set this parameter, take note of the following items: • Select one or more vSwitches in the VPC. • IP addresses from the CIDR block of a vSwitch are allocated to the ECS instances that are attached to the vSwitch. • You can create multiple vSwitches in a VPC. However, the CIDR blocks of these vSwitches cannot overlap with each other. • A node vSwitch and the vSwitch that assigns IP addresses to pods on the node must be in the same zone. For more information about zones, see Regions and zones.	The IP addresses of ECS instances are assigned from vSwitches. This allows nodes in a cluster to communicate with each other. The CIDR blocks that you specify when you create vSwitches in the VPC must be subsets of the VPC CIDR block. This means that the vSwitch CIDR blocks must fall within or the same as the VPC CIDR block. When you set this parameter, take note of the following items: • Select one or more vSwitches in the VPC. • IP addresses from the CIDR block of a vSwitch are allocated to the ECS instances that are attached to the vSwitch. • You can create multiple vSwitches in a VPC. However, the CIDR blocks of these vSwitches cannot overlap with each other.

Parameter	Terway	Flannel
Pod VSwitch	The IP addresses of pods are assigned from the CIDR block of the pod vSwitches. This allows pods to communicate with each other. A pod is a group of containers in a Kubernetes cluster. Each pod has an IP address. The CIDR blocks that you specify when you create pod vSwitches in the VPC must be subsets of the VPC CIDR block. When you set this parameter, take note of the following items: • Select one or more vSwitches in the VPC. • In an ACK cluster that has Terway installed, the IP addresses of pods are allocated from pod vSwitches. • The CIDR blocks of pod vSwitches cannot overlap with the CIDR blocks of **vSwitches**. • The CIDR blocks of pod vSwitches cannot overlap with the CIDR block specified by **Service CIDR**. • A node vSwitch and the vSwitch that assigns IP addresses to pods on the node must be in the same zone. For more information about zones, see Regions and zones.	You do not need to set this parameter if you choose to install Flannel in an ACK cluster.
Pod CIDR Block	You do not need to set this parameter if you choose to install Terway in an ACK cluster.	The IP addresses of pods are allocated from the pod CIDR block. This allows pods to communicate with each other. A pod is a group of containers in a Kubernetes cluster. Each pod has an IP address. When you set this parameter, take note of the following items: • Enter a CIDR block in the Pod CIDR Block field. • The CIDR block of pods cannot overlap with the CIDR blocks of **vSwitches**. • The CIDR block of pods cannot overlap with the CIDR block specified by **Service CIDR**.

Parameter	Terway	Flannel
Service CIDR	The CIDR block of Services. Service is an abstraction in Kubernetes. Each `ClusterIP` Service has an IP address. When you set this parameter, take note of the following items: • The IP address of a Service is effective only within the Kubernetes cluster. • The CIDR block of Services cannot overlap with the CIDR blocks of **vSwitches**. • The CIDR block of Services cannot overlap with the CIDR blocks of **Pod vSwitches**.	The CIDR block of Services. Service is an abstraction in Kubernetes. Each `ClusterIP` Service has an IP address. When you set this parameter, take note of the following items: • The IP address of a Service is effective only within the Kubernetes cluster. • The CIDR block of Services cannot overlap with the CIDR blocks of **vSwitches**. • The CIDR block of Services cannot overlap with **Pod CIDR Block**.

Considerations

Network planning

To use Kubernetes clusters that are supported by ACK on Alibaba Cloud, you must first set up networks for the clusters based on the cluster sizes and business scenarios. You can refer to the following tables to set up networks for Kubernetes clusters. Change specifications as needed in unspecified scenarios.

Cluster size	Scenario	VPC	Zone
< 100 nodes	Regular business	Single VPC	1
Unlimited	Cross-zone deployment is required	Single VPC	≥ 2
Unlimited	High reliability and cross-region deployment are required	Multiple VPCs	≥ 2

The following tables describe how to plan CIDR blocks for clusters that use Flannel or Terway.

- **Clusters that use Flannel**

VPC CIDR block	vSwitch CIDR block	Pod CIDR block	Service CIDR block	Maximum number of pod IP addresses
192.168.0.0/16	192.168.0.0/24	172.20.0.0/16	172.21.0.0/20	65536

- **Clusters that use Terway**
  - Exclusive elastic network interface (ENI) mode or IPVLAN mode is enabled

VPC CIDR block	vSwitch CIDR block	CIDR block of pod vSwitches	Service CIDR block	Maximum number of pod IP addresses
192.168.0.0/16	192.168.0.0/19	192.168.32.0/19	172.21.0.0/20	8192

- **Multi-zone deployment**

VPC CIDR block	vSwitch CIDR block	CIDR block of pod vSwitches	Service CIDR block	Maximum number of pod IP addresses
192.168.0.0/16	Zone I 192.168.0.0/19	192.168.32.0/19	172.21.0.0/20	8192
	Zone J 192.168.64.0/19	192.168.96.0/19		8192

Plan the network of a VPC

Plan CIDR blocks for clusters

How to plan CIDR blocks

- **Scenario 1: One VPC and one Kubernetes cluster**

  This is the simplest scenario. The CIDR block of a VPC is specified when you create the VPC. When you create a cluster in the VPC, make sure that the CIDR block of pods and the CIDR block of Services do not overlap with the VPC CIDR block.

- **Scenario 2: One VPC and multiple Kubernetes clusters**

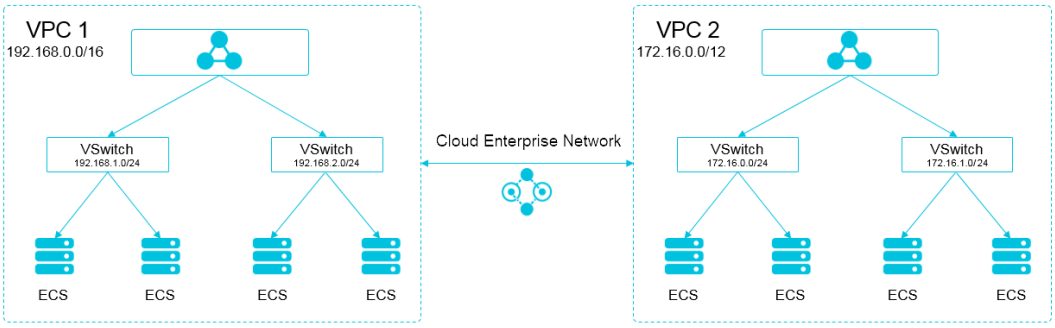
  You want to create more than one cluster in a VPC.

  - The CIDR block of the VPC is specified when you create the VPC. When you create clusters in the VPC, make sure that the VPC CIDR block, Service CIDR block, and pod CIDR block of each cluster do not overlap with one another.
  - The Service CIDR blocks of the clusters can overlap with each other. However, the pod CIDR blocks cannot overlap with each other.
  - In the default network mode (Flannel), the packets of pods must be forwarded by the VPC router. ACK automatically generates a route table for each destination pod CIDR block on the VPC router.

  > **Note** In this case, a pod in one cluster can communicate with the pods and ECS instances in another cluster. However, the pod cannot communicate with the Services in another cluster.

- **Scenario 3: Two connected VPCs**

  If two VPCs are connected, you can use the route table of one VPC to specify the packets that you want to send to the other VPC. The CIDR block of VPC 1 is 192.168.0.0/16 and the CIDR block of VPC 2 is 172.16.0.0/12, as shown in the following figure. You can use the route table of VPC 1 to forward all packets that are destined for 172.16.0.0/12 to VPC 2.



Connected VPCs

VPC	CIDR block	Destination CIDR block	Destination VPC
VPC 1	192.168.0.0/16	172.16.0.0/12	VPC 2
VPC 2	172.16.0.0/12	192.168.0.0/16	VPC 1

In this scenario, make sure that the following conditions are met when you create a cluster in VPC 1 or VPC 2:

- The CIDR blocks of the cluster do not overlap with the CIDR block of VPC 1.
- The CIDR blocks of the cluster do not overlap with the CIDR block of VPC 2.
- The CIDR blocks of the cluster do not overlap with those of other clusters.
- The CIDR blocks of the cluster do not overlap with those of pods.
- The CIDR blocks of the cluster do not overlap with those of Services.

In this example, you can set the pod CIDR block of the cluster to a subset of 10.0.0.0/8.

> **Note** All IP addresses in the destination CIDR block of VPC 2 can be considered in use. Therefore, the CIDR blocks of the cluster cannot overlap with the destination CIDR block.

To access pods in VPC 1 from VPC 2, you must configure a route in VPC 2. The route must point to the pod CIDR block of a cluster in VPC 1.

- **Scenario 4: a VPC connected to a data center**

  If a VPC is connected to a data center, packets of specific CIDR blocks are routed to the data center. In this case, the pod CIDR block of a cluster in the VPC cannot overlap with these CIDR blocks. To access pods in the VPC from the data center, you must configure a route in the data center to enable VBR-to-VPC peering connection.

Related information

- Overview
- Work with Terway
- Use network policies
- FAQ about network management

1.4. Increase the capacity of etcd for dedicated Kubernetes clusters

Dedicated Kubernetes clusters use etcd v3.3.8 by default, which functions as a backend store that supports up to 2 GB. A cluster can no longer write data to etcd if the data stored in etcd reaches 2 GB. This topic describes how to upgrade etcd to v3.4.3. This increases the capacity of etcd for dedicated Kubernetes clusters.

Prerequisites

- The etcd version is earlier than 3.4.3.
- You want to store more than 2 GB of data in etcd. The upgrade is not mandatory if you do not require it.

Context

The publicly available version of etcd 3.4.3 provides up to 100 GB of storage.

Procedure

1. Use SSH to log on to the master nodes where etcd is installed and verify that the current etcd version is 3.3.8.
2. Run the following shell script to download etcd-v3.4.3 binary and start the new version:

   > **Note**
   > - Perform the upgrade on each node. Wait until the upgraded node is in the Ready state before you move on to the next one.
   > - etcd is highly available. You can still access etcd during the upgrade.

   ```
   ! /usr/bin/env bash
   ```

```
etcdbin=http://aliacs-k8s-cn-hangzhou.oss.aliyuncs.com/etcd/etcd-v3.4.3/etcd
etcdctlbin=http://aliacs-k8s-cn-hangzhou.oss.aliyuncs.com/etcd/etcd-v3.4.3/etcdctl
function download(){
    wget -O etcd ${etcdbin}
    wget -O etcdctl ${etcdctlbin}
    chmod +x {etcd,etcdctl}
    mv etcd /usr/bin/etcd
    mv etcdctl /usr/bin/etcdctl
    etcd --version
}
function config() {
    ETCD_FILE=/lib/systemd/system/etcd.service
    sed -i "/ETCD_EXPERIMENTAL_BACKEND_BBOLT_FREELIST_TYPE/ d" ${ETCD_FILE}
    sed -i "/ETCD_QUOTA_BACKEND_BYTES/ d" ${ETCD_FILE}
    sed -i "/^\[Service\]/a\Environment=\"ETCD_EXPERIMENTAL_BACKEND_BBOLT_FREELIST_TYPE
=map\"" ${ETCD_FILE}
    sed -i "/^\[Service\]/a\Environment=\"ETCD_QUOTA_BACKEND_BYTES=100000000000\"" ${ET
CD_FILE}
    sed -i "s/initial-cluster-state new/initial-cluster-state existing/g" ${ETCD_FILE}
    systemctl daemon-reload
    systemctl restart etcd
}
download; config
ENDPOINTS=`ps -eaf|grep etcd-servers|grep -v grep|awk -F "=" '{print $22}'|awk -F " " '
{print $1}'`
ETCDCTL_API=3 etcdctl \
        --endpoints=${ENDPOINTS}         \
        --cacert=/var/lib/etcd/cert/ca.pem         \
        --cert=/var/lib/etcd/cert/etcd-client.pem         \
        --key=/var/lib/etcd/cert/etcd-client-key.pem \
        member list
```

3. Run the following command to check whether etcd is running:

```
ps aux|grep etcd
```

What's next

Check the health status of etcd.

```
ETCDCTL_API=3 etcdctl --endpoints=${ENDPOINTS} \
      --cacert=/var/lib/etcd/cert/ca.pem \
      --cert=/var/lib/etcd/cert/etcd-client.pem \
      --key=/var/lib/etcd/cert/etcd-client-key.pem endpoint health
```

```
ENDPOINTS is healthy
```

2.Node pools

2.1. Best practices for preemptible instance-based node pools

Preemptible instances are a type of on-demand instances and are discounted compared to pay-as-you-go instances. You can specify a ratio of preemptible instances to pay-as-you-go instances in a node pool. This helps reduce the resource cost of the node pool. This topic describes preemptible instance-based node pools and the use scenarios. This topic also describes how to configure instance types for a preemptible instance-based node pool.

Context

Preemptible instances use the pay-as-you-go billing method. You pay for preemptible instances after you use them. Bills are calculated based on the market price and billing duration. For more information, see Preemptible instances.

Introduction

A preemptible instance-based node pool consists of preemptible instances and pay-as-you-go instances. You must specify a ratio of preemptible instances to pay-as-you-go instances in a preemptible instance-based node pool.

Preemptible instances are pay-as-you-go instances whose prices dynamically fluctuate based on factors such as the resource stock. Therefore, the price of a preemptible instance can be up to 90% lower than the price of a pay-as-you-go instance. The market price of a preemptible instance fluctuates based on changes in supply and demand for the instance type. When you create a preemptible instance, you must select a bidding mode to bid for a specified instance type. If your bid price is higher than the market price and the stock of the instance type is sufficient, a preemptible instance is created.

After a preemptible instance is created, you can use the preemptible instance in the same way you use a pay-as-you-go instance. You can also use the preemptible instance together with other cloud resources such as cloud disks and elastic IP addresses (EIPs). The default protection period of a preemptible instance is 1 hour. After the protection period ends, the system checks the market price and resource stock of the instance type every 5 minutes. If the market price is higher than your bid price or if the stock of the instance type is insufficient, the preemptible instance is released.

Scenarios

- **Preemptible instance-based node pools**

  Preemptible instances in a preemptible instance-based node pool may be unexpectedly reclaimed. Therefore, preemptible instance-based node pools are suitable for stateless applications with high fault tolerance. You can use preemptible instance-based node pools for workloads that use batch processing, machine learning training jobs, queued transaction processing applications, applications that use the REST API, and extract, transform, load (ETL) workloads of big data computing, such as Apache Spark jobs.

If you want to deploy workloads in a preemptible instance-based node pool, you must make sure that the workloads are tolerant to node resource unavailability. If the workloads are not tolerant to node resource unavailability, we recommend that you deploy the workloads in node pools that consist of pay-as-you-go instances or subscription instances. Workloads that are not tolerant to node resource unavailability include but not limited to the following types:

- Cluster management tools, such as monitoring tools and O&M tools.
- Stateful workloads or applications, such as database services.

- **Preemptible instance-based node pools that have auto scaling enabled**

  If your workloads are deployed in a preemptible instance-based node pool and process user traffic that fluctuates in a specific pattern, we recommend that you enable auto scaling for the preemptible instance-based node pool in which your workloads are deployed.

  After you enable auto scaling, the auto scaling component automatically checks whether the node pool needs to be scaled out to host more pods and automatically scales in the node pool when the scale-in threshold is reached. Preemptible instance-based node pools that have auto scaling enabled can scale out faster than preemptible instance-based node pools that have auto scaling disabled. Idle resources are released more promptly after you enable auto scaling for a preemptible instance-based node pool. Faster scale-out helps ensure sufficient node resources when preemptible instances are reclaimed. The automatic release of idle resources helps reduce the resource cost.

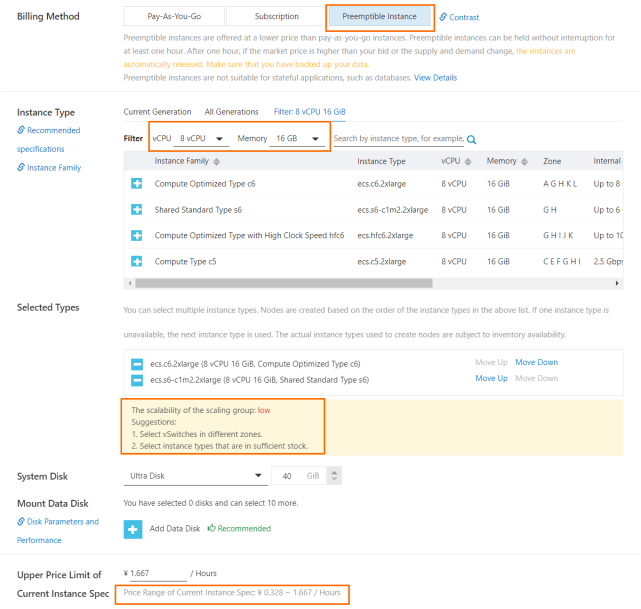
Select instance types for preemptible instances

We recommend that you select instance types based on your business requirements and balance the instance resource stock, resource cost, and instance performance. Elastic Compute Service (ECS) provides a large variety of instance types to meet the requirement of different workloads. When you use a preemptible instance-based node pool, you must select a combination of instance types that have minimal impacts on your workloads when preemptible instances are reclaimed.

You can use one of the following methods to select instance types:

- **Use the ACK console**

After you select instance types in the console, suggestions on the selected instance types automatically appear on the page. When you create or modify a node pool in the ACK console, instance types that are available in the region that you selected are displayed on the page. You can select instance types based on your business requirements. After you select instance types, the ACK console automatically provides suggestions on the selected instance types, and displays the scalability of the node pool and the price range of each preemptible instance. You can change the instance types and specify the price upper limit based on the information. For more information about how to create or modify a node pool, see Create a node pool.



- **Use the spot-instance-advisor command-line tool**

  `spot-instance-advisor` is an open source command-line tool provided by ACK. You can use spot-instance-advisor to query the historical prices and current price of a preemptible instance. `spot-instance-advisor` calls API operations to obtain the historical prices of the instance types in a region. spot-instance-advisor calculates the hourly vCPU unit price based on the obtained statistics and lists the instance types with the lowest hourly vCPU unit prices. spot-instance-advisor also calculates the entropy of the hourly vCPU unit price of each instance type. A higher value of entropy indicates more frequent fluctuation in the price. We recommend that you select instance types whose hourly vCPU unit prices have a low value of entropy.

  `spot-instance-advisor` supports the following parameters:

```
Usage of ./spot-instance-advisor:
  -accessKeyId string
        Your accessKeyId of cloud account
  -accessKeySecret string
        Your accessKeySecret of cloud account
  -cutoff int
        Discount of the spot instance prices (default 2)
  -family string
        The spot instance family you want (e.g. ecs.n1,ecs.n2)
  -limit int
        Limit of the spot instances (default 20)
  -maxcpu int
        Max cores of spot instances  (default 32)
  -maxmem int
        Max memory of spot instances (default 64)
  -mincpu int
        Min cores of spot instances (default 1)
  -minmem int
        Min memory of spot instances (default 2)
  -region string
        The region of spot instances (default "cn-hangzhou")
  -resolution int
        The window of price history analysis (default 7)
```

Run the following command to query the prices of the instance types in the current region:

```
./spot-instance-advisor --accessKeyId=<id> --accessKeySecret=<secret> --region=<cn-zhangj
iakou>
```

Note The `accessKeyId`, `accessKeySecret`, and `region` parameters are required. Set the parameters to the actual values.

Expected output:

```
Initialize cache ready with 619 kinds of instanceTypes
Filter 93 of 98 kinds of instanceTypes.
Fetch 93 kinds of InstanceTypes prices successfully.
Successfully compare 199 kinds of instanceTypes
      InstanceTypeId               ZoneId     Price(Core)        Discount            ratio
        ecs.c6.large     cn-zhangjiakou-c          0.0135             1.0              0.0
        ecs.c6.large     cn-zhangjiakou-a          0.0135             1.0              0.0
      ecs.c6.2xlarge     cn-zhangjiakou-a          0.0136             1.0              0.0
      ecs.c6.2xlarge     cn-zhangjiakou-c          0.0136             1.0              0.0
      ecs.c6.3xlarge     cn-zhangjiakou-a          0.0137             1.0              0.0
      ecs.c6.3xlarge     cn-zhangjiakou-c          0.0137             1.0              0.0
       ecs.c6.xlarge     cn-zhangjiakou-c          0.0138             1.0              0.0
       ecs.c6.xlarge     cn-zhangjiakou-a          0.0138             1.0              0.0
     ecs.hfc6.xlarge     cn-zhangjiakou-a          0.0158             1.0              0.0
      ecs.hfc6.large     cn-zhangjiakou-a          0.0160             1.0              0.0
      ecs.hfc6.large     cn-zhangjiakou-c          0.0160             1.0              0.0
      ecs.g6.3xlarge     cn-zhangjiakou-a          0.0175             1.0              0.0
      ecs.g6.3xlarge     cn-zhangjiakou-c          0.0175             1.0              0.0
        ecs.g6.large     cn-zhangjiakou-a          0.0175             1.0              0.0
       ecs.g6.xlarge     cn-zhangjiakou-a          0.0175             1.0              0.0
      ecs.g6.2xlarge     cn-zhangjiakou-a          0.0175             1.0              1.0
      ecs.g6.2xlarge     cn-zhangjiakou-c          0.0175             1.0              3.0
        ecs.g6.large     cn-zhangjiakou-c          0.0175             1.0              30.
8
       ecs.g6.xlarge     cn-zhangjiakou-c          0.0175             1.0              9.7
      ecs.hfg6.large     cn-zhangjiakou-c          0.0195             1.0              0.2
```

For the top-ranking instance types in the output, the hourly vCPU unit price and the value of the price entropy are low. The value in the `ratio` column indicates the value of the price entropy. Instance types other than the top-ranking instance types offer a 90% discount but have a higher value of the price entropy. When you select instance types, we recommend that you select the top-ranking instance types with lower prices and lower values of the price entropy.

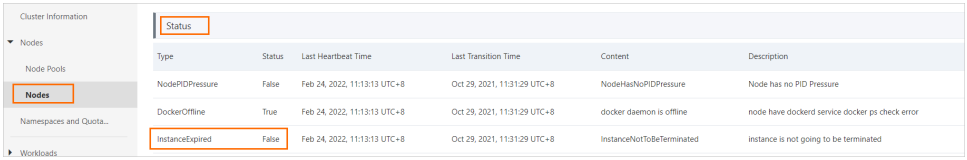
Graceful shutdown of preemptible instances

The graceful shutdown of preemptible instances includes the following processes: monitoring and notification, supplementation of preemptible instances before the preemptible instances are reclaimed, and custom operations on nodes to be released.

- **Monitoring and notification**

  ACK uses Node Problem Detector (NPD) to monitor the status of preemptible instances and sends notifications when preemptible instances are about to expire.

  - If a preemptible instance is not about to expire, the value of **InstanceExpired** is **False**.



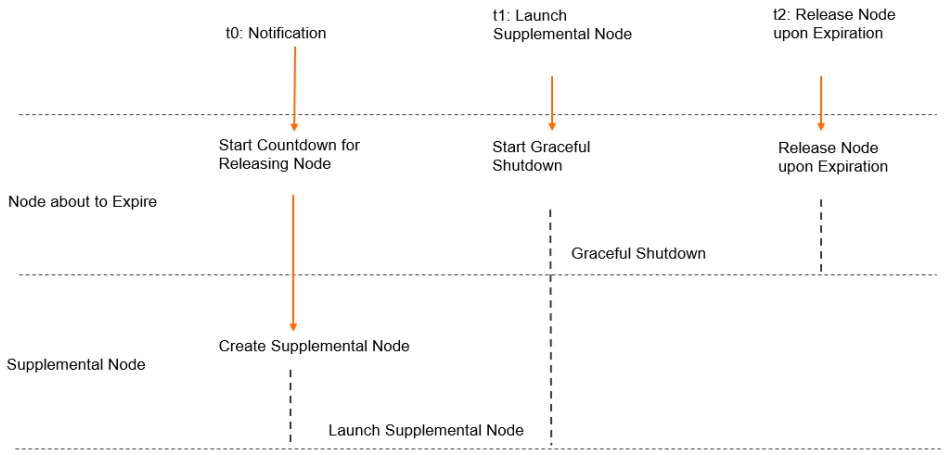
- If a preemptible instance is about to expire, the value of **InstanceExpired** is **True**. In this case, ACK generates a cluster event to remind you that the preemptible instance is about to expire.

- **Supplementation of preemptible instances before the preemptible instances are reclaimed**

  After a preemptible instance expires and is released, services that are deployed on the instances are suspended. ACK provides various methods to help you respond to the expiration and release of preemptible instances at the earliest opportunity. You can configure auto scaling for preemptible instances, monitor the status of preemptible instances, and configure the system to send notifications when preemptible instances are about to expire. However, these methods are implemented after preemptible instances are reclaimed. Available resources in the cluster are not increased until new instances are added to the cluster. To resolve this issue, ACK supplements preemptible instances before they are reclaimed. This feature enables ACK to create new preemptible instances before existing preemptible instances expire.

  After you enable this feature, ACK automatically checks whether preemptible instances are about to expire. If a preemptible instance is about to expire, ACK automatically triggers a scale-out activity to add a new preemptible instance. The newly created instance is known as a supplemental instance. After the supplemental instance starts running, the release process of the preemptible instance starts. The status of the preemptible instance is set to unschedulable. Then, the preemptible instance is drained and removed. This way, workloads on the preemptible instance are smoothly migrated to other nodes in the cluster to avoid service interruptions.

  > **Note** The supplementation of preemptible instances does not interrupt the reclaim of preemptible instances that are about to expire. A preemptible instance will be reclaimed 5 minutes after ACK notifies that the instance is about to expire, regardless of whether supplemental instances are created.



- **Custom operations on nodes to be released**

When you gracefully shut down a node, you may need to perform other operations on the node, for example, deleting the node information from the DNS records. To perform custom operations on nodes to be released, we recommend that you monitor the value of the **InstanceExpired** field in the node status or configure a listener to listen for **InstanceToBeTerminated** events. If you receive a notification that a preemptible instance is about to expire or be released, you can perform a graceful shutdown for the instance and then perform custom operations on the instance. For more information about how to check whether a preemptible instance is about to expire, see Check whether a preemptible instance is about to expire.

Related information

- Preemptible instances
- Set the ratio of preemptible instances to pay-as-you-go instances
- Manage node pools
- Check whether a preemptible instance is about to expire

3.Network

3.1. Use AES to manage Ingresses

Ambassador Edge Stack (AES) is built on Envoy Proxy. It can function as a high-performance API gateway and Ingress controller for Kubernetes clusters. AES functions as a control plane that translates Kubernetes custom resources into Envoy configurations for Envoy to handle requests. AES is integrated with rate limiting, request identification, load balancing, and observability. This topic describes how to use AES to manage Ingresses.

Prerequisites

- Create an ACK managed cluster.
- Install and set up kubectl.

Deploy AES in your Kubernetes cluster

By default, Container Service for Kubernetes does not deploy AES in Kubernetes clusters. You must manually deploy AES in Kubernetes clusters. The following example shows how to deploy AES in a Kubernetes clusters by using a YAML file. For more information about deployment methods, see the AES documents.

1. Run the following commands to deploy AES in a Kubernetes cluster:

```
kubectl apply -f https://www.getambassador.io/yaml/aes-crds.yaml && \
kubectl wait --for condition=established --timeout=90s crd -lproduct=aes && \
kubectl apply -f https://www.getambassador.io/yaml/aes.yaml && \
kubectl -n ambassador wait --for condition=available --timeout=90s deploy -lproduct=aes
```

The following output indicates that AES is deployed:

```
customresourcedefinition.apiextensions.k8s.io/authservices.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/consulresolvers.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/hosts.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/kubernetesendpointresolvers.getambassador
.io created
customresourcedefinition.apiextensions.k8s.io/kubernetesserviceresolvers.getambassador.
io created
customresourcedefinition.apiextensions.k8s.io/logservices.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/mappings.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/modules.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/ratelimitservices.getambassador.io create
d
customresourcedefinition.apiextensions.k8s.io/tcpmappings.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/tlscontexts.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/tracingservices.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/filterpolicies.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/filters.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/ratelimits.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/projectcontrollers.getambassador.io creat
ed
customresourcedefinition.apiextensions.k8s.io/projects.getambassador.io created
customresourcedefinition.apiextensions.k8s.io/projectrevisions.getambassador.io created
```

```
customresourcedefinition.apiextensions.k8s.io/authservices.getambassador.io condition m
et
customresourcedefinition.apiextensions.k8s.io/consulresolvers.getambassador.io conditio
n met
customresourcedefinition.apiextensions.k8s.io/filterpolicies.getambassador.io condition
met
customresourcedefinition.apiextensions.k8s.io/filters.getambassador.io condition met
customresourcedefinition.apiextensions.k8s.io/hosts.getambassador.io condition met
customresourcedefinition.apiextensions.k8s.io/kubernetesendpointresolvers.getambassador
.io condition met
customresourcedefinition.apiextensions.k8s.io/kubernetesserviceresolvers.getambassador.
io condition met
customresourcedefinition.apiextensions.k8s.io/logservices.getambassador.io condition me
t
customresourcedefinition.apiextensions.k8s.io/mappings.getambassador.io condition met
customresourcedefinition.apiextensions.k8s.io/modules.getambassador.io condition met
customresourcedefinition.apiextensions.k8s.io/projectcontrollers.getambassador.io condi
tion met
customresourcedefinition.apiextensions.k8s.io/projectrevisions.getambassador.io conditi
on met
customresourcedefinition.apiextensions.k8s.io/projects.getambassador.io condition met
customresourcedefinition.apiextensions.k8s.io/ratelimits.getambassador.io condition met
customresourcedefinition.apiextensions.k8s.io/ratelimitservices.getambassador.io condit
ion met
customresourcedefinition.apiextensions.k8s.io/tcpmappings.getambassador.io condition me
t
customresourcedefinition.apiextensions.k8s.io/tlscontexts.getambassador.io condition me
t
customresourcedefinition.apiextensions.k8s.io/tracingservices.getambassador.io conditio
n met
Warning: kubectl apply should be used on resource created by either kubectl create --sa
ve-config or kubectl apply
namespace/ambassador configured
serviceaccount/ambassador created
clusterrole.rbac.authorization.k8s.io/ambassador created
clusterrolebinding.rbac.authorization.k8s.io/ambassador created
clusterrole.rbac.authorization.k8s.io/ambassador-projects created
clusterrolebinding.rbac.authorization.k8s.io/ambassador-projects created
service/ambassador-redis created
deployment.apps/ambassador-redis created
ratelimitservice.getambassador.io/ambassador-edge-stack-ratelimit created
authservice.getambassador.io/ambassador-edge-stack-auth created
secret/ambassador-edge-stack created
mapping.getambassador.io/ambassador-devportal created
mapping.getambassador.io/ambassador-devportal-api created
service/ambassador created
service/ambassador-admin created
deployment.apps/ambassador created
deployment.extensions/ambassador condition met
deployment.extensions/ambassador-redis condition met
```

Verify that AES functions as an Ingress controller

To verify that AES functions as an Ingress controller, you must create a Deployment.

1. Use the following template to create a Deployment:

```
cat <<-EOF | kubectl apply -f -
apiVersion: v1
kind: Service
  metadata:
    name: quote
spec:
  ports:
  - name: http
    port: 80
    targetPort: 8080
  selector:
    app: quote
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quote
spec:
  replicas: 1
  selector:
    matchLabels:
      app: quote
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: quote
    spec:
      containers:
      - name: backend
        image: quay.io/datawire/quote:0.3.0
        ports:
        - name: http
          containerPort: 8080
EOF
```

2. Use the following template to create an Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: ambassador
  name: test-ingress
spec:
  rules:
  - http:
      paths:
      - path: /backend/
        backend:
          service:
            name: quote
            port:
              number: 80
        pathType: ImplementationSpecific
EOF
```

The following Mapping resource is equivalent to the preceding Ingress resource. For more information about how to use custom resource definitions (CRDs) in AES, see the AES documents.

```
cat > quote-backend.yaml <<-EOF
apiVersion: getambassador.io/v2
kind: Mapping
metadata:
    name: backend
spec:
    prefix: /backend/
    service: quote
EOF
```

3. Run the following command to query the IP address of the cluster:

```
kubectl get -n ambassador service ambassador -o "go-template={{range .status.loadBalanc
er.ingress}}{{or .ip .hostname}}{{end}}"
```

4. Run the following command to verify that AES functions as an Ingress controller:

```
curl -k https://{{AMBASSADOR_IP}}/backend/    # Replace the value of AMBASSADOR_IP with
the cluster IP address that you have obtained.
```

The following output indicates that AES functions as an Ingress controller:

```
  {
    "server": "icky-grapefruit-xar5if66",
    "quote": "A small mercy is nothing at all?",
    "time": "2020-07-17T09:00:57.646315605Z"
  }
```

3.2. Use Ambassador to expose an application API

Ambassador Edge Stack (AES) includes a high-performance Ingress controller and a high-performance API gateway that are built on top of Envoy Proxy. AES allows you to use various features of Envoy with Custom Resource Definitions (CRDs). These features include rate limiting, identity verification, load balancing, and observability. This topic describes how to use AES to expose the API of an application that is deployed in a Container Service for Kubernetes (ACK) cluster.

Prerequisites

- Create an ACK managed cluster
- Install and set up kubectl

Step 1: Deploy AES

1. Deploy AES.

```
kubectl apply -f https://www.getambassador.io/yaml/aes-crds.yaml
kubectl wait --for condition=established --timeout=90s crd -lproduct=aes
kubectl apply -f https://www.getambassador.io/yaml/aes.yaml
kubectl -n ambassador wait --for condition=available --timeout=90s deploy -lproduct=aes
```

2. Verify that AES is deployed.

```
kubectl get pod -n ambassador
```

Expected output:

```
NAME                                READY   STATUS    RESTARTS   AGE
ambassador-566d496b77-tg6ng         1/1     Running   0          2m
ambassador-redis-5fbd59f4fb-88gm8   1/1     Running   0          2m
```

If the pod where Ambassador is deployed is in the Running state, AES is deployed.

Step 2: Expose the API of an application

1. Create an application and a Service.

```
cat <<-EOF | kubectl apply -f -
---
apiVersion: v1
kind: Service
metadata:
  name: quote
  namespace: ambassador
spec:
  ports:
  - name: http
    port: 80
    targetPort: 8080
  selector:
    app: quote
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quote
  namespace: ambassador
spec:
  replicas: 1
  selector:
    matchLabels:
      app: quote
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: quote
    spec:
      containers:
      - name: backend
        image: docker.io/datawire/quote:0.4.1
        ports:
        - name: http
          containerPort: 8080
EOF
```

2. Verify that the application and Service are created.
   - Verify that the application is deployed.

     ```
     kubectl get pod -n ambassador -l app=quote
     ```

     Expected output:

     ```
     NAME                     READY   STATUS    RESTARTS   AGE
     quote-d57b799b4-****   1/1     Running   0          6m29s
     ```

     If the pod where the application is deployed is in the Running state, the application is deployed.
   - Verify that the Service is deployed.

```
kubectl get svc -n ambassador quote
```

Expected output:

```
NAME    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
quote   ClusterIP   172.23.XX.XX    <none>        80/TCP     5m45s
```

If the output displays the name of the Service, the Service is deployed.

3. Create a Mapping CRD to expose the application API.

```
cat <<-EOF | kubectl apply -f -
apiVersion: getambassador.io/v2
kind: Mapping
metadata:
    name: backend
    namespace: ambassador
spec:
    prefix: /backend/
    service: quote
EOF
```

4. Verify that the Mapping CRD is created.

```
kubectl get mappings -n ambassador
```

Expected output:

```
NAME                        PREFIX      SERVICE          STATE   REASON
ambassador-devportal        /docs/      12*. *. *.*:8500
ambassador-devportal-api    /openapi/   12*. *. *.*:8500
quote-backend               /backend/   quote
```

If the output displays the Mapping CRD, the Mapping CRD is deployed.

5. Obtain the service IP address of AES.

```
kubectl get -n ambassador service ambassador -o "go-template={{range .status.loadBalanc
er.ingress}}{{or .ip .hostname}}{{end}}"
```

Expected output:

```
47.94.**. **
```

6. Call the exposed application API.

```
curl -k https://{{IP}}/backend/
{
  "server": "frosty-kiwi-ewe7vk96",
  "quote": "A principal idea is omnipresent, much like candy.",
  "time": "2020-08-01T08:41:37.054259819Z"
}
```

Replace `${IP}` with the service IP address of AES that is obtained in Step .

Related information

- Ambassador Edge Stack
-

3.3. Improve the performance of the NetworkPolicy feature for a large ACK cluster in Terway mode

In a Container Service for Kubernetes (ACK) cluster that has the network plug-in Terway installed, you can use the NetworkPolicy feature to control communication among pods. When an ACK cluster that has Terway installed contains more than 100 nodes, the NetworkPolicy proxies cause a heavy load on the management of the cluster. To resolve this issue, you must optimize the NetworkPolicy feature for the cluster. This topic describes how to optimize the performance of the NetworkPolicy feature for a large ACK cluster.

Prerequisites

- An ACK cluster that has Terway installed and contains more than 100 nodes is created. For more information, see Create an ACK managed cluster.
- Connect to ACK clusters by using kubectl.

Context

Terway implements the NetworkPolicy feature by using the Felix agent of Calico. In an ACK cluster that contains more than 100 nodes, the Felix agent on each node retrieves proxy rules from the API server. This increases the load of the API server. To reduce the load of the API server, you can disable the NetworkPolicy feature or deploy the Typha component as a repeater.

You can improve the performance of the NetworkPolicy feature for a large ACK cluster in the following ways:

- Deploy Typha as a repeater.
- Disable the NetworkPolicy feature.

> **Note** After you disable the NetworkPolicy feature, you cannot use network policies to control communication among pods.

Deploy Typha as a repeater

1.
2. Upgrade Terway to the latest version. For more information, see Manage system components.
3. Use the following template and run the `kubectl apply -f` command to deploy Typha as a repeater.

```
apiVersion: v1
kind: Service
metadata:
  name: calico-typha
  namespace: kube-system
  labels:
    k8s-app: calico-typha
```

```
spec:
  ports:
    - port: 5473
      protocol: TCP
      targetPort: calico-typha
      name: calico-typha
  selector:
    k8s-app: calico-typha
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: calico-typha
  namespace: kube-system
  labels:
    k8s-app: calico-typha
spec:
  replicas: 3
  revisionHistoryLimit: 2
  selector:
    matchLabels:
      k8s-app: calico-typha
  template:
    metadata:
      labels:
        k8s-app: calico-typha
      annotations:
        cluster-autoscaler.kubernetes.io/safe-to-evict: 'true'
    spec:
      nodeSelector:
        kubernetes.io/os: linux
      hostNetwork: true
      tolerations:
        - operator: Exists
      serviceAccountName: terway
      priorityClassName: system-cluster-critical
      containers:
      - image: registry-vpc.{REGION-ID}.aliyuncs.com/acs/typha:v3.20.2
        name: calico-typha
        ports:
        - containerPort: 5473
          name: calico-typha
          protocol: TCP
        env:
          - name: TYPHA_LOGSEVERITYSCREEN
            value: "info"
          - name: TYPHA_LOGFILEPATH
            value: "none"
          - name: TYPHA_LOGSEVERITYSYS
            value: "none"
          - name: TYPHA_CONNECTIONREBALANCINGMODE
            value: "kubernetes"
          - name: TYPHA_DATASTORETYPE
            value: "kubernetes"
```

```
          - name: TYPHA_HEALTHENABLED
            value: "true"
        livenessProbe:
          httpGet:
            path: /liveness
            port: 9098
            host: localhost
          periodSeconds: 30
          initialDelaySeconds: 30
        readinessProbe:
          httpGet:
            path: /readiness
            port: 9098
            host: localhost
          periodSeconds: 10
---
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: calico-typha
  namespace: kube-system
  labels:
    k8s-app: calico-typha
spec:
  maxUnavailable: 1
  selector:
    matchLabels:
      k8s-app: calico-typha
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: bgppeers.crd.projectcalico.org
spec:
  scope: Cluster
  group: crd.projectcalico.org
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          apiVersion:
            type: string
  names:
    kind: BGPPeer
    plural: bgppeers
    singular: bgppeer
```

> Note
>
> - Replace the value of the {REGION-ID} field with the specified region ID.
> - Modify the value of the replicas field based on the cluster size. Create 1 replica for every 200 nodes. You must create at least 3 replicas.

4. Run the following command to add the felix_relay_service: calico-typha setting to the ConfigMap of the Terway plug-in:

```
kubectl edit cm eni-config -n kube-system
#Add the following trunk configuration. The configuration must be aligned with the eni_
conf parameter.
felix_relay_service: calico-typha
```

5. Run the following command to restart Terway:

```
kubectl get pod -n kube-system  | grep terway | awk '{print $1}' | xargs kubectl delete
-n kube-system pod
```

```
#Output:
pod "terway-eniip-8hmz7" deleted
pod "terway-eniip-dclfn" deleted
pod "terway-eniip-rmctm" deleted
```

Disable the NetworkPolicy feature

If you no longer need to use network policies, you can disable the NetworkPolicy feature to reduce the heavy load on the API server. The heavy load is caused by the NetworkPolicy proxies.

1. Add disable_network_policy: "true" to ConfigMap of the Terway plug-in to disable the NetworkPolicy feature.

```
kubectl edit cm -n kube-system eni-config
#Add or modify (if this key exists) the following setting:
disable_network_policy: "true"
```

2. Run the following command to restart Terway:

```
kubectl get pod -n kube-system  | grep terway | awk '{print $1}' | xargs kubectl delete
-n kube-system pod
```

```
#Output:
pod "terway-eniip-8hmz7" deleted
pod "terway-eniip-dclfn" deleted
pod "terway-eniip-rmctm" deleted
```

Result

After the preceding operations are complete, the NetworkPolicy proxies start to use the Typha component. This reduces the load on the API server. You can monitor the traffic that is distributed to the Server Load Balancer (SLB) instances to check whether the load on the API server is reduced.

3.4. Best practices for DNS services

The Domain Name System (DNS) service is a basic service for Kubernetes clusters. If the DNS settings on your client are not properly configured or if you use a large cluster, DNS resolution timeouts and failures may occur. This topic describes best practices for configuring DNS services in Kubernetes clusters to help you avoid the issues.

Prerequisites

- Create an ACK managed cluster
- Connect to ACK clusters by using kubectl

Context

The best practices in this topic involve clients and DNS servers:

- Clients: You can optimize DNS queries submitted by clients to reduce resolution latency. You can also reduce DNS resolution errors by using proper container images, node operating systems, and NodeLocal DNSCache.
  - Optimize DNS queries
  - Use proper container images
  - Reduce the adverse effect of occasional DNS resolution timeouts caused by IPVS defects
  - Use NodeLocal DNSCache to optimize DNS resolution
  - Use proper CoreDNS versions
- CoreDNS servers: You can identify DNS errors by monitoring the status of CoreDNS. This helps you quickly locate the causes. You can modify the CoreDNS Deployment to improve the availability of CoreDNS and increase the queries per second (QPS).
  - Monitor the status of CoreDNS
    - Monitoring metrics
    - Operational Log
  - Modify the CoreDNS Deployment
    - Modify the number of CoreDNS pods
    - Schedule CoreDNS pods to proper nodes
    - Manually increase the number of CoreDNS pods
    - Use cluster-autoscaler to automatically increase the number of CoreDNS pods
    - Use HPA to increase the number of CoreDNS pods based on CPU utilization
  - Properly configure CoreDNS
    - Disable the affinity settings for the kube-dns Service
    - Disable the autopath plug-in
    - Configure graceful shutdown for CoreDNS
    - Configure the default protocol for the forward plug-in and upstream DNS servers of VPC

For more information about CoreDNS, see Official Documentation of CoreDNS.

Optimize DNS queries

DNS queries are frequently submitted in Kubernetes. A large number of DNS queries can be optimized or avoided. You can optimize DNS queries by using one of the following methods:

- Use a connection pool: If a client pod frequently accesses a Service, we recommend that you use a connection pool. A connection pool can cache connections to the upstream Service in the memory. This way, the client pod no longer needs to send a DNS query and establish a TCP connection each time it accesses the Service.
- Use DNS caching:
  - If you cannot use a connection pool to connect a client pod to a Service, we recommend that you cache DNS resolution results on the client pod side. For more information, see Use NodeLocal DNSCache to optimize DNS resolution.
  - If you cannot use NodeLocal DNSCache, you can use the built-in Name Service Cache Daemon (NSCD) in containers. For more information about how to use NSCD, see Use NSCD in Kubernetes clusters.
- Optimize the *resolv.conf* file: Due to the use of the ndots and search parameters in the *resolv.conf* file, the DNS resolution efficiency is affected by how you specify domain names in containers. For more information about the ndots and search parameters, see Configure DNS resolution.
- Optimize domain name settings: You can specify the domain name that a client pod needs to access based on the following rules. These rules can help minimize the number of attempts for resolving the domain name and make the DNS resolution service more efficient.
  - If the client pod needs to access a Service in the same namespace, use `<service-name>` as the domain name. `<service-name>` indicates the name of the Service.
  - If the client pod needs to access a Service in another namespace, use `<service-name>.<namespace-name>` as the domain name. `namespace-name` specifies the namespace to which the Service belongs.
  - If the client pod needs to access an external domain name, you can specify the domain name in the Fully Qualified Domain Name (FQDN) format by appending a period (.) to the domain name. This avoids invalid DNS lookups caused by `search` domains. For example, if the client pod needs to access *www.aliyun.com*, you can specify the domain name as *www.aliyun.com.*.

Use proper container images

The implementation of the built-in musl libc in Alpine container images is different from that of glibc:

- Alpine 3.3 and earlier versions do not support the search parameter which allows you to specify search domains. As a result, Services cannot be discovered.
- musl libc processes queries that are sent to the DNS servers that are specified in the */etc/resolv.conf* file in parallel. As a result, NodeLocal DNSCache fails to optimize DNS resolution.
- musl libc processes A and AAAA queries that use the same socket in parallel. This causes packet loss on the conntrack port in earlier kernel versions.

For more information, see musl libc.

If containers that are deployed in a Kubernetes cluster use Alpine as the base image, domain names may not be resolved due to the use of musl libc. We recommend that you replace the image with an image that is based on Debian or CentOS.

Reduce the adverse effect of occasional DNS resolution timeouts caused by IPVS defects

If the load balancing mode of kube-proxy is set to IPVS in your cluster, DNS resolution timeouts may occur when CoreDNS pods are scaled in or restarted. The issues are caused by the kernel bugs of Linux. For more information, see IPVS.

You can use the following methods to reduce the adverse effect of this issue:

- Use NodeLocal DNSCache to optimize DNS resolution. For more information, see Use NodeLocal DNSCache to optimize DNS resolution.
- Modify the timeout period for IPVS UDP sessions in the kube-proxy configuration. For more information, see How do I modify the timeout period for IPVS UDP sessions in the kube-proxy configuration?.

Use NodeLocal DNSCache to optimize DNS resolution

Container Service for Kubernetes (ACK) allows you to deploy NodeLocal DNSCache to improve the stability and performance of service discovery. NodeLocal DNSCache is implemented as a DaemonSet and runs a DNS caching agent on cluster nodes to improve the efficiency of DNS resolution for ACK clusters.

For more information about NodeLocal DNSCache and how to deploy NodeLocal DNSCache in ACK clusters, see Configure NodeLocal DNSCache.

Use proper CoreDNS versions

CoreDNS is backward-compatible with Kubernetes. We recommend that you use a new stable version of CoreDNS. You can install, upgrade, and CoreDNS on the Add-ons page of the ACK console. If the status of the CoreDNS component indicates that CoreDNS is upgradable, we recommend that you upgrade the component during off-peak hours at the earliest opportunity.

- For more information about how to upgrade CoreDNS, see Configure ACK to automatically update CoreDNS.
- For more information about the release notes for CoreDNS, see CoreDNS.

The following issues may occur in CoreDNS versions earlier than 1.7.0:

- If connectivity exceptions occur between CoreDNS and the API server, such as network jitters, API server restarts, or API server migrations, CoreDNS pods may be restarted because error logs cannot be written. For more information, see Set klog's logtostderr flag.
- CoreDNS occupies extra memory resources during the initialization process. In this process, the default memory limit may cause out of memory (OOM) errors in large clusters. If this situation intensifies, CoreDNS pods may be repetitively restarted but fail to be started. For more information, see CoreDNS uses a lot memory during initialization phase.
- CoreDNS has issues that may affect the domain name resolution of headless Services and requests from outside the cluster. For more information, see plugin/kubernetes: handle tombstones in default processor and Data is not synced when CoreDNS reconnects to kubernetes api server after protracted disconnection.
- Some earlier CoreDNS versions are configured with default toleration rules that may cause CoreDNS pods to be deployed on abnormal nodes and fail to be automatically evicted when exceptions occur on the nodes. This may lead to domain name resolution errors in the cluster.

The following table describes the recommended CoreDNS versions for clusters that run different Kubernetes versions.

Kubernetes	CoreDNS
Earlier than 1.14.8 (discontinued)	v1.6.2
1.14.8 and later but earlier than 1.20.4	v1.7.0.0-f59c03d-aliyun

Kubernetes	CoreDNS
1.20.4 and later	v1.8.4.1-3a376cc-aliyun

Note Kubernetes versions earlier than 1.14.8 are discontinued. We recommend that you upgrade the Kubernetes version before you upgrade CoreDNS.

Monitor the status of CoreDNS

CoreDNS uses the standard Prometheus API to collect metrics such as DNS resolution results. This allows you to identify exceptions in CoreDNS and upstream DNS servers at the earliest opportunity.

By default, monitoring metrics and alerting rules related to CoreDNS are predefined in Application Real-Time Monitoring Service (ARMS) Prometheus provided by Alibaba Cloud. You can log on to the to enable Prometheus and dashboards. For more information, see Enable ARMS Prometheus.

If you use open source Prometheus to monitor the Kubernetes cluster, you can view the related metrics in Prometheus and create alert rules based on the following key metrics. For more information, see CoreDNS Prometheus official documentation. The following table describes the key metrics.

Metric	Description	Alert setting
coredns_dns_requests_total	The number of requests.	You can create alert rules based on the number of requests. This helps you check whether the current DNS QPS is high.
coredns_dns_responses_total	The number of responses.	You can create alert rules based on the number of responses with different response codes (RCODE). For example, you can create alert rules based on the SERVFAIL error.
coredns_panics_total	The number of abnormal exits of CoreDNS.	You can create alert rules based on the number of abnormal exits of CoreDNS. If the value of this metric is greater than 0, abnormal exits occur and alerts are triggered.
coredns_dns_request_duration_seconds	Duration to process a DNS query.	Alerts are triggered when the duration to process a DNS query exceeds the specified threshold.

When DNS resolution errors occur, you can view the log of CoreDNS to identify the causes. We recommend that you enable logging for CoreDNS and use Log Service to collect the log data. For more information, see Monitor CoreDNS and analyze the CoreDNS log.

Metrics

Operational log

Modify the CoreDNS Deployment

You must deploy CoreDNS in Kubernetes clusters. By default, CoreDNS and your client pod run on the same cluster node. Take note of the following items:

- Modify the number of CoreDNS pods
- Schedule CoreDNS pods to proper nodes
- Manually increase the number of CoreDNS pods
- Use cluster-autoscaler to automatically increase the number of CoreDNS pods
- Use HPA to increase the number of CoreDNS pods based on CPU utilization

We recommend that you provision at least two CoreDNS pods. You must make sure that the number of CoreDNS pods is sufficient to handle DNS queries within the cluster.

The DNS QPS of CoreDNS is related to CPU usage. A single CPU can handle more than 10,000 DNS QPS if you enable DNS caching. The DNS QPS required by different workloads may vary. You can evaluate the DNS QPS based on the peak CPU usage of each CoreDNS pod. We recommend that you increase the number of CoreDNS pods if a CoreDNS pod occupies more than one vCPU during peak hours. If you cannot confirm the peak CPU usage of each CoreDNS pod, you can set the ratio of CoreDNS pods to cluster nodes to 1:8. This way, each time you add eight nodes to a cluster, a CoreDNS pod is created. The total number of CoreDNS pods must not exceed 10. If your cluster contains more than 100 nodes, we recommend that you use NodeLocal DNSCache. For more information, see Use NodeLocal DNSCache to optimize DNS resolution.

- You can manually increase the number of CoreDNS pods if the number of cluster nodes remains unchanged for a long period of time. For more information, see Manually increase the number of CoreDNS pods.
- You can set the system to automatically increase the number of CoreDNS pods if the number of cluster nodes increases. For more information, see Use cluster-autoscaler to automatically increase the number of CoreDNS pods.

> **Note** UDP does not support retransmission. When CoreDNS pods terminate, UDP packets may be dropped if a CoreDNS pod is deleted or restarted. As a result, the cluster may experience DNS query timeouts or failures. If UDP packet loss caused by IPVS issues occurs on cluster nodes, the cluster may experience DNS query timeouts or failures that last for 5 minutes after a CoreDNS pod is deleted or restarted. For more information about how to resolve DNS query failures caused by IPVS issues, see What do I do if DNS resolutions fail due to IP Virtual Server (IPVS) errors?.

When you deploy CoreDNS pods in a cluster, we recommend that you deploy the CoreDNS pods on different cluster nodes across multiple zones. This prevents service disruptions when a single node or zone fails. By default, soft anti-affinity settings based on nodes are configured for CoreDNS. Some or all CoreDNS pods may be deployed on the same node due to insufficient nodes. In this case, we recommend that you delete the CoreDNS pods and reschedule the pods.

CoreDNS pods must not be deployed on cluster nodes whose CPU and memory resources are fully utilized. Otherwise, DNS QPS and response time are adversely affected.

If the number of cluster nodes remains unchanged for a long period of time, you can run the following command to increase the number of CoreDNS pods:

```
kubectl scale --replicas={target} deployment/coredns -n kube-system
```

> **Note** Replace `{target}` with the required value.

If the number of cluster nodes increases, you can use the following YAML template to deploy cluster-proportional-autoscaler and dynamically increase the number of CoreDNS pods:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: dns-autoscaler
  namespace: kube-system
  labels:
    k8s-app: dns-autoscaler
spec:
  selector:
    matchLabels:
      k8s-app: dns-autoscaler
  template:
    metadata:
      labels:
        k8s-app: dns-autoscaler
    spec:
      serviceAccountName: admin
      containers:
      - name: autoscaler
        image: registry.cn-hangzhou.aliyuncs.com/acs/cluster-proportional-autoscaler:1.8.4
        resources:
          requests:
            cpu: "200m"
            memory: "150Mi"
        command:
        - /cluster-proportional-autoscaler
        - --namespace=kube-system
        - --configmap=dns-autoscaler
        - --nodelabels=type!=virtual-kubelet
        - --target=Deployment/coredns
        - --default-params={"linear":{"coresPerReplica":64,"nodesPerReplica":8,"min":2,"max
":100,"preventSinglePointFailure":true}}
        - --logtostderr=true
        - --v=9
```

In the preceding example, a linear scaling policy is used. The number of CoreDNS pods is calculated based on the following formula: Replicas (pods) = Max (Ceil (Cores × 1/coresPerReplica), Ceil (Nodes × 1/nodesPerReplica)). The number of CoreDNS pods is subject to the values of `max` and `min` in the linear scaling policy. The following code block shows the parameters of the linear scaling policy:

```
{
      "coresPerReplica": 64,
      "nodesPerReplica": 8,
      "min": 2,
      "max": 100,
      "preventSinglePointFailure": true
}
```

Horizontal Pod Autoscaler (HPA) frequently triggers scale-in activities for CoreDNS pods. We recommend that you do not use HPA. If HPA is required in specific scenarios, you can refer to the following policy configurations based on CPU utilization:

```
---
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: coredns-hpa
  namespace: kube-system
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: coredns
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50
```

Note For more information about how to use HPA, see HPA.

Modify the number of CoreDNS pods

Schedule CoreDNS pods to proper nodes

Manually increase the number of CoreDNS pods

Dynamically increase the number of CoreDNS pods by using cluster-autoscaler

Use HPA to increase the number of CoreDNS pods based on CPU utilization

Properly configure CoreDNS

ACK provides only the default settings for CoreDNS. You can modify the parameters to optimize the settings and enable CoreDNS to provide normal DNS services for your client pods. You can modify the configurations of CoreDNS on demand. For more information, see Configure DNS resolution and CoreDNS official documentation.

The default configurations of earlier CoreDNS versions that are deployed together with your Kubernetes clusters may pose risks. We recommend that you check and optimize the configurations by using the following methods:

- Disable the affinity settings for the kube-dns Service
- Disable the autopath plug-in
- Configure graceful shutdown for CoreDNS
- Configure the default protocol for the forward plug-in and upstream DNS servers of VPC

The affinity settings may cause CoreDNS pods to handle different loads. To disable the affinity settings, perform the following steps:

- **Use the ACK console**
  i.
  ii. In the left-side navigation pane, click **Clusters**.
  iii. On the **Clusters** page, find the cluster that you want to manage and click its name or click **Details** in the **Actions** column.
  iv. In the left-side navigation pane of the details page, choose **Network > Services**.
  v. In the kube-system namespace, find the **kube-dns** Service and click **View in YAML** in the Actions column.
     - If the value of the sessionAffinity field is `None`, skip the following steps.
     - If the value of the sessionAffinity field is `ClientIP`, perform the following steps.
  vi. Delete sessionAffinity, sessionAffinityConfig, and all of the subfields. Then, click **Update**.

```
#Delete the following content.
sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
```

  vii. Find the **kube-dns** Service and click **View in YAML** in the Actions column again to check whether the value of the sessionAffinity field is `None`. If the value is `None`, the kube-dns Service is modified.

- **Use the CLI**
  i. Run the following command to query the configurations of the kube-dns Service:

```
kubectl -n kube-system get svc kube-dns -o yaml
```

     - If the value of the sessionAffinity field is `None`, skip the following steps.
     - If the value of the sessionAffinity field is `ClientIP`, perform the following steps.
  ii. Run the following command to modify the kube-dns Service:

```
kubectl -n kube-system edit service kube-dns
```

  iii. Delete all of the fields that are related to sessionAffinity, including sessionAffinity, sessionAffinityConfig, and all of the subfields. Then, save the change and exit.

```
#Delete the following content.
sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
```

iv. After you modify the kube-dns Service, run the following command again to check whether the value of the sessionAffinity field is `None`. If the value is `None`, the kube-dns Service is modified.

```
kubectl -n kube-system get svc kube-dns -o yaml
```

The autopath plug-in is enabled for CoreDNS of earlier versions and may cause DNS resolution errors in specific scenarios. If the autopath plug-in is enabled, you must disable the plug-in in the coredns ConfigMap. For more information, see #3765.

> **Note** After you disable the autopath plug-in, the number of DNS queries sent from the client per second is increased by three times at most. Therefore, the amount of time required to resolve a domain name is also increased by three times at most. You must pay close attention to the load on CoreDNS and the impacts on your business.

1. Run the `kubectl -n kube-system edit configmap coredns` command to modify the coredns ConfigMap.
2. Delete `autopath @kubernetes`. Then, save the change and exit.
3. Check the status and logs of the CoreDNS pods. If the log data contains the `reload` keyword, the new configuration is loaded.

> **Note** ACK may consume additional memory resources when it updates the coredns ConfigMap. After you modify the coredns ConfigMap, check the status of the CoreDNS pods. If the memory resources of the pods are exhausted, change the memory limit of pods in the CoreDNS Deployment. We recommend that you change the memory limit to 2 GB.

- **Use the ACK console**
  i.
  ii. In the left-side navigation pane, click **Clusters**.
  iii. On the **Clusters** page, find the cluster that you want to manage and click its name or click **Details** in the **Actions** column.
  iv. In the left-side navigation pane of the details page, choose **Configurations > ConfigMaps**.
  v. Select the kube-system namespace. Find the *coredns* ConfigMap and click **Edit YAML** in the Actions column.
  vi. Refer to the following YAML content and make sure that the health plug-in is enabled. Then, set lameduck to `15s` and click **OK**.

```
.:53 {
        errors
        #The setting of the health plug-in may vary based on the CoreDNS version.
        #Scenario 1: The health plug-in is disabled by default.
        #Scenario 2: The health plug-in is enabled by default but lameduck is not set
.
        health
        #Scenario 3: The health plug-in is enabled by default and lameduck is set to
5s.
        health {
            lameduck 5s
        }
        #In the preceding scenarios, change the value of lameduck to 15s.
        health {
            lameduck 15s
        }
        #You do not need to modify other plug-ins.
    }
```

If the CoreDNS pods run as normal, CoreDNS can be gracefully shut down. If the CoreDNS pods do not run as normal, you can check the pod events and log to identify the cause.

- **Use the CLI**
  i. Run the following command to open the coredns ConfigMap:

  ```
  kubectl -n kube-system edit configmap coredns
  ```

  ii. Refer to the following YAML content and make sure that the `health` plug-in is enabled. Then, set lameduck to `15s`.

  ```
  .:53 {
          errors
          #The setting of the health plug-in may vary based on the CoreDNS version.
          #Scenario 1: The health plug-in is disabled by default.
          # Scenario 2: The health plug-in is enabled by default but lameduck is not se
  t.
          health
          #Scenario 3: The health plug-in is enabled by default and lameduck is set to
  5s.
          health {
              lameduck 5s
          }
          #In the preceding scenarios, change the value of lameduck to 15s.
          health {
              lameduck 15s
          }
          #You do not need to modify other plug-ins.
      }
  ```

  iii. After you modify the coredns ConfigMap, save the change and exit.

  If the CoreDNS pods run as normal, CoreDNS can be gracefully shut down. If the CoreDNS pods do not run as normal, you can check the pod events and log to identify the cause.

NodeLocal DNSCache uses the TCP protocol to communicate with CoreDNS. CoreDNS communicates with the upstream DNS servers based on the protocol used by the source of DNS queries. Therefore, DNS queries sent from a client pod for external domain names pass through NodeLocal DNSCache and CoreDNS, and then arrive at the DNS servers in the VPC over TCP. The IP addresses of the DNS servers are 100.100.2.136 and 100.100.2.13. These IP addresses are automatically configured on the Elastic Compute Service (ECS) instances.

DNS servers in a VPC have limited support for TCP. If you use NodeLocal DNSCache, you must modify the configurations of CoreDNS and enable CoreDNS to use UDP for communication with the upstream DNS servers. This prevents DNS resolution issues. We recommend that you modify the ConfigMap of CoreDNS based on the following modifications. The ConfigMap is named *coredns* and belongs to the kube-system namespace. Modify the setting of the `forward` plug-in and set the protocol that is used to request upstream servers to `prefer_udp`. This way, CoreDNS preferentially uses the UDP protocol to communicate with the upstream DNS servers. You can modify the setting based on the following modifications:

```
#The original settings.
forward . /etc/resolv.conf
#The modified settings.
forward . /etc/resolv.conf {
  prefer_udp
}
```

Disable the autopath plug-in

Configure graceful shutdown for CoreDNS

Configure the default protocol for the forward plug-in and upstream DNS servers of VPC

Related information

- DNS overview
- Configure DNS resolution
- Configure NodeLocal DNSCache

3.5. Best practices for the NGINX Ingress controller

The NGINX Ingress controller is deployed in Container Service for Kubernetes (ACK) clusters and used to control Ingresses. The NGINX Ingress controller provides high performance and allows you to customize configurations. The NGINX Ingress controller provided by ACK is developed based on the open source version and is integrated with various features of Alibaba Cloud services. The NGINX Ingress controller provides a simplified user experience. The NGINX Ingress controller is deployed in ACK clusters. The stability of the NGINX Ingress controller is reliant on its configurations and the status of the cluster. This topic describes the best practices for configuring the NGINX Ingress controller that is deployed in an ACK cluster.

Context

The best practices for the NGINX Ingress controller include the following content:

Performance and stability of the NGINX Ingress controller

Specify a proper number of controller pods and configure proper resource limits

By default, two pods are provisioned if you install the NGINX Ingress controller from the Add-ons page or when you create a cluster. You can adjust the number of controller pods based on your business requirements.

When you deploy the NGINX Ingress controller, make sure that the controller pods are distributed across different nodes. This helps prevent resource contention among controller pods and single points of failure (SPOFs). You can schedule the controller pods to exclusive nodes to ensure the performance and stability of the NGINX Ingress controller. For more information, see Use exclusive nodes to ensure the performance and stability of the NGINX Ingress controller. We recommend that you do not set resource limits for the NGINX Ingress controller pods. This helps prevent service interruptions that are caused by out of memory (OOM) errors. If resource limits are required, we recommend that you set the CPU limit to 1,000 millicores or greater, and set the memory limit to 2 GiB or greater. The format of the CPU limit in the YAML file is `1000m`.

Use exclusive nodes to ensure the performance and stability of the NGINX Ingress controller

- If you want to ensure the high stability of the NGINX Ingress controller, you can schedule the controller pods to exclusive nodes. This helps prevent resource contention among pods. For more information, see Deploy Ingresses in a high-reliability architecture.
- You can also configure the NGINX Ingress controller for applications that need to handle heavy traffic. For more information, see Deploy Nginx Ingress controllers in high-load scenarios.

Optimize the performance of the NGINX Ingress controller

You can modify system settings and tune NGINX parameters to optimize the performance of the NGINX Ingress controller:

- Modify system settings: By default, some common settings of the operating systems provided by Alibaba Cloud are optimized. You can also modify other system settings, such as the backlog queue and the maximum range of ports that are available. After you modify the system settings, the NGINX

Ingress controller can process a large number of concurrent connections. This also prevents connection failures due to insufficient ports.

- Tune NGINX parameters:
  - Maximum number of connections that each worker process can handle: You can modify the maximum number of connections to each worker to ensure that the NGINX Ingress controller can process a large number of concurrent connections.
  - Keepalive requests: The NGINX Ingress controller sends requests to backend pods through keepalive connections. You can change the value of keepalive requests to a greater value. This way, keepalive connections can be persisted longer to forward more requests.
  - Keepalive timeout: Make sure that the timeout period of keepalive connections to the backend pods is not shorter than the timeout period of connections to the NGINX Ingress controller. By default, the timeout period of connections to the NGINX Ingress controller is 900 seconds for ACK clusters.

By default, the preceding parameters are optimized in the NGINX Ingress component. In most cases, the configurations of the NGINX Ingress controller can meet your business requirements. If you have other requirements, you can modify the system settings and NGINX configurations in the ConfigMap. For more information, see ConfigMaps.

Configure HPA to perform auto scaling based on loads

In most cases, the NGINX Ingress controller can handle traffic spikes. If the NGINX Ingress controller cannot meet your requirements in heavy load scenarios, you can configure Horizontal Pod Autoscaler (HPA) to scale out the Ingress controller pods. For more information, see HPA.

> **Notice** When HPA scales the controller pods, service interruptions may occur. Proceed with caution when you configure HPA.

The following YAML template provides an example on how to configure HPA:

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-ingress-controller-hpa
  namespace: kube-system
spec:
   scaleTargetRef:
     apiVersion: apps/v1
     kind: Deployment
     name: nginx-ingress-controller
   minReplicas: 2
   maxReplicas: 5
   metrics:
    - type: Resource
      resource:
        name: cpu
        targetAverageUtilization: 50
```

Improve the observability of the NGINX Ingress controller

Use Log Service and ARMS Prometheus to improve observability

The NGINX Ingress controller provides the Ingress dashboard based on Log Service and ARMS Prometheus, which helps you gain better insights into the traffic to your service.

- Log Service:
  - If you select Enable Log Service and Create Ingress Dashboard when you create a cluster, you can go to the and choose **Network > Ingresses**. On the Ingresses page, click **Ingress Overview** to view the Ingress dashboard provided by ACK based on Log Service. You can also choose **Operations > Log Center** to view the log that is generated by the NGINX Ingress controller. For more information, see Analyze and monitor the access log of nginx-ingress.
  - If you do not select Enable Log Service and Create Ingress Dashboard when you create a cluster, you can manually configure components and rules for log collection. For more information, see Analyze and monitor the access log of nginx-ingress. For more information about application monitoring, see Ingress Dashboard.
- ARMS Prometheus: You can install Application Real-Time Monitoring Service (ARMS) Prometheus when you create a cluster. You can also choose **Operations > Prometheus Monitoring** to install and access ARMS Prometheus after you create a cluster. For more information, see Enable ARMS Prometheus.

> **Note** If you use ARMS Prometheus, add the `host` field to the Ingresses that are created in the cluster. Otherwise, some Ingress metrics are not displayed by default. You can also add `--metrics-per-host=false` to the startup parameter of the `controller` in the Deployment of the NGINX Ingress controller to resolve the issue.

Advanced features of the NGINX Ingress controller

Use multiple NGINX Ingress controllers

You may want to deploy multiple NGINX Ingress controllers in a cluster to isolate the internal network from the Internet. In addition to the NGINX Ingress controller that can be installed from the Add-ons page of the , you can also deploy an NGINX Ingress controller on the Marketplace page. For more information, see Deploy multiple Ingress controllers in a cluster.

Access the NGINX Ingress controller from within the cluster

In most cases, requests to the public IP address of a LoadBalancer Service in an ACK cluster are blocked by iptables and IP Virtual Server (IPVS). The public IP address of the LoadBalancer Service is the same as that of the NGINX Ingress controller. If `externalTrafficPolicy` of the Service is set to `Local` and no NGINX Ingress controller pod is scheduled to the cluster node, connectivity issues occur. By default, the NGINX Ingress controller in ACK clusters uses the LoadBalancer Service in Local mode. Therefore, when you access the IP address of the Classic Load Balancer (CLB) instance that is associated with the NGINX Ingress controller from within the cluster, you may encounter connectivity issues.

If you want to use a public IP address or a domain name that is mapped to a public IP address to access the NGINX Ingress controller from within the cluster, we recommend that you use the cluster IP address of the LoadBalancer Service or the internal domain name `nginx-ingress-lb.kube-system`. If you access the NGINX Ingress controller from within the cluster, connectivity issues may occur due to the hairpin issue. For more information about how to resolve the issue, see What Can I Do if the Cluster Cannot Access the IP Address of the SLB Instance Exposed by the LoadBalancer Service.

Use WAF or transparent WAF

To block malicious requests, you can log on to the Web Application Firewall (WAF) console or the Application Load Balancer (ALB) console and enable WAF or transparent WAF for the CLB instance that is used by the NGINX Ingress controller. To enable WAF or transparent WAF on HTTPS ports, you must configure the required certificate in the console. The following issues may occur when you configure the certificate:

- Transport Layer Security (TLS) requests are blocked by WAF or transparent WAF. Therefore, the certificate that is configured in the Secret of the cluster is not exposed to the Internet.
- When you use the IP address of the CLB instance or the cluster IP address of the LoadBalancer Service to access port 443 from within the cluster, traffic may fail to pass through WAF or transparent WAF. As a result, an error occurs when the system returns the certificate.
- When WAF or transparent WAF is enabled, the NGINX Ingress controller cannot preserve client IP addresses by default. You can add the following content to the ConfigMap of the NGINX Ingress controller to enable the realip module of NGINX and use the `X-Forwarded-For` header to preserve client IP addresses. If you install the NGINX Ingress controller from the Add-ons page, the ConfigMap is named nginx-configuration and belongs to the kube-system namespace by default.

```
use-forwarded-headers: "true" # Use this option if the version of the NGINX Ingress contr
oller is 0.30.0 or earlier.
enable-real-ip: "true" # Use this option if the version of the NGINX Ingress controller i
s 0.44.0 or later.
proxy-real-ip-cidr: <The back-to-origin CIDR block that you obtain from WAF>
```

Use the NGINX Ingress controller to perform blue-green releases and canary releases

You can use the canary release feature provided by the NGINX Ingress controller in the or by adding annotations. For more information, see Use the NGINX Ingress controller to implement canary releases and blue-green releases.

> **Notice** Make sure that the old version and new version of the Service for which you want to perform canary releases are associated only with the canary Ingress. Otherwise, conflicts may occur in canary release rules and traffic is distributed to other Ingresses.

Use the NGINX Ingress controller as a proxy to distribute non-HTTP requests

By default, the NGINX Ingress controller connects to backend Services over HTTP. The NGINX Ingress controller supports the following protocols for connecting to backend Services: WebSocket, HTTPS, and gRPC. For more information about the protocols that are supported by the NGINX Ingress controller, see Backend Protocol.

- WebSocket: The NGINX Ingress controller supports the WebSocket by default. You do not need to configure the NGINX Ingress controller to distribute WebSocket requests. If you want to keep WebSocket connections alive, you can adjust the timeout period of connections to backend Services by using annotations. This prevents service interruptions that are caused by connection timeouts. For more information, see Custom timeouts.
- HTTPS: You can add the `nginx.ingress.kubernetes.io/backend-protocol:"HTTPS"` annotation in the Ingress if you want to access HTTPS Services through the NGINX Ingress controller.
- gRPC: gRPC Services can be accessed only through TLS ports. Make sure that TLS-encrypted connections are used when you access gRPC Services through the NGINX Ingress controller. For more

information about how to configure gRPC, see Use an Ingress controller to access gRPC services.

Related information

- Ingress overview
- Ingress FAQ

4.Security

4.1. Security overview

This topic provides an overview of the Alibaba Cloud standard of best practices for the security of Container Service for Kubernetes (ACK). Each of the following topics in this chapter provides an introduction to a category of security challenges and describes how to implement the best practices. Security engineers of enterprises can refer to topics in this chapter to meet their security requirements.

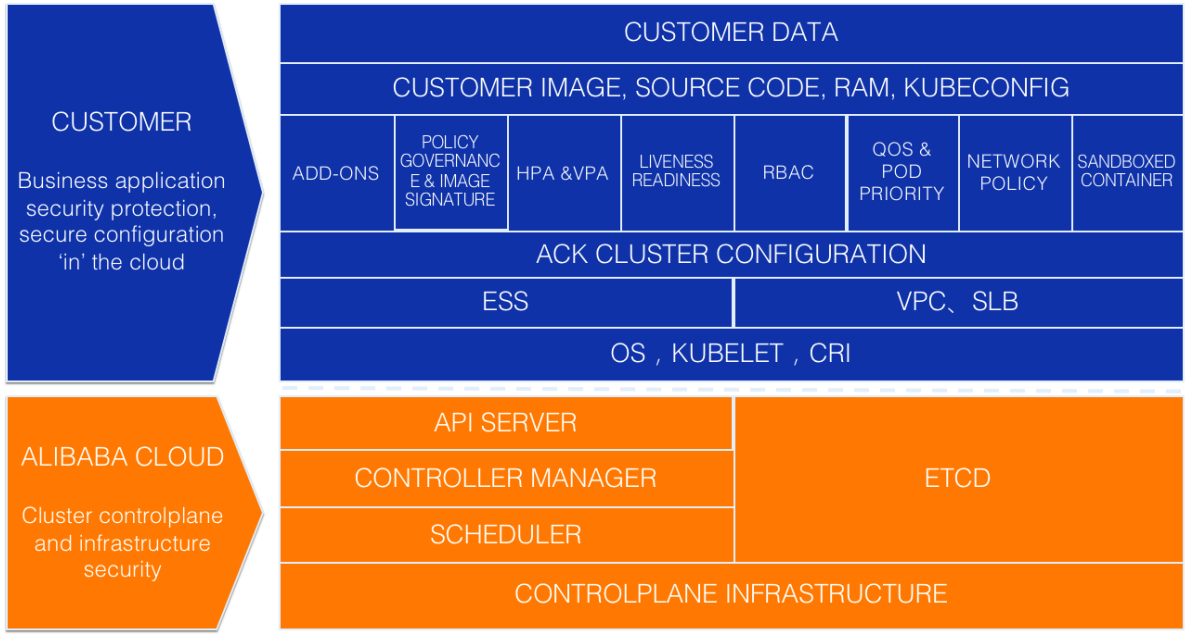
The following categories of security challenges are included:

- Identity verification and access control
- Pod security
- Runtime security
- Supply chain security
- Data encryption and Secret management
- Network security
- Logging and auditing
- Multi-tenancy security
- Host security

Understand the shared responsibility model

In the managed cluster architecture of ACK, security compliance must follow the principle of shared responsibility. ACK is responsible for ensuring the security of the infrastructure resources on which ACK clusters are deployed and the security of control plane components, such as components on master nodes and etcd.

ACK is responsible for the implementation of security from various aspects, including access control, pod security, runtime security, and network security, based on the security capabilities provided by Kubernetes and Alibaba Cloud. ACK is responsible for patching security vulnerabilities related to the node OS and Kubernetes components based on the mitigation plans provided by Alibaba Cloud. You can refer to the topics in this chapter for recommendations on security reinforcement based on your business requirements.



Cloud service provider side

A cloud service provider must provide a cloud platform that has the security capabilities necessary for deploying infrastructure resources on which containerized applications can securely and stably run. In addition, a cloud service provider must provide measures to protect containers throughout the entire container lifecycle, including image building, application deployment, and runtime monitoring. The container security system must comply with the following basic principles:

- **Security of the infrastructure resources managed by the cloud container platform**

  Business services run on the infrastructure resources that are managed by the cloud container platform. The cloud container platform must have the security capabilities necessary to ensure the security of infrastructure resources.

  - Comprehensive security capabilities of the platform: The security of infrastructure resources is the foundation of platform security. Alibaba Cloud provides security configurations for Virtual Private Cloud (VPC), access control for Server Load Balancer (SLB), and security features for DDoS mitigation. Alibaba Cloud also provides an account system to manage access to cloud resources.
  - Version updates and emergency response to vulnerabilities: Version updates and vulnerability patching for operating systems that run on VMs are also part of the basic security measures. Attackers can exploit vulnerabilities of other open source projects related to containerization, such as Kubernetes. The cloud service provider must develop an emergency response system for vulnerabilities of all severity levels and enable version upgrade for these open source projects.
  - Security compliance of the platform: The security compliance of the platform is the hard requirement for migrating the business of financial and public service sectors to the cloud. The cloud service provider must ensure security compliance based on the industrial standards, ensure the security of service and component configurations, and provide a comprehensive audit system for the platform and security auditors.

- **In-depth defense system for containerized applications**

In addition to setting up a comprehensive defense system on the control plane, the cloud service provider must provide in-depth security measures for applications throughout the entire lifecycle. Cloud-native computing is based on dynamic and elastic infrastructure resources, distributed application architectures, and advanced delivery and O&M methods. A cloud service provider must upgrade traditional security models to support cloud-native computing and develop a new security system for cloud-native computing.

Enterprise side

The security administrators and O&M engineers of an enterprise must understand the shared responsibility model and the responsibility boundary between the cloud service provider and the enterprise. The enterprise must assume its own security responsibilities. Cloud-native applications that are developed based on the microservice architecture use data centers and the cloud for deployment and communication. Traditional security boundaries no longer exist between networks. The network security system of enterprise applications must be developed based on the zero-trust security model. An access control system must be developed based on authentication and authorization. To reinforce application security in production environments throughout the entire application lifecycle, enterprise administrators must ensure security from the following aspects:

- **Ensure supply chain security for application artifacts**

  With the development of cloud-native computing, an increasing number of large-scale containerized applications are deployed in the production environments of enterprises. A wide variety of cloud-native application artifacts, such as container images and Helm charts, are also used to deploy containerized applications. Supply chain security is the basic requirement for the security of enterprise applications in production environments. To achieve supply chain security, the enterprise must ensure the security of artifacts during the application building stage. In addition, the enterprise must develop access control, security scan, auditing, and admission mechanisms to detect security risks when artifacts are registered, distributed, and deployed.

- **Grant permissions and deliver credentials in compliance with the principle of least privilege**

  An authorization approach that is based on a unified identity authentication system is the basis of an access control system that follows the zero-trust security model. Security administrators must manage access to cloud resources and containerized applications with strict adherence to the principle of least privilege. To do this, security administrators must combine the access control capabilities provided by cloud service providers and the account system of the enterprise. In addition, security administrators must set strict rules for approving the delivery of credentials. Security administrators must revoke credentials that may be exposed and exploited to escalate privileges at the earliest opportunity. Security administrators must reject the deployment of containers that are granted excessive permissions. This minimizes the chances of attacks.

- **Focus on data security and runtime security**

  Security administrators must continue the security work after applications are deployed. To focus on the runtime security of applications, administrators must develop a comprehensive audit system for resource requests. In addition, security administrators must use the runtime monitoring system, runtime alerting system, and event notification system that are provided by cloud service providers. This way, security administrators can receive notifications of attacks and security risks in a timely manner. To ensure security for all classes of sensitive data, such as database passwords and private keys of certificates, security administrators must enforce key encryption and use the key management methods, disk encryption capabilities, and confidential computing features provided by cloud service providers. This secures data transfer and data storage.

- **Patch vulnerabilities and update components promptly**

Attackers can exploit vulnerabilities of operating systems that run on VMs, vulnerabilities of container images, and vulnerabilities of the container platform to intrude into applications. Therefore, security administrators and O&M engineers must fix vulnerabilities and update component versions, such as Kubernetes versions and image versions, based on the suggestions of cloud service providers. In addition, security training is required to improve the security awareness of employees.

4.2. Identity verification and access control

The identity verification and access control system provides the authentication and authorization features. Authentication is used to verify the identity of a user when the user accesses a cluster. Authorization is used to grant a user permissions to access resources in a cluster.

Authentication and authorization in ACK clusters

Kubernetes uses X.509 certifications, bearer tokens, authentication proxies, or the OpenID Connect (OIDC) protocol to authenticate API requests. For more information about the default methods that are used to access Container Service for Kubernetes (ACK) clusters, see client certificates and service account tokens.

You can obtain a kubeconfig file that contains a client certificate in the ACK console or by calling the ACK API. For more information, see Query the kubeconfig file of a cluster and Use service account tokens to access ACK clusters.

The authorization mechanism of ACK consists of Resource Access Management (RAM) authorization and role-based access control (RBAC) authorization. If you want to modify the configuration of a cluster, scale a cluster, add nodes to a cluster, or access a cluster as a RAM user, you must perform RAM authorization. RAM authorization allows you to attach system policies or custom policies to a RAM user. If you want to manage Kubernetes resources in a specified cluster by using a RAM user, you must go to the Authorizations page of the console and grant the RAM user resource-level permissions, such as the permissions to view information about pods and nodes. For more information, see Authorization overview.

- **Use a temporary kubeconfig file for authentication when you access the Kubernetes API server of a cluster**

  The default kubeconfig file of an ACK cluster contains a certificate that is valid for three years. If the kubeconfig file is exposed, attackers can use it to gain access to the Kubernetes API server of the cluster. A default service account token is a static credential that is permanently valid. If a default service account token is exposed, attackers can exploit the token to gain the same permissions as the relevant service account until it is deleted.

  If a delivered kubeconfig file may be exposed, we recommend that you revoke the kubeconfig file at the earliest opportunity. For more information, see Revoke a KubeConfig credential. For more information about the kubeconfig file of an ACK cluster, see Generate a temporary kubeconfig.

- **Grant access to Alibaba Cloud resources in compliance with the principle of least privilege**

  When you authorize a user to access ACK clusters, do not grant the permissions to access resources of other Alibaba Cloud services. You can use RAM authorization and RBAC authorization to grant a user the permissions to access ACK clusters.

- **Create RoleBindings and ClusterRoleBindings in compliance with the principle of least privilege**

You must create `RoleBindings` and `ClusterRoleBindings` in compliance with the principle of least privilege. When you define a `Role` or `ClusterRole`, specify the required actions instead of `["*"]` in the `Verbs` field. If you are not sure about the permissions to grant, you can use tools, such as audit2rbac, to help create roles and role bindings. audit2rbac can generate roles and role bindings that cover all API operations that are recorded in the `Kubernetes` audit log.

- **Limit access to the endpoint of an ACK cluster**

  By default, the endpoint of an ACK cluster can be accessed only from within the cluster and the virtual private cloud (VPC) in which the cluster is deployed. For more information about how to enable Internet access for Services and modify the control access to the endpoint, see Use annotations to configure load balancing.

- **Audit the permissions of users on a regular basis**

  The permissions that a user have on a cluster may change by time. The security administrators and O&M engineers of a cluster must regularly check the RAM permissions and RBAC permissions of each RAM user and make sure that only the required permissions are granted. The security administrators and O&M engineers can use open source tools to check the RBAC permissions that are granted to a service account, user, or group, and remove permissions that are not required. For more information, see kubectl-who-can and rbac-lookup.

Pod authentication

In Kubernetes clusters, some applications and programs require the permissions to call the API operations of Kubernetes. In ACK clusters, the cloud controller manager (CCM) requires the permissions to add, delete, modify, and query nodes.

- **Kubernetes Service Accounts**

  A service account can be used to assign an RBAC role to a pod. Kubernetes creates a default service account for each namespace in the cluster. When you create a pod, if you do not specify a service account, the pod is automatically assigned the default service account in the same namespace, and a Secret that stores a JSON web token (JWT) is mounted to the */var/run/secrets/kubernetes.io/serviceaccount* directory of the pod as a volume. If you decode the token, the following metadata is returned:

```
{
  "iss": "kubernetes/serviceaccount",
  "kubernetes.io/serviceaccount/namespace": "default",
  "kubernetes.io/serviceaccount/secret.name": "default-token-vpc2x",
  "kubernetes.io/serviceaccount/service-account.name": "default",
  "kubernetes.io/serviceaccount/service-account.uid": "1d059c50-0818-4b15-905d-bbf05e1d**
**",
  "sub": "system:serviceaccount:default:default"
}
```

  The default service account has the following permissions on the Kubernetes API:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:discovery
rules:
- nonResourceURLs:
  - /api
  - /api/*
  - /apis
  - /apis/*
  - /healthz
  - /openapi
  - /openapi/*
  - /version
  - /version/
  verbs:
  - get
```

The role can be used to grant unauthenticated and authenticated users the permissions to read API information. This role is deemed safe to be publicly accessible.

When an application or program that runs in a pod calls the Kubernetes API, the pod must be assigned a service account that has the required permissions. The permissions defined in the Role or ClusterRole that is associated with the service account must be scoped to the API resources that the pod needs to access and the methods that the pod needs to use. This complies with the principle of least privilege. To use non-default service accounts, set the `spec.serviceAccountName` field in the pod configuration to the name of the service account that you want to use. For more information about how to create a service account, see ServiceAccount permissions.

- **Enable service account token volume projection**

  ACK provides service account token volume projection to reduce security risks when pods use service accounts to access the Kubernetes API server. This feature enables kubelet to request and store the token on behalf of a pod. This feature also allows you to configure token properties, such as the audience and validity period. If a token has existed for more than 24 hours or only 20% or fewer of its validity period remains, kubelet automatically rotates the token. For more information, see Enable service account token volume projection.

- **Limit access to instance metadata**

  Elastic Compute Service (ECS) instance metadata contains the information about ECS instances in Alibaba Cloud. You can view the metadata of running instances and configure or manage the instances based on their metadata. Instance metadata contains sensitive information, such as the used cloud resources and user data. If the metadata of an instance is exposed to pods that are deployed on the instance, the information may be exploited by attackers in multi-tenant scenarios. You can use network policies to allow or deny pods to access meta-server. The following code block shows a network policy that allows pods with the labels that are specified in the `podSelector` parameter to access meta-server:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-metadata-access
  namespace: example
spec:
  podSelector:
    matchLabels:
      app: myapp
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 100.100.100.200/32
```

For more information, see Use network policies.

- **Disable automatic mounting of service account tokens for pods**

  You can use one of the following methods to disable automatic mounting of service account tokens for pods:

  Method 1: Set `automountServiceAccountToken` in PodSpec to `false` by using the following YAML template:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: build-robot
  automountServiceAccountToken: false
  ...
```

  Method 2: Run the following command to set the `automountServiceAccountToken` parameter of the default service account in a namespace to `false` by using a patch:

```
kubectl patch serviceaccount default -p $'automountServiceAccountToken: false'
```

- **Assign a separate service account to each application**

  To enable fine-grained application permission management, you must assign a separate service account to each application. For more information, see Configure service accounts for pods.

- **Run applications as non-root users**

  By default, a container runs as a root user. This allows processes in a container to perform read and write operations on data in the container, which violates the best practices for security. To run a container as a non-root user, add the `spec.securityContext.runAsUser` parameter to `PodSpec`. Set `runAsUser` to a proper value based on your requirements.

  The following template specifies that all processes in the pod run with the user ID that is specified in the `runAsUser` field:

```
apiVersion: v1
kind: Pod
metadata:
  name: security-test
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
  containers:
  - name: sec-test
    image: busybox
    command: [ "sh", "-c", "sleep 1h" ]
```

This way, processes in the container cannot access files that require root privileges. If you add `fsgroup=65534 [Nobody]` to the `securityContext` parameter, processes in the container are allowed to access these files.

```
spec:
  securityContext:
    fsGroup: 65534
```

- **Grant permissions on Alibaba Cloud resources in compliance with the principle of least privilege**

  Do not grant unnecessary permissions to RAM users or RAM roles. You must configure RAM policies in compliance with the principle of least privilege and must not configure settings that may grant unnecessary permissions, such as specifying `["*"]` in the policy content.

Related information

- Create a custom RAM policy
- Assign RBAC roles to RAM users or RAM roles
- Overview of ECS instance metadata

4.3. Pod security

This topic describes how to enforce pod security policies to protect your Container Service for Kubernetes (ACK) clusters.

Prevent container escapes that allow attackers to escalate privileges

Kubernetes developers or O&M administrators must focus on how to prevent container escapes that allow attackers to escalate privileges to control the host. Preventing container escapes is important due to the following reasons. By default, processes within a container run under the context of the `[Linux] root` user. The operations that the `root` user can perform are limited due to the `Linux capabilities` that are assigned by Docker to the container. However, an attacker can exploit default capabilities to escalate privileges or access sensitive information on the host, such as `Secrets` and `ConfigMaps`. The following code block shows the default `capabilities` that are assigned to a Docker container. For more information, see capabilities(7) - Linux manual page.

`cap_chown, cap_dac_override, cap_fowner, cap_fsetid, cap_kill, cap_setgid, cap_setuid, cap_setpcap, cap_net_bind_service, cap_net_raw, cap_sys_chroot, cap_mknod, cap_audit_write, cap_setfcap`.

To prevent container escapes, you must avoid running Docker containers with the `privileged` flag because a privileged container is assigned all Linux `capabilities` of the root user.

All Kubernetes worker nodes use the node authorizer, which is a special-purpose authorization mode. The node authorizer is used to authorize all API requests that are sent by a kubelet. The node authorizer also allows a node to perform the following operations:

Read operations:

- Services
- Endpoints
- Nodes
- Pods
- Secrets, ConfigMaps, persistent volumes (PVs), and persistent volume claims (PVCs) of pods that are deployed on the node where the kubelet runs.

Write operations:

- Nodes and node status. Enable the `NodeRestriction` admission plug-in to allow the kubelet to modify only the node where the kubelet runs.
- Pods and pod status. Enable the `NodeRestriction` admission plug-in to allow the kubelet to modify only the pod that is deployed on the node where the kubelet runs.
- Events

Authentication-related operations:

- Read and write permissions on the `CertificateSigningRequest (CSR)` API for Transport Layer Security (TLS) bootstrapping.
- Create `TokenReview` and `SubjectAccessReview` for reviewing delegated identity authentication and authorization.

By default, ACK clusters use the NodeRestriction admission controller. The NodeRestriction admission controller allows a kubelet to modify only the Node API object and Pod API objects that are bound to the node. However, the admission controller cannot prevent attackers from collecting sensitive information from the Kubernetes API. For more information, see NodeRestriction.

PodSecurityPolicy (PSP) is deprecated in Kubernetes 1.21. We recommend the users that use the PSP feature to find an alternative feature before the PSP feature is removed in Kubernetes 1.25. The Kubernetes community is developing a built-in admission controller to replace PSP. ACK will provide policy governance solutions that use the Open Policy Agent (OPA) to replace the PSP feature in later versions.

Suggestions on pod security

- **Limit the containers that can run in privileged mode**

  Privileged containers inherit all Linux capabilities of the root user on the same host. In most scenarios, containers do not need these capabilities to handle workloads. You can create a pod security policy to forbid pods to run in privileged mode. The pod security policy is a group of constraints that a pod must meet before the pod can be created. The PodSecurityPolicy admission controller of Kubernetes validates requests for creating and updating pods in your cluster based on the rules that you configured. If a request for creating or updating a pod does not meet the rules, the request is rejected and an error is returned.

By default, the PodSecurityPolicy admission controller is enabled for ACK clusters and a pod security policy named `ack.privileged` is created. The default pod security policy allows all types of pods. We recommend that you manage pod security policies by namespace based on the principle of least privilege. For example, privileged pods cannot be provisioned in a specified namespace, only read-only file systems can be used, or only host paths within a specified range can be mounted. The following code block shows an example of a pod security policy:

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
    name: restricted
    annotations:
        seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default,runtime
/default'
        apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
        seccomp.security.alpha.kubernetes.io/defaultProfileName:  'runtime/default'
        apparmor.security.beta.kubernetes.io/defaultProfileName:  'runtime/default'
spec:
    privileged: false
    # Prevent escalations to root.
    allowPrivilegeEscalation: false
    requiredDropCapabilities:
    - ALL
    # Allow core volume types.
    volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that the PVs created by the cluster administrator are safe to use.
    - 'persistentVolumeClaim'
    hostNetwork: false
    hostIPC: false
    hostPID: false
    runAsUser:
        # Require pods to run without root privileges.
        rule: 'MustRunAsNonRoot'
    seLinux:
        # Assume that all nodes use AppArmor instead of SELinux.
        rule: 'RunAsAny'
    supplementalGroups:
        rule: 'MustRunAs'
        ranges:
        # Forbid users to add the root group.
        - min: 1
          max: 65535
    fsGroup:
        rule: 'MustRunAs'
        ranges:
        # Forbid users to add the root group.
        - min: 1
          max: 65535
    readOnlyRootFilesystem: false
```

The preceding policy can be used to prevent privileged pods or escalations to root. The policy can also be used to limit the types of volumes that can be mounted and forbid users to add the root group. For more information about how to enhance pod security by using pod security policies, see Pod Security Policies.

- **Run pods as a non-root user**

By default, all pods run with the `root` privileges. Attackers can exploit vulnerabilities in applications and then gain access to the `shell` of a pod. This poses risks to pod security. You can use multiple methods to mitigate the risks. You can delete the `shell` from the container image. You can also add the `USER` instruction to the `Dockerfile` or run the containers as a `non-root` user. The `spec.securityContext` attribute in the `podSpec` contains the `runAsUser` and `runAsGroup` fields. The two fields specify the users and groups for the containers that you want to run. You can create a pod security policy to forcibly enable these fields. For more information, see Users and groups.

- **Forbid users to run Docker-in-Docker containers or mount Docker.sock to containers**

  You can build or deploy container images inside a Docker container by using the Docker-in-Docker method or mounting Docker.sock. However, the process that runs inside the container gains access to the node. For more information about building container images on Kubernetes, see Use Container Registry Enterprise Edition instances to build images, Kaniko, and img.

- **Limit the use of hostPath volumes, or allow only hostPath volumes that are set to read-only and mount paths that start with specified prefixes**

  A `hostPath` volume mounts a path from the host to a pod. In most cases, pods do not require hostPath volumes. Make sure that you understand the risks if you want to use hostPath volumes. By default, pods that run with the root privileges have read permissions on file systems that are exposed by using `hostPath` volumes. Attackers can modify the `kubelet` settings, and then create symbolic links to paths or files that are not directly exposed by `hostPath` volumes. For example, attackers can access `/etc/shadow`, install SSH keys, read Secrets that are mounted to the host, and perform other malicious activities. To mitigate the risks that arise from `hostPath` volumes, set `spec.containers.volumeMounts` to read-only. Example:

  ```
  volumeMounts:
  - name: hostPath-volume
      readOnly: true
      mountPath: /host-path
  ```

  You can also use a pod security policy to limit the paths that can be mounted by using `hostPath` volumes. For example, the following pod security policy specifies that only paths that start with `/foo` on the host can be mounted.

  ```
  allowedHostPaths:
  # This allows "/foo", "/foo/", "/foo/bar" etc., but
  # disallows "/fool", "/etc/foo" etc.
  # "/foo/../" is never valid.
  - pathPrefix: "/foo"
      readOnly: true # only allow read-only mounts
  ```

- **Set resource requests and limits for each container to avoid resource competition or prevent DoS attacks**

  A pod without resource requests or limits can consume all of the resources on a host. If additional pods are scheduled to a node, the CPU or memory resources of the node may be exhausted. As a result, the `kubelet` may crash or pods may be evicted from the node. This issue is inevitable. However, you can set resource requests and limits to minimize resource competition and reduce the risks from improperly programmed applications that consume excessive resources.

You can specify requests and limits for CPU and memory resources in the podSpec. You can set a resource quota or limit range on a namespace to force the use of requests and limits. A resource quota specifies the total amount of resources that are allocated to a namespace, such as CPU and memory resources. After you apply a resource quota to a namespace, the resource quota forces you to specify requests and limits for all containers deployed in the namespace. A limit range can be used to enforce fine-grained control on the resources that are allocated. You can set limit ranges to specify the maximum and minimum amounts of CPU and memory resources that each pod or container in a namespace can use. You can also use limit ranges to set the default request values or limit values if no default values are provided. For more information, see Managing Resources for Containers.

- **Forbid privileged escalation**

  Privileged escalation allows a process to change the security context under which it runs. For example, `sudo` files are binary files with the `SUID` or `SGID` bit. Privileged escalation is a method that can be used by a user to execute a file with the permissions of another user or user group. To prevent privileged escalation, you can use a `pod security policy` that has `allowPrivilegedEscalation` set to `false` or specify `securityContext.allowPrivilegedEscalation` in the `podSpec`.

- **Disable automatic ServiceAccount token mounting**

  For pods that do not need to access the Kubernetes API, you can disable automatic `ServiceAccount` token mounting in the `PodSpec` of specific pods, or disable this feature for all pods that use a specific `ServiceAccount`.

  ```
  apiVersion: v1
  kind: Pod
  metadata:
    name: pod-no-automount
  spec:
    automountServiceAccountToken: false
  ```

  After you disable automatic `ServiceAccount` token mounting for a pod, the pod can still access the Kubernetes API. To prevent a pod from accessing the Kubernetes API, you must regulate access control on the `endpoint` of the ACK cluster and configure network policies to block the pod. For more information, see Use network policies.

  ```
  apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: sa-no-automount
  automountServiceAccountToken: false
  ```

- **Disable service discovery**

  You can reduce the amount of information provided to a pod if the pod does not need to look up and call cluster services. You can modify the CoreDNS policy of a pod to not use CoreDNS and to not expose Services as environment variables in the namespace of the pod. For more information, see Environment variables.

  By default, the DNS policy of a pod is set to `ClusterFirst`, which requires the pod to use the in-cluster DNS service. If the DNS policy is set to `Default`, the pod is required to use the DNS resolution configurations from the underlying node. For more information, see Pod DNS policy.

After you disable service links and change the DNS policy of a pod, the pod can still access the in-cluster DNS service. Attackers can enumerate Services in an ACK cluster by accessing the in-cluster DNS service. Example: `dig SRV *.*.svc.cluster.local @$CLUSTER_DNS_IP`. For more information about how to prevent service discovery within a cluster, see Use network policies.

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-no-service-info
spec:
    dnsPolicy: Default # The value Default is not the default setting of a DNS policy.
    enableServiceLinks: false
```

- **Configure container images to use a read-only file system**

  You can configure container images to use a read-only file system to prevent attackers from overwriting files in the file system that is used by your application. If your application must write data to the file system, you can set the application to write to a temporary directory or mount a volume to the application. You can configure container images to use a read-only file system by setting the following pod `SecurityContext`:

```
...
securityContext:
  readOnlyRootFilesystem: true
...
```

Related information

- resource quota
- limit range

4.4. Runtime security

The intention of runtime security is to detect and prevent malicious activities in running containers to ensure the security of containers. This topic describes how to use secure computing (seccomp) to enhance the security of containerized applications.

Use seccomp to prevent containerized applications from making specific syscalls to the kernel of the underlying host operating system

- **Configure a container or a pod to use the default seccomp profile by adding an annotation**

  The Linux operating system provides hundreds of syscalls, but most of the syscalls are not required to run containers. To get started with seccomp, use strace to generate a stack trace and check which syscalls your application is making. Then, use a tool such as syscall2seccomp to create a seccomp profile from the data collected from the trace. For more information, see strace and syscall2seccomp.

  Unlike SELinux, seccomp is not designed to isolate containers. However, seccomp can protect the host kernel against unauthorized syscalls. seccomp intercepts syscalls and allows only syscalls that are included in the whitelist. Docker has a default seccomp profile, which is suitable for most general-purpose workloads. For more information, see Default seccomp profile.

You can configure a container or a pod to use the default seccomp profile by adding the following annotation to the specifications of the container or pod:

- Versions earlier than Kubernetes 1.19:

```
annotations:
  seccomp.security.alpha.kubernetes.io/pod: "runtime/default"
```

- Kubernetes 1.19 and later versions:

```
securityContext:
  seccompProfile:
    type: RuntimeDefault
```

- **Create profiles for applications that require additional privileges**

  seccomp profiles are provided by Kubelet alpha. If you want to use seccomp profiles, you must add the `--seccomp-profile-root` flag to the Kubelet arguments.

  AppArmor is similar to seccomp. AppArmor limits the capabilities of a container, including the accessing parts of the file system, to ensure the security of the container runtime. AppArmor can run in enforcement or complain mode. For more information about how to create AppArmor profiles, see bane.

  Apparmor applies only to Ubuntu and Debian distributions of Linux.

  Kubernetes does not provide mechanisms for loading AppArmor or seccomp profiles onto Nodes. You must manually load AppArmor or seccomp profiles or install them onto Nodes when the nodes are bootstrapped. You must perform these operations before your pods are created because the Kubernetes scheduler is unaware of which nodes have profiles.

Suggestions on how to use seccomp

- **Use a third-party solution to maintain seccomp and AppArmor profiles**

  Creating and managing seccomp and AppArmor profiles can be difficult if you are not familiar with Linux security. If you cannot maintain seccomp and AppArmor profiles on your own, you can choose to use a commercial solution provided by a third party. These third-party solutions use machine learning to block or alert abnormal activities, which provide better protection than static profiles such as Apparmor and seccomp.

- **Add or remove Linux capabilities before you configure seccomp policies**

  Capabilities involve various checks on kernel functions that are reachable through syscalls. In most cases, if a kernel function fails the check, the syscall returns an error. The check can be performed at the beginning of a specific syscall, or in areas of the kernel that may be reachable through different syscalls, such as writing to a specific privileged file. Seccomp is also a syscall filter, which is applied to all syscalls before the syscalls are run. A process can set up a filter, which allows seccomp to revoke the permissions to run specific syscalls or specific arguments for specific syscalls.

  Before you get started with seccomp, you must consider whether you can gain control over applications by adding or removing Linux capabilities. For more information, see Set capabilities for a container.

- **Check whether you can secure container runtimes by using PodSecurityPolicy (PSP)**

  PSP provides various methods to improve the security posture of your clusters without increasing complexity. Before you create seccomp and AppArmor profiles, you can check whether the options in pod security policies meet your business requirements.

PSP is deprecated in Kubernetes 1.21. We recommend the users that use the PSP feature to find an alternative feature before the feature is removed in Kubernetes 1.25. The Kubernetes community is developing a built-in admission controller to replace PSP. ACK will provide policy governance solutions that use the Open Policy Agent (OPA) to replace the PSP feature in later versions.

- **Use Alibaba Cloud Security Center**

  You can use Security Center to detect and block threats in runtimes of cloud-native applications. This secures the runtime of each pod. Security Center can automatically obtain information about threats in cloud-native applications and use the information to analyze the threats, identify the sources of the threats, generate suitable responses, and handle the threats. Security Center also associates different types of logs, analyzes contexts, and detects risks in real time, such as malicious code or command execution, SQL injections, and data breaches. This can help prevent intrusions and identify vulnerabilities in your business systems. Security Center can audit actions and identify risks in real time based on Kubernetes logs and operations logs. This helps you mitigate the risks of container escapes, AccessKey breaches, and unauthorized access in ACK and other orchestration platforms. For more information, see What is Security Center?.

Related information

- AppArmor Loader
- Setting up nodes with profiles
- zaz
- seccomp-operator
- Aqua
- Qualys
- Stackrox
- Sysdig Secure
- Twistlock

4.5. Supply chain security

This topic describes how to test, build, and deploy an end-to-end supply chain to secure the lifecycle of a Container Service for Kubernetes (ACK) cluster.

Context

A software supply chain consists of the build, deploy, and runtime stages. Attackers can launch different types of attacks against your workloads in these stages.

Stage	Type of attack
Build	• Pollution attacks against IDE tools. • Exploiting of vulnerabilities and webshells in third-party libraries. • Pollution attacks against source code.

Stage	Type of attack
Deploy	• Substitution and tampering attacks against stored data. • Hijacking of transmitted data and drive-by download attacks.
Runtime	• Hijacking of software updates. • Exploiting of vulnerabilities and webshells in runtimes. • Exploiting of zero-day vulnerabilities in third-party libraries.

Artifact security is the first defense in the build and test stages. An attacker can exploit a malicious container image to escape from the container and gain control over the host. As a result, the attacker can move through the network to access sensitive data or other valued data assets with your Alibaba Cloud account. The admission control mechanism provided by open source Kubernetes can help you verify the security of the provisioned pods in the deploy stage. After you deploy an application, you must monitor the runtime of the application in real time. This way, you can handle security events at the earliest opportunity.

Suggestions on cluster security

We recommend that you follow these suggestions to secure the lifecycle of an ACK cluster.

- **Create smaller container images**

  Delete binary files that you do not need from the container image that you want to use. If the container image is downloaded from Docker Hub, you must use Dive to analyze the image. Dive can help you explore the content of each image layer. For more information, see Dive.

  You can also view the content of each image layer in the Container Registry console after you push the image to Alibaba Cloud Container Registry. You must delete all binary files with the `SETUID` and `SETGID` bits because these files can be exploited to escalate privileges. You can also delete shells and applications that may be exploited by attackers, such as `nc` and `curl`. You can run the following command to search for binary files with the `SETUID` and `SETGID` bits:

  ```
  find / -perm /6000 -type f -exec ls -ld {} \;
  ```

  Add the following command to the container image if you want to remove permissions from the binary files:

  ```
  RUN find / -xdev -perm /6000 -type f -exec chmod a-s {} \; || true
  ```

- **Use multi-stage builds**

  You can use multi-stage builds to create smaller container images. Multi-stage builds can be used to automate continuous integration in `CI/CD` pipelines. Multi-stage builds allow you to include multiple FROM statements in your `Dockerfile`. Each FROM statement can use a different base image, and each FROM statement starts a new build stage. Multi-stage builds can help you quickly build trusted container images and keep the size of the container images down. For more information, see Build an image for a Java application by using a Dockerfile with multi-stage builds.

- **Create RAM policies for Container Registry repositories**

  Multiple teams in a company may want to use the same Alibaba Cloud account to manage cloud resources. If these teams do not want to share resources, you must create Resource Access Management (RAM) policies to limit the namespaces or repositories that each team can access. For example, `cr:ListInstance*` specifies all actions that start with `cr:ListInstance`. You can set acs:cr:*:*:repository/$instanceid/$namespace/* to `acs:cr:cn-hangzhou:1234567:repository/cri-123456/ns/*` to grant the instance cri-123456 in the cn-hangzhou region the permissions to query repositories in namespaces. The instance belongs to the Alibaba Cloud account 1234567.

  ```
  {
      "Statement": [
          {
              "Effect": "Allow",
              "Action": [
                  "cr:ListRepository",
                  "cr:GetImageLayer",
                  "cr:GetRepoTag"
              ],
              "Resource": "*"
          },
          {
              "Action": [
                  "cr:List*"
              ],
              "Effect": "Allow",
              "Resource": [
                  "acs:cr:cn-hangzhou:1234567:repository/cri-123456/ns/*",
              ]
          }
      ],
      "Version": "1"
  }
  ```

  For more information about how to configure RAM policies in Container Registry, see Configure policies for RAM users to access Container Registry and Use RAM to grant permissions to access custom OSS buckets.

- **Use Container Registry Enterprise Edition**

  Container Registry Enterprise Edition allows you to encrypt cloud-native artifacts that are stored in Container Registry. Container Registry Enterprise Edition also supports image scans and multi-dimensional vulnerability reports to help you secure storage and content. You can enforce access control on container images and Helm charts, and perform fine-grained action auditing to ensure the security of artifacts. We recommend that you perform the following tasks to enhance security: use Container Registry Enterprise Edition in production environments, set repositories to private, use internal endpoints to access Container Registry instances over virtual private clouds (VPCs), disable Internet access, and configure network access control lists (ACLs). For more information, see Create a Container Registry Enterprise Edition instance.

- **Use the cloud-native application delivery chain provided by Container Registry**

  Container Registry provides a cloud-native application delivery chain. You can streamline tasks such as image building, image scanning, image global synchronization, and image distribution in the delivery chain. The entire delivery chain is observable, traceable, and secured. For more information, see Create a delivery chain.

After you push container images to Container Registry, Container Registry automatically scans the images. You can also configure security policies to enable Container Registry to identify security risks in container images and block images whose severity levels are high. Only images that are allowed by security policies are distributed and deployed. The delivery chain ensures the secure delivery and efficient deployment of containerized applications. You can also integrate the API operations for image scanning into your system to schedule image scans.

- **Periodically scan container images for vulnerabilities**

  Similar to OS images used by virtual machines, container images may contain binary files or applications in which vulnerabilities exist, or vulnerabilities may be added after the binary files or applications are updated. Therefore, we recommend that you periodically scan your container images for vulnerabilities. Container Registry can automatically scan newly uploaded container images and existing container images every 24 hours. If a container image contains vulnerabilities whose severity levels are `HIGH` or `CRITICAL`, you must delete or rebuild the container image. If vulnerabilities are detected in container images that have already been deployed, you must replace the relevant containers at the earliest opportunity.

  You can also invoke the `Kubernetes validating webhook` to identify critical vulnerabilities in container images. You can invoke the `validating webhook` before you call the `Kubernetes API` to reject requests that do not comply with the admission policies defined in the `webhook`. You can call the CreateRepoTagScanTask operation of the Container Registry API to check whether a container image that is being pulled by a cluster contains critical vulnerabilities. If the container image contains critical vulnerabilities, Container Registry forbid the cluster to deploy pods from the container image and generates an event. The event contains the detected vulnerabilities. For more information, see CreateRepoTagScanTask.

- **Add the USER instruction to your Dockerfiles and run containers as a non-root user**

  To ensure pod security, you must avoid running containers as a root user. You can add the USER instruction to the PodSpec. You can use the USER instruction in Dockerfiles as a best practice to ensure pod security. After you add the USER instruction, RUN, ENTRYPOINT, and CMD are executed with the specified user account.

- **Download dependencies from trusted sources**

  You must avoid using dependencies that are downloaded from untrusted sources in the software development stage. For more information about container images, artifacts, or dependencies that are trusted by Alibaba Cloud, see Alibaba Cloud images.

  You can use Apsara Devops to build your own repositories. Apsara Devops Artifact Repository Packages is an enterprise-class private repository service provided by Alibaba Cloud. This service allows enterprises to build private repositories to manage Maven, Gradle, and npm packages and repositories. You can also use this service to manage Maven and npm artifacts, build remote repositories, and migrate your repositories with a few clicks. The Apsara Devops Artifact Repository Packages service supports tenant isolation, permission control, and high-availability storage to secure your artifacts.

- **Use image signing and configure signature verification policies**

  When you deploy applications in a cluster, you must verify the signatures of container images to ensure that only container images that are signed by trusted authorities are used. This helps you prevent exceptions and malicious code execution.

Container Registry Enterprise Edition supports image signing. This prevents man-in-the-middle (MITM) attacks and unauthorized image updates or deployments. This way, image consistency and security from distribution to deployment are ensured. Container Registry can automatically sign images in specific namespaces. After an image is pushed to Container Registry, Container Registry automatically signs the image based on the matched signature signing rules. This ensures that your container images are trusted. For more information, see Sign container images.

You can install the Kritis-validation-hook component in ACK clusters to automatically verify the signatures of container images that are signed by Key Management Service (KMS). This component is developed based on open source Kritis and can be deeply integrated with Container Registry. For more information, see Use kritis-validation-hook to automatically verify the signatures of container images, Kritis, and KMS.

After you enable image signing, you can configure signature verification policies. Only trusted images that are verified by signature verification policies can be deployed in ACK clusters. You can also configure a signature verification whitelist, and then add sidecar container images injected by third-party components to the whitelist. This way, the system does not verify the signatures of the sidecar container images and pods can be deployed from these images as expected. For more information, see Introduction to kritis-validation-hook.

- **Use Alibaba Cloud Security Center**

  You can use Security Center to detect and block threats in runtimes of cloud-native applications. This secures the runtime of each pod. Security Center can automatically obtain information about threats in cloud-native applications and use the information to analyze the threats, identify the sources of the threats, generate suitable responses, and handle the threats. Security Center also associates different types of logs, analyzes contexts, and detects risks in real time, such as malicious code or command execution, SQL injections, and data breaches. This can help prevent intrusions and identify vulnerabilities in your business systems. Security Center can audit actions and identify risks in real time based on Kubernetes logs and operations logs. This helps you mitigate the risks of container escapes, AccessKey breaches, and unauthorized access in ACK and other orchestration platforms. For more information, see What is Security Center?.

4.6. Data encryption and Secret management

You can encrypt the data stored in Elastic Compute Service (ECS) instances to ensure data privacy and autonomy. This topic describes how to encrypt disks that are provided by Alibaba Cloud.

Use specific keys stored in KMS to encrypt disk volumes

Data encryption is suitable for scenarios that require high security or have compliance requirements. Storage encryption helps you ensure the privacy and autonomy of the data that is stored in ECS instances without the need to maintain the underlying key management system. For more information, see Encrypt disk volumes.

This topic describes how to use a specific key stored in KMS to encrypt disk volumes when you create a cluster that uses the disk volumes.

1. Create a StorageClass.

i. Create a file named *sc-kms.yaml* and copy the following content to the file:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: csi-disk
provisioner: diskplugin.csi.alibabacloud.com
parameters:
    fsType: ext4
    type: cloud_ssd
    encrypted: "true"
    kmsKeyId: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
reclaimPolicy: Delete
```

ii. Run the following command to create a StorageClass:

```
kubectl create -f sc-kms.yaml
```

2. Create a persistent volume claim (PVC).

i. Create a file named *sc-pvc.yaml* and copy the following content to the file:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: disk-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  storageClassName: csi-disk
```

ii. Run the following command to create a PVC:

```
kubectl create -f sc-pvc.yaml
```

Suggestions on disk encryption

- **Enable disk encryption**

  You can use static keys to encrypt disks as a best practice to ensure the security of your system. For more information, see Encrypt disk volumes.

- **Rotate CMKs periodically**

  You can periodically rotate keys and configure key versioning to enhance the security of customer master keys (CMKs). For more information, see Automatic key rotation.

Secret management

Kubernetes Secrets are used to store business-critical data and sensitive information, such as passwords, certificates, and API credentials. In open source Kubernetes, all Secrets are encoded by using Base64 and are stored in etcd. In a managed Kubernetes cluster, disk encryption is enabled for all disks that are mounted to etcd nodes in the control plane. This protects the privacy of business data. You can configure a pod to use a specified Secret by specifying specific environment variables or mounting a volume to the pod. For more information, see Secrets.

- **Use KMS to encrypt Kubernetes Secrets**

  Professional managed Kubernetes clusters allow you to use a CMK in KMS to encrypt Secrets. The KMS encryption provider mechanism of Kubernetes is used during encryption. A KMS encryption provider uses envelope encryption to encrypt or decrypt Secrets that are stored in etcd. For more information, see KMS encryption provider mechanism, What is envelope encryption?, and Use KMS to encrypt Kubernetes Secrets.

- **Create a separate namespace to isolate Secrets from applications**

  If you have secrets that cannot be shared among applications in a namespace, you can create a separate namespace for each application, and limit the read and write permissions on Secrets.

- **Use volumes instead of environment variables to mount Secrets**

  The values of environment variables may accidentally appear in logs. `Secrets` that are mounted as volumes are instantiated as `tmpfs` volumes. These volumes are automatically removed from a node when the pod on the node is deleted.

- **Use an external Secret management system**

  You can use an external Secret management system to manage your Secrets based on advanced features, such as fine-grained access control, multiple encryption algorithms, and automatic rotation of Secrets. For more information, see Overview and Vault.

  When an application in an ACK cluster requires a Secret, the Secret management system can synchronize the corresponding external Secret to the ACK cluster in real time, and then inject the Secret to the application pod as a Secret of open source Kubernetes. For example, ACK Secret Manager can synchronize Secrets to ACK clusters in real time. For more information, see ack-secret-manager.

- **Use TEE-based confidential computing**

  ACK allows you to create managed Kubernetes clusters for confidential computing that are developed based on Intel Software Guard Extensions (SGX) 2.0. These clusters can help you ensure the security, integrity, and confidentiality of data computing, and also save the expenses on developing, delivering, and managing trusted or confidential applications. Confidential computing allows you to isolate sensitive data and code in a trusted execution environment (TEE). This prevents the data and code from being accessed by the rest of the system. The data stored within TEEs is inaccessible to external applications, the BIOS, the operating system, the kernel, administrators, O&M engineers, cloud service providers, and hardware components except the CPU. This reduces the possibility of data breaches and simplifies data management. For more information, see TEE-based confidential computing.

4.7. Network security

This topic describes how to ensure network security by enforcing access control on services and encrypting data transmission.

Network policies

By default, pods can communicate with each other in a Kubernetes cluster. This poses security risks in production environments. `Kubernetes network policies` allow you to control traffic between pods and traffic between pods and external services. The traffic between pods refers to the east-west traffic. Network policies use pod selectors and labels to identify source pods and destination pods. In addition, you can specify IP addresses, ports, protocols, and a combination of them in network policies. When you use the Terway network plug-in, you can configure network policies for specific applications if you want to control network traffic at the IP address or port level. For more information, see Use network policies and Kubernetes Network Policy Recipes.

Notice Network policies may increase the loads of the API server in large-scale production environments. Proceed with caution when you use network policies.

Create a default network policy that denies all traffic

Similar to `role-based access control (RBAC)` policies, you must follow the principle of least privilege when you create `network policies`. You can create a default network policy that denies all inbound and outbound traffic from a namespace. You can also create a global network policy by using Calico.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: default
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
```

Create a network policy that allows DNS queries

After you create the default network policy that denies all inbound and outbound traffic, you can create network policies for specific purposes. For example, you can create a global network policy that allows pods to send DNS queries to CoreDNS.

1. Run the following command to add a label to a namespace:

   ```
   kubectl label namespace kube-system name=kube-system
   ```

2. Create a `network policy` by using the following YAML template:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-access
  namespace: default
spec:
  podSelector:
    matchLabels: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: kube-system
    ports:
    - protocol: UDP
      port: 53
```

Notice For more information about how to control network traffic between pods by using Kubernetes network policies, see Official Documentation.

The following example describes how to associate a network policy with a service account named ali-sa. The following example also describes how to forbid a role named readonly-sa-role to modify the ali-sa service account in the default namespace. The role is associated with the readonly-sa-group RBAC group.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ali-sa
  namespace: default
  labels:
    name: ali-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: readonly-sa-role
rules:
# Allows the user to read the ali-sa service account.
- apiGroups: [""]
  resources: ["serviceaccounts"]
  resourceNames: ["ali-sa"]
  verbs: ["get", "watch", "list"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  namespace: default
  name: readonly-sa-rolebinding
# Associates the readonly-sa-role role with an RBAC group named readonly-sa-group.
subjects:
- kind: Group
  name: readonly-sa-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: readonly-sa-role
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: crd.projectcalico.org/v1
kind: NetworkPolicy
metadata:
  name: netpol-sa-demo
  namespace: default
# Allows inbound traffic to services in the default namespace.
# Specifies the service account that is named ali-sa.
spec:
  ingress:
    - action: Allow
      source:
        serviceAccounts:
          selector: 'name == "ali-sa"'
  selector: all()
```

Add custom rules to allow traffic between specific pods in a namespace

After you can create a network policy that allows communication between pods in a namespace, you can add custom rules to limit communication between specific pods in the namespace. For more information, see Kubernetes Network Policy Recipes.

Monitor and analyze traffic data

Alibaba Cloud Virtual Private Cloud (VPC) provides flow logs that record information about inbound and outbound traffic of elastic network interfaces (ENIs). Flow logs help verify access control list (ACL) rules, monitor network traffic, and troubleshoot network issues. You can identify abnormal traffic between resources (including pods) in a VPC by analyzing flow logs. For more information, see Overview of the flow log feature.

Security groups

ACK uses security groups to manage traffic between master nodes and worker nodes. You can also use security groups to manage traffic between worker nodes, other VPC resources, and external IP addresses. When you create an ACK cluster, the system automatically creates a security group for the cluster. The security group allows communication among nodes within the cluster. You can add inbound and outbound rules to the security group based on the settings in the following table to enforce the principle of least privilege.

Rule type	Protocol	Port range	Source	Destination
Inbound rule for least privilege (from the control plane and other nodes)	TCP or protocols that you want to use for communication between nodes	443, 10250, or ports that you want to use for communication between nodes	Cluster security group	N/A
Recommended inbound rule	ALL/TCP	ALL/443, 1025-65535	Cluster security group	N/A
Outbound rule for least privilege	TCP	443	N/A	Cluster security group
Recommended outbound rule	ALL	ALL	N/A	0.0.0.0/0

For more information, see Configure security groups in different scenarios and Configure a security group.

Use AHAS to throttle traffic for an ASM instance

Encrypt data transmission

- **Alibaba Cloud Service Mesh (ASM)**

  ASM can encrypt data transmitted among services. In addition to mutual Transport Layer Security (mTLS) authentication, you can use Envoy Secret Discovery Service (SDS) to enhance the security of service gateways with HTTPS support and dynamic certificate loading. You can use ASM with Application High Availability Service (AHAS) to manage traffic of applications that are deployed in ASM instances. ASM is integrated with Tracing Analysis to provide capabilities for distributed application developers, such as trace mapping, service call counting, trace topology, and application dependency analysis. Developers can use these capabilities to identify and diagnose performance bottlenecks in a distributed application architecture and make development and diagnostics more efficient.

- **Use a Secret to configure TLS to enable HTTPS access**

  You must enable HTTPS access for services that are exposed by Ingresses in clusters. For more information, see Use a Secret to configure TLS to enable HTTPS access.

Related information

- Cilium
- NetworkPolicy Editor
- Kinvolk's Network Policy Advisor
- Alibaba Cloud Service Mesh (ASM)
- What is ASM?
- Use an Istio gateway to enable HTTPS
- Use Tracing Analysis to trace applications inside and outside an ASM instance

4.8. Logging and auditing

This topic describes how to configure Container Service for Kubernetes (ACK) to collect and analyze the audit logs of Kubernetes components, including the audit logs of the API server, Ingresses, control plane components, and key Kubernetes events. This helps you locate causes when security issues or cluster issues are found in the log data.

Use cluster auditing

The audit log of an API server in a Kubernetes cluster helps administrators track operations performed by different users. This plays an important role in the security and maintenance of the cluster. For more information about how to collect and analyze audit logs by using Log Service, set custom alert rules, and disable cluster auditing, see Use cluster auditing.

ACK provides the following audit policies:

```
apiVersion: audit.k8s.io/v1beta1 # This is required.
kind: Policy
#Do not generate audit events for requests in the RequestReceived stage.
omitStages:
- "RequestReceived"
rules:
#Ignore the following requests because the requests are manually identified as high-volume
and low-risk.
- level: None
  users: ["system:kube-proxy"]
  verbs: ["watch"]
  resources:
    - group: "" # core
      resources: ["endpoints", "services"]
- level: None
  users: ["system:unsecured"]
  namespaces: ["kube-system"]
  verbs: ["get"]
  resources:
    - group: "" # core
      resources: ["configmaps"]
- level: None
  users: ["kubelet"] # legacy kubelet identity
```

```
  users: ["kubelet"] # legacy kubelet identity
  verbs: ["get"]
  resources:
    - group: "" # core
      resources: ["nodes"]
- level: None
  userGroups: ["system:nodes"]
  verbs: ["get"]
  resources:
    - group: "" # core
      resources: ["nodes"]
- level: None
  users:
    - system:kube-controller-manager
    - system:kube-scheduler
    - system:serviceaccount:kube-system:endpoint-controller
  verbs: ["get", "update"]
  namespaces: ["kube-system"]
  resources:
    - group: "" # core
      resources: ["endpoints"]
- level: None
  users: ["system:apiserver"]
  verbs: ["get"]
  resources:
    - group: "" # core
      resources: ["namespaces"]
#Do not audit requests that are sent to the following read-only URLs.
- level: None
  nonResourceURLs:
    - /healthz*
    - /version
    - /swagger*
#Do not audit requests that generated upon audit events.
- level: None
  resources:
    - group: "" # core
      resources: ["events"]
#Secrets, ConfigMaps, and token reviews can contain sensitive and binary data.
#Therefore, you can audit only the metadata of these resources.
- level: Metadata
  resources:
    - group: "" # core
      resources: ["secrets", "configmaps"]
    - group: authentication.k8s.io
      resources: ["tokenreviews"]
- level: Request
  verbs: ["get", "list", "watch"]
  resources:
    - group: "" # core
    - group: "admissionregistration.k8s.io"
    - group: "apps"
    - group: "authentication.k8s.io"
    - group: "authorization.k8s.io"
    - group: "autoscaling"
```

```
    - group: "batch"
    - group: "certificates.k8s.io"
    - group: "extensions"
    - group: "networking.k8s.io"
    - group: "policy"
    - group: "rbac.authorization.k8s.io"
    - group: "settings.k8s.io"
    - group: "storage.k8s.io"
#The default audit level for known API requests and responses.
- level: RequestResponse
  resources:
    - group: "" # core
    - group: "admissionregistration.k8s.io"
    - group: "apps"
    - group: "authentication.k8s.io"
    - group: "authorization.k8s.io"
    - group: "autoscaling"
    - group: "batch"
    - group: "certificates.k8s.io"
    - group: "extensions"
    - group: "networking.k8s.io"
    - group: "policy"
    - group: "rbac.authorization.k8s.io"
    - group: "settings.k8s.io"
    - group: "storage.k8s.io"
    - group: "autoscaling.alibabacloud.com"
#The default audit level for other requests.
- level: Metadata
```

Use the audit log metadata

The Kubernetes audit log contains two annotations: `authorization.k8s.io/decision` and `authorization.k8s.io/reason`. The authorization.k8s.io/decision annotation indicates whether a request is authorized. The authorization.k8s.io/reason annotation indicates the reason for making the decision. The annotations are used to specify the reasons why specific API operations can be called.

Use node-problem-detector with the Kubernetes event center of Log Service to identify abnormal cluster events

node-problem-detector is a tool maintained by ACK to diagnose Kubernetes nodes. node-problem-detector detects node exceptions, generates node events, and works with kube-eventer to generate alerts upon these events and enable closed-loop management of alerts. node-problem-detector generates node events when the following exceptions are detected: Docker engine hangs, Linux kernel hangs, outbound traffic exceptions, and file descriptor exceptions. In addition to node issues and exceptions detected by node-problem-detector, a Kubernetes cluster also generates events when the status of the cluster changes. For example, a Kubernetes cluster generates events when a pod is evicted and the cluster fails to pull an image. The Kubernetes event center of Log Service collects all events generated in Kubernetes clusters and provides the following capabilities: storage, query, analytics, visualization, and alerting. The Kubernetes event center helps O&M administrators identify issues that may affect the cluster stability and abnormal events, such as regular users running the exec command to log on to specific containers. For more information, see Event monitoring.

Enable the Ingress dashboard

Ingress controllers of ACK allow you to stream all HTTP request log data to standard outputs. ACK is also integrated with Log Service. You can create dashboards to monitor and analyze log data. The Ingress dashboard displays the following information about the status of Ingresses in a cluster: the number of page views (PVs), the number of unique visitors (UVs), inbound and outbound traffic, the average latency, and top URLs. This helps you gain insights into the service traffic, and detect malicious traffic and DDoS attacks at the earliest opportunity. For more information, see Ingress Dashboard.

Enable logging for CoreDNS

CoreDNS is deployed in ACK clusters and serves as a DNS server. You can check the log of CoreDNS to locate the causes of slow DNS resolution or analyze DNS queries for high-risk domain names. You can view the analytical report of the CoreDNS log in Log Service dashboards. This helps you identify DNS queries for high-risk domain names. For more information, see Monitor CoreDNS and analyze the CoreDNS log.

4.9. Multi-tenancy security

This topic describes how to fairly allocate resources to tenants in a shared cluster to prevent malicious tenants from attacking other tenants.

Context

Multi-tenancy is classified into soft multi-tenancy and hard multi-tenancy based on the level of isolation.

- Soft multi-tenancy is suitable for trusted tenants. For example, soft multi-tenancy can separate a Kubernetes cluster among multiple departments of an enterprise. In this scenario, the intention of multi-tenancy is to protect the workloads of each department and prevent potential threats.
- Hard multi-tenancy is suitable for untrusted tenants. For example, a service provider may need to provide infrastructure resources to different organizations. Hard multi-tenancy enforces stricter tenant isolation by preventing a tenant from accessing other tenants or the Kubernetes system.

Soft multi-tenancy

You can use the following `Kubernetes` -native resources to implement soft multi-tenancy: `namespaces` , `roles` , `role bindings` , and `network policies` . This enables logical isolation among tenants. For example, you can use `role-based access control (RBAC)` to prevent tenants from accessing or manipulating the resources of other tenants. You can use `quotas` and `limit ranges` to manage the amount of cluster resources that can be consumed by each tenant. You can also use `network policies` to prevent applications that are deployed in different namespaces from communicating with each other.

However, you cannot use these control methods to prevent `pods` of different tenants from sharing a node. You can use `node selectors` , `anti-affinity` rules, `taints` , and `tolerations` to forcibly schedule pods of different tenants to separate nodes. These nodes are known as sole tenant nodes. The complexity and cost of sole tenant nodes are significantly increased when a cluster is shared by a large number of tenants.

If you use `namespaces` to implement soft multi-tenancy, you are not allowed to provide tenants with a filtered list of namespaces. This is because namespaces are globally scoped resources. If a tenant can access a specific namespace in a cluster, the tenant can access all namespaces in the cluster.

When soft multi-tenancy is used, tenants can query `CoreDNS` for all services that run in the cluster by default. Attackers can exploit this capability by running `dig SRV ..svc.cluster.local` from a `pod` in the cluster. If you want to limit access to DNS records of services that run in a cluster, you can use the firewall or policy plug-ins for `CoreDNS`. For more information, see kubernetes-metadata-multi-tenancy-policy.

- **Enterprise setting**

  The soft multi-tenancy model is widely used by Kubernetes enterprise users. If all tenants of a Kubernetes cluster belong to the same enterprise, the roles of the service users are manageable. This helps enterprises gain control over the security of their business. Each tenant corresponds to an administrative division, such as a department or team.

  In this scenario, cluster administrators are responsible for creating namespaces and managing policies. The cluster administrators can use a delegated administration model where specific tenants are granted permissions on namespaces. The tenants are granted permissions to perform `CRUD` operations on non-policy related objects, such as `Deployments`, `Services`, `pods`, and `Jobs`.

  Isolation methods that are provided by Docker are suitable for this scenario. You can also use other methods, such as Pod Security Policies (PSPs) based on your business requirements. You may want to limit communication among services in different namespaces if stricter isolation is required.

- **Kubernetes as a Service (KaaS)**

  You can use soft multi-tenancy in scenarios in which you want to provide Kubernetes as a service. In KaaS environments, your applications are hosted in a shared cluster. The shared cluster contains controllers and CustomResourceObjects (CRDs) that provide a set of Platform as a Service (PaaS) services. Tenants interact with the Kubernetes API server and are allowed to perform CRUD operations on non-policy objects. A self-service feature is provided. For example, tenants are allowed to create and manage their namespaces. In this type of environment, tenants are considered to be running untrusted code.

  If you want to isolate tenants in this type of environment, you may need to use network policies and pod sandboxing. For more information, see Sandboxed containers.

- **Software as a Service (SaaS)**

  In SaaS environments, each tenant is associated with a specific instance of an application that runs in the cluster. Each instance contains data and uses separate access control policies other than `Kubernetes RBAC`.

  Tenants in a SaaS environment do not interact with the `Kubernetes API server`. The SaaS application interacts with the `Kubernetes API server` to create objects that are required by each tenant.

Kubernetes configurations

Kubernetes is a single-tenant orchestration platform that is used to manage containerized workloads. When you use Kubernetes, the control plane is shared among all tenants in a cluster. You can use various Kubernetes objects to isolate tenants. For example, you can use namespaces and `RBAC` to logically isolate tenants from each other. You can also use `quotas` and `limit ranges` to control the amount of cluster resources that can be consumed by each tenant. Each cluster provides a strong security boundary. Attackers that gain access to a host in the cluster can retrieve all `Secrets`, `ConfigMaps`, and `volumes` that are mounted to the host. The attackers can also exploit the `kubelet`, which allows them to manipulate the attributes of the node or move laterally within the cluster. The following Kubernetes-native resources can help you mitigate the risks that may occur when you use single-tenant orchestration platforms, such as Kubernetes. You can also use the Kubernetes resources to isolate tenants by using the methods that are provided in the preceding section.

- **Namespaces**

  Namespaces are the basis for implementing soft multi-tenancy. You can use namespaces to divide a cluster into logical partitions. Quotas, network policies, service accounts, and other objects that are required to implement soft multi-tenancy must be scoped to a namespace.

- **AuthN&AuthZ&Admission**

  The authorization of Container Service for Kubernetes (ACK) clusters consists of Resource Access Management (RAM) authorization and RBAC authorization. You can use RAM authorization to regulate cluster-level access control, including CRUD operations on a cluster. For example, you can manage the visibility of a cluster, scale a cluster, and add nodes to a cluster by granting the required permissions to a RAM user. RBAC authorization is used to control access to Kubernetes resources in a cluster. You can use RBAC authorization to enforce fine-grained access control on a specified resource in a namespace. ACK provides templates of different predefined roles for users in a tenant. You can also associate multiple custom cluster roles with a user and grant permissions to multiple users at a time. For more information, see Authorization overview.

- **Network policies**

  By default, all pods in a Kubernetes cluster are allowed to communicate with each other. You can use network policies to modify this default setting.

- Network policies limit communication among pods based on labels or CIDR blocks. If you require network isolation among tenants in a multi-tenant environment, add the following rules:
  - A default rule that denies communication among pods.
  - A rule that allows pods to send DNS queries to the DNS server.

- **Quotas and limit ranges**

  Quotas are used to define limits on workloads that are hosted in your cluster. You can use quotas to specify the maximum amount of CPU and memory resources that can be consumed by a pod. You can also use quotas to specify the amount of resources that can be allocated for a cluster or namespace. Limit ranges allow you to specify the minimum, maximum, and default values for each limit.

  To maximize resource utilization, you can overcommit resources in a shared cluster. If access to a cluster is unlimited, resources in the cluster may be exhausted. This degrades the performance of the cluster and affects the availability of your applications. If you set the request threshold of a pod to a small value and the actual resource utilization exceeds the capacity of the node, the CPU or memory resources of the node may be exhausted. When the CPU or memory resources are exhausted, the pod may be restarted or evicted from the node.

To prevent this issue, you must add quotas for namespaces in a multi-tenant environment. This forces tenants to specify request and limit thresholds when they schedule pods in a cluster. In addition, the amount of resources that can be consumed by a pod is limited. This mitigates the risk of service unavailability.

In KaaS scenarios, you can use quotas to allocate cluster resources to meet the requirements of tenants.

- **Pod priority and preemption**

  If you want to provide different quality of service (QoS) levels for customers, you can use pod priority and preemption. For example, you can specify a higher priority value for pods of Customer A than Customer B. If the resource capacity is insufficient, the kubelet evicts pods with lower priority values from customer B to satisfy pods with higher priority values of customer A. You can use this method in a SaaS environment to provide a higher QoS level for customers that pay a premium fee.

Mitigation methods

The major concern of administrators in a multi-tenant environment is preventing attackers from gaining access to underlying hosts. You can use one of the following methods to prevent attackers from gaining access to underlying hosts:

- **Sandboxed-Container**

  Sandboxed-Container is an alternative to the Docker runtime. Sandboxed-Container allows you to run applications in a sandboxed and lightweight virtual machine that has a dedicated kernel. This enhances resource isolation and improves security.

  Sandboxed-Container is suitable in scenarios such as untrusted application isolation, fault isolation, performance isolation, and load isolation among multiple users. Sandboxed-Container provides enhanced security, has minor impacts on application performance, and offers the same user experience as Docker in terms of logging, monitoring, and elastic scaling. For more information, see Sandboxed-Container overview.

- **Open Policy Agent (OPA) & Gatekeeper**

  Open Policy Agent (OPA) is a powerful policy engine. OPA supports decoupled policy decisions and is used in Kubernetes. If the security requirements of enterprise applications are not met after RBAC is used to perform isolation at the namespace level, you can control access policies at the object level by using OPA. Gatekeeper is a Kubernetes admission controller that enforces policies created by OPA during application deployment. For more information, see Gatekeeper.

  In addition, OPA supports Layer 7 network policies and access control across namespaces based on labels and annotations. You can use OPA to manage Kubernetes network policies in an efficient manner.

- **Kyverno**

  Kyverno is a Kubernetes-native policy engine. Kyverno provides policies that can be used to validate, modify, and generate configurations for Kubernetes resources. Kyverno uses Kustomize-style overlays for validation and supports strategic merge patch for mutation. In addition, Kyverno can clone resources across namespaces based on flexible triggers. For more information, see Kyverno.

  You can use Kyverno to isolate namespaces, enforce pod security and other best practices, and generate default configurations, such as network policies. For more information, see Policy repository.

Hard multi-tenancy

You can implement hard multi-tenancy by creating a separate cluster for each tenant. Hard multi-tenancy provides strong isolation among tenants. The following section describes the disadvantages of hard multi-tenancy:

- The cost of hard multi-tenancy significantly increases if you manage a large number of tenants. You are charged a control plane fee for each cluster that you use. In addition, computing resources cannot be shared among clusters. As a result, fragmentation occurs in scenarios in which specific clusters are underutilized or overutilized.
- You may need to purchase or build special tooling to manage the clusters. Hundreds of clusters need to be managed over a long period of time.
- Compared with namespaces, clusters for tenants require a longer period of time to be created. Hard multi-tenancy is suitable for highly regulated industries and SaaS environments that require strong isolation.

Future trend

The Kubernetes community recognizes the disadvantages of soft multi-tenancy and the challenges with hard multi-tenancy. The Multi-Tenancy Special Interest Group (SIG) attempts to address the disadvantages by using several projects:

- The Virtual Cluster proposal describes a mechanism to create separate instances of the control plane services for each tenant in the cluster, including the API server, controller manager, and scheduler. The tenants in the cluster refer to Kubernetes on Kubernetes. For more information, visit Virtual Cluster.
- The Hierarchical Namespace Controller (HNC) proposal proposed in Kubernetes Enhancement Proposal (KEP) describes a way to create parent-child relationships between namespaces with policy object inheritance. HNC also allows tenant administrators to create subnamespaces. For more information, see HNC.
- The Multi-Tenancy Benchmarks proposal provides guidelines on how to share clusters after the clusters are isolated and segmented by using namespaces. The proposal also describes how to use the kubectl-mtb CLI to ensure compliance with the guidelines. For more information, see Multi-Tenancy Benchmarks.

Related information

- k-rail
- kiosk
- Loft
- DevSpace
- Multi-Tenancy Special Interest Group (SIG)

4.10. Host security

This topic describes how to ensure the security of hosts in a Container Service for Kubernetes (ACK) cluster.

Periodically check whether the configurations of your ACK cluster comply with CIS benchmarks and the baseline for classified protection of cybersecurity.

Center for Internet Security (CIS) is a third-party security organization that is committed to leading a global community of enterprises, public service sectors, and academia to provide security best practices.

The CIS Kubernetes Benchmark is written for open source Kubernetes distributions and intended to be applicable to all distributions as possible. Each CIS Kubernetes Benchmark version is tied to a specific Kubernetes release. For more information, see CIS Kubernetes Benchmark.

The generic version of the CIS Kubernetes Benchmark consists of items that are related to the control plane and data plane. Most cloud service providers manage and maintain the control plane of Kubernetes clusters. Therefore, the generic version of the CIS Kubernetes Benchmark is not suitable for these cloud service providers. To resolve this issue, Alibaba Cloud released the CIS ACK Benchmark in the CIS community. The CIS ACK Benchmark can be used to audit the security compliance of ACK clusters.

ACK clusters provided by Alibaba Cloud are optimized based on the CIS ACK Benchmark to support a stronger security posture. For more information, see Use security-inspector to audit the CIS Kubernetes Benchmark.

Most OS images released by cloud service providers for nodes in Kubernetes clusters are tied to specific CIS Benchmarks. These OS images include Alibaba Cloud Linux 2, CentOS, and Ubuntu. Alibaba Cloud Linux 2 is an OS image released by Alibaba Cloud and is used as the default OS image by ACK clusters. Alibaba Cloud Linux 2 was certified by CIS on August 16, 2019. Then, CIS released CIS Aliyun Linux 2 Benchmark version 1.0.0. For more information, see CIS Aliyun Linux 2 Benchmark version 1.0.0.

You can enhance the OS security of all nodes in an ACK cluster. For more information, see CIS reinforcement.

Alibaba Cloud issued baselines for classified protection of OS security based on Information security technology - Baseline for classified protection of cybersecurity (GB/T 22239-2019) issued by State Market Regulatory Administration and Standardization Administration of PRC. These baselines help ensure the security of Alibaba Cloud Linux.

You can use the following security reinforcement configurations to ensure that your ACK clusters comply with the required baselines:

- Identity authentication
- Access control
- Security audit
- Intrusion prevention
- Protection against malicious code execution

Use Alibaba Cloud Security Center to protect your ACK clusters

The following features of Alibaba Cloud Security Center can help ensure that the default configurations of the nodes in an ACK cluster are secure:

- Vulnerability patching: detects common vulnerabilities and allows you to patch the vulnerabilities with a few clicks. You can view detected vulnerabilities or manually run scan tasks on the Vulnerabilities page. This feature helps you identify vulnerabilities and potential risks in your assets.
- Baseline check: checks the configurations of server operating systems, databases, software, and containers, generates reports, and provides security suggestions. The baseline check feature can reinforce the security of your assets and reduce the risk of intrusions to comply with the baselines for classified protection.
- Cloud service configuration check: checks the configurations of cloud services based on identity authentication and permissions, network access control, data security, log audit, monitoring and

alerting, and basic security. Security Center also provides suggestions on how to mitigate the detected risks.

- Container image scan: detects and identifies high-risk system vulnerabilities, application vulnerabilities, malicious samples, configuration risks, and sensitive data in container images. Security Center also provides suggestions on how to handle these issues. Security Center simplifies how you can patch vulnerabilities for container images.

Limit access to nodes in an ACK cluster

If you want to access a remote node, you can log on to the and use Workbench or Virtual Network Computing (VNC) to access the node over the internal network. In this scenario, you do not need to associate an elastic IP address (EIP) with the node. If you want to access the node over the Internet, you must add access control list (ACL) rules to the security group of the ACK cluster to limit access to the node.

To further limit access to the node, you must modify the security group to limit access to the ports of the node that are exposed to the Internet.

Comply with the best practices to ensure ECS instance security

By default, Elastic Compute Service (ECS) instances that host the nodes of an ACK cluster run Alibaba Cloud Linux 2. For more information about how to reinforce the security of ECS instances, see Best practices for security.

Related information

- CIS reinforcement
-
- Use Alibaba Cloud Linux 2
- Vulnerability patching overview

5.Storage

5.1. Enable a StatefulSet to support persistent storage

You can create a pair of persistent volume (PV) and persistent volume claim (PVC) for each pod by setting the VolumeClaimTemplate in a StatefulSet. If pods are deleted or scaled in, PVs and PVCs of the StatefulSet application are not deleted. This topic describes how to enable a StatefulSet to support persistent storage by setting the VolumeClaimTemplate.

Prerequisites

Create an ACK managed cluster

Scenarios

When to use a StatefulSet:

- Predefined deployment order: Pods are deployed or scaled out from 0 to N-1 in sequence. The system must wait until all of the preceding pods reach the Running or Ready state before it can deploy another pod.
- Predefined scale-in order: Pods are scaled in or deleted from N-1 to 0 in sequence. The system must wait until all of the preceding pods are deleted before it can delete another pod.
- Consistent network identifiers: After a pod is rescheduled, its PodName and HostName values remain unchanged.
- Stable data persistence: After a pod is rescheduled, the pod can still access the same persisted data.

How to use a StatefulSet:

PVCs and PVs are automatically created based on the `VolumeClaimTemplates` .

Deploy a StatefulSet application

> **Note** volumeClaimTemplates: The system uses this template to create PVCs. The number of PVCs equals the number of replicas that are deployed for the StatefulSet application. The configurations of these PVCs are the same except for the PVC names.

1. Use the following template to create a file named *statefulset.yaml*.

   Deploy a Service and a StatefulSet, and provision two pods for the StatefulSet.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: disk-ssd
          mountPath: /data
  volumeClaimTemplates:
  - metadata:
      name: disk-ssd
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "alicloud-disk-ssd"
      resources:
        requests:
          storage: 20Gi
```

- `replicas`: the parameter is set to 2 in this example. This indicates that two pods are created.
- `mountPath`: the path where the disk is mounted in the container.
- `accessModes`: the access mode of the StatefulSet.
- `storageClassName`: the parameter is set to `alicloud-disk-ssd` in this example. This

indicates that an Alibaba Cloud standard SSD is used.

- `storage`: specifies the storage that is required by the application.

2. Run the following command to deploy the StatefulSet application:

```
kubectl create -f statefulset.yaml
```

3. Run the following command to view the deployed pods:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          6m
web-1                         1/1     Running   0          6m
```

4. Run the following command to view the PVCs:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   7m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   6m
```

Verify that the PVCs are scaled out together with the StatefulSet application

1. Run the following command to scale out the StatefulSet application to three pods:

```
kubectl scale sts web --replicas=3
```

Expected output:

```
statefulset.apps/web scaled
```

2. Run the following command to view the pods after the StatefulSet application is scaled out:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          34m
web-1                         1/1     Running   0          33m
web-2                         1/1     Running   0          26m
```

3. Run the following command to view the PVCs after the StatefulSet application is scaled out:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   35m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   34m
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
isk-ssd   27m
```

The output indicates that three PVCs are provisioned for the StatefulSet application after the StatefulSet application is scaled to three pods.

Verify that the PVCs remain unchanged after the StatefulSet application is scaled in.

1. Run the following command to scale the StatefulSet application to two pods:

   ```
   kubectl scale sts web --replicas=2
   ```

   Expected output:

   ```
   statefulset.apps/web scaled
   ```

2. Run the following command to view the pods after the StatefulSet application is scaled in:

   ```
   kubectl get pod
   ```

   Expected output:

   ```
   NAME                         READY   STATUS    RESTARTS   AGE
   web-0                        1/1     Running   0          38m
   web-1                        1/1     Running   0          38m
   ```

   Only two pods are deployed for the StatefulSet application.

3. Run the following command to view the PVCs after the StatefulSet application is scaled in:

   ```
   kubectl get pvc
   ```

   Expected output:

   ```
   NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
   SS        AGE
   disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
   isk-ssd   39m
   disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
   isk-ssd   39m
   disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
   isk-ssd   31m
   ```

   After the StatefulSet application is scaled to two pods, the StatefulSet application still has three PVCs. This indicates that the PVCs are not scaled in together with the StatefulSet application.

Verify that the PVCs remain unchanged when the StatefulSet application is scaled out again

When the StatefulSet is scaled out again, verify that the PVCs remain unchanged.

1. Run the following command to scale out the StatefulSet application to three pods:

   ```
   kubectl scale sts web --replicas=3
   ```

   Expected output:

   ```
   statefulset.apps/web scaled
   ```

2. Run the following command to view the pods after the StatefulSet application is scaled out:

   ```
   kubectl get pod
   ```

   Expected output:

   ```
   NAME                          READY   STATUS    RESTARTS   AGE
   web-0                         1/1     Running   0          1h
   web-1                         1/1     Running   0          1h
   web-2                         1/1     Running   0          8s
   ```

3. Run the following command to view the PVCs after the StatefulSet application is scaled out:

   ```
   kubectl get pvc
   ```

   Expected output:

   ```
   NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
   SS        AGE
   disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
   isk-ssd   1h
   disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
   isk-ssd   1h
   disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
   isk-ssd   1h
   ```

   The newly created pod uses an existing PVC.

Verify that the PVCs remain unchanged when the pod of the StatefulSet application is deleted

1. Run the following command to view the PVC that is used by the pod named *web-1*:

   ```
   kubectl describe pod web-1 | grep ClaimName
   ```

   Expected output:

   ```
   ClaimName:  disk-ssd-web-1
   ```

2. Run the following command to delete the pod named *web-1*:

   ```
   kubectl delete pod web-1
   ```

   Expected output:

   ```
   pod "web-1" deleted
   ```

3. Run the following command to view the pod:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          1h
web-1                         1/1     Running   0          25s
web-2                         1/1     Running   0          9m
```

The recreated pod uses the same name as the deleted pod.

4. Run the following command to view the PVCs:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   1h
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   1h
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
isk-ssd   1h
```

The recreated pod uses an existing PVC.

Verify that the StatefulSet application supports persistent storage

1. Run the following command to view the file in the */data* path:

```
kubectl exec web-1 ls /data
```

Expected output:

```
lost+found
```

2. Run the following command to create a file named *statefulset* in the */data* path:

```
kubectl exec web-1 touch /data/statefulset
```

3. Run the following command to view the file in the */data* path:

```
kubectl exec web-1 ls /data
```

Expected output:

```
lost+found
statefulset
```

4. Run the following command to delete the pod named *web-1*:

```
kubectl delete pod web-1
```

Expected output:

```
pod "web-1" deleted
```

5. Run the following command to view the file in the */data* path:

```
kubectl exec web-1 ls /data
```

Expected output:

```
lost+found
statefulset
```

The *statefulset* file still exists in the /data path. This indicates that data is persisted to the disk.

5.2. Use FlexVolume to enable a StatefulSet to support persistent storage

In a StatefulSet, you can create a pair of persistent volume (PV) and persistent volume claim (PVC) for each pod by configuring VolumeClaimTemplates. If pods are deleted or scaled in, PVs and PVCs of stateful applications are not deleted. This topic describes how to enable a StatefulSet to support persistent storage by configuring a VolumeClaimTemplate.

Background information

StatefulSets provides the following benefits:

- A stable deployment order: Pods are deployed in sequence from 0 to N-1. Before a pod is deployed, all its predecessors must be in the Running or Ready state.
- A stable scale-in order: Pods are scaled in sequence from N-1 to 0. Before a pod is deleted, all its predecessors must be deleted.
- Stable and unique network identifiers: After a pod is rescheduled to another node, the PodName and HostName values remain unchanged.
- Stable data persistence: After a pod is rescheduled, the pod can still access the same persisted data.

How to use a StatefulSet:

PVCs and PVs are automatically created based on `VolumeClaimTemplates`.

This topic describes how to manage a stateful application by performing the following operations:

- Deploy the stateful application
- Scale the stateful application
- Delete the stateful application
- Manage persistent storage of the stateful application

Prerequisites

- Create an ACK managed cluster
- Connect to ACK clusters by using kubectl

Deploy the stateful application

Note `volumeClaimTemplates`: The system uses this template to create PVCs. The number of PVCs equals the number of replicas that are deployed for the stateful application. The configurations of these PVCs are the same except for the PVC names.

1. Create a *statefulset.yaml* file.

   Note Set `storageClassName` to *alicloud-disk-ssd*, which specifies an Alibaba Cloud standard SSD.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: disk-ssd
          mountPath: /data
  volumeClaimTemplates:
  - metadata:
      name: disk-ssd
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "alicloud-disk-ssd"
      resources:
        requests:
          storage: 20Gi
```

2. Run the following command to deploy the stateful application:

```
kubectl create -f statefulset.yaml
```

3. Open another command-line interface (CLI) of kubectl and run the following command to check whether the pods are deployed in sequence:

```
kubectl get pod -w -l app=nginx
```

Expected output:

```
NAME          READY   STATUS    RESTARTS   AGE
web-0         0/1     Pending   0          0s
web-0         0/1     Pending   0          0s
web-0         0/1     ContainerCreating   0          0s
web-0         1/1     Running   0          20s
web-1         0/1     Pending   0          0s
web-1         0/1     Pending   0          0s
web-1         0/1     ContainerCreating   0          0s
web-1         1/1     Running   0          7s
```

4. Run the following command to view the deployed pods:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          6m
web-1                         1/1     Running   0          6m
```

5. Run the following command to view the PVCs:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   7m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   6m
```

Scale the stateful application

Scale out the stateful application

1. Run the following command to scale out the stateful application to three pods:

```
kubectl scale sts web --replicas=3
```

Expected output:

```
statefulset.apps/web scaled
```

2. Run the following command to view the pods after the stateful application is scaled out:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          34m
web-1                         1/1     Running   0          33m
web-2                         1/1     Running   0          26m
```

3. Run the following command to view the PVCs after the stateful application is scaled out:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   35m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   34m
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
isk-ssd   27m
```

Scale in the stateful application

1. Run the following command to scale in the stateful application to two pods:

```
kubectl scale sts web --replicas=2
```

Expected output:

```
statefulset.apps/web scaled
```

2. Run the following command to view the pods after the stateful application is scaled in:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          38m
web-1                         1/1     Running   0          38m
```

Only two pods are deployed for the stateful application.

3. Run the following command to view the PVCs after the stateful application is scaled in:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   39m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   39m
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
isk-ssd   31m
```

PVCs and PVs are not deleted together with the pods during the scale-in operation.

Scale out the stateful application again

1. Run the following command to scale out the stateful application to three pods:

```
kubectl scale sts web --replicas=3
```

Expected output:

```
statefulset.apps/web scaled
```

2. Run the following command to view the pods after the stateful application is scaled out:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          1h
web-1                         1/1     Running   0          1h
web-2                         1/1     Running   0          8s
```

3. Run the following command to view the PVCs after the stateful application is scaled out:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   1h
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   1h
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
isk-ssd   1h
```

The newly created pod uses the existing PVC and PV.

Delete the stateful application

1. Run the following command to view the PVC that is used by the pod named *web-1*:

```
kubectl describe pod web-1 | grep ClaimName
```

Expected output:

```
ClaimName:  disk-ssd-web-1
```

2. Run the following command to delete the pod named *web-1*:

```
kubectl delete pod web-1
```

Expected output:

```
pod "web-1" deleted
```

3. Run the following command to view the pod:

```
kubectl get pod
```

Expected output:

```
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          1h
web-1                         1/1     Running   0          25s
web-2                         1/1     Running   0          9m
```

The recreated pod has the same name as the deleted pod.

4. Run the following command to view the PVCs:

```
kubectl get pvc
```

Expected output:

```
NAME             STATUS   VOLUME                   CAPACITY   ACCESS MODES   STORAGECLA
SS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO            alicloud-d
isk-ssd   1h
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO            alicloud-d
isk-ssd   1h
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO            alicloud-d
isk-ssd   1h
```

The recreated pod uses the same PVC as the deleted pod.

5. Open another CLI of kubectl and run the following command to view how the pod is deleted and recreated:

```
kubectl get pod -w -l app=nginx
```

Expected output:

```
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          102m
web-1   1/1     Running   0          69s
web-2   1/1     Running   0          10m
web-1   1/1   Terminating   0     89s
web-1   0/1   Terminating   0     89s
web-1   0/1   Terminating   0     90s
web-1   0/1   Terminating   0     90s
web-1   0/1   Pending   0     0s
web-1   0/1   Pending   0     0s
web-1   0/1   ContainerCreating   0     0s
web-1   1/1   Running   0     20s
```

Manage persistent storage of the stateful application

1. Run the following command to view the files in the */data* path:

```
kubectl exec web-1 ls /data
```

Expected output:

```
lost+found
```

2. Run the following command to create a *statefulset* file in the */data* path:

```
kubectl exec web-1 touch /data/statefulset
```

3. Run the following command to view the files in the */data* path of each pod:

```
kubectl exec web-1 ls /data
```

Expected output:

```
lost+found
statefulset
```

4. Run the following command to delete the pod named *web-1*:

```
kubectl delete pod web-1
```

Expected output:

```
pod "web-1" deleted
```

5. Run the following command to verify that the file *statefulset* exists in the */data* path. This indicates that data is persisted to the disk.

```
kubectl exec web-1 ls /data
```

Expected output:

```
lost+found
statefulset
```

5.3. Use CNFS to automatically collect the heap dumps of a JVM

If your application is written in Java and the heap size of the Java virtual machine (JVM) is small, the application may encounter out of memory (OOM) errors. You can mount a Container Network File System (CNFS) volume to the log directory of your application. This way, the log that records OOM errors is automatically stored in the CNFS volume. This topic describes how to use CNFS to automatically collect the heap dumps of a JVM.

Prerequisites

- A Kubernetes cluster is created. The Container Storage Interface (CSI) plug-in is used as the volume plug-in. For more information, see Create an ACK managed cluster.
- An instance of Container Registry Enterprise Edition is created. For more information, see Create an instance of Container Registry Enterprise Edition.
- CNFS is used to manage Apsara File Storage NAS (NAS) file systems. For more information, see Use CNFS to manage NAS file systems.

Context

- CNFS allows you to abstract NAS file systems as custom Kubernetes objects by using the CustomResourceDefinition (CRD) resource. You can use the custom objects to create, delete, describe, mount, monitor, and expand NAS file systems. For more information, see CNFS overview.
- Container Registry is a secure platform that allows you to manage and distribute cloud-native artifacts that meet the standards of Open Container Initiative (OCI) in an effective manner. The artifacts include container images and Helm charts. For more information, see What is Container Registry?.

Considerations

- Set the Java argument Xmx to a value smaller than the memory limit of your application pod. This prevents the situations where OOM errors occur in the pod but not in the JVM.
- To collect the heap dumps of a JVM, we recommend that you mount a new CNFS volume. This way, your application data and the heap dumps are stored in separate CNFS volumes. This prevents the *.hprof* file from occupying excessive storage, which may cause a negative impact on your application.

Procedure

1. You can use the *registry.cn-hangzhou.aliyuncs.com/acs1/java-oom-test:v1.0* image to deploy a Java program that is used to trigger OOM errors in the JVM.

   For more information about how to build an image, see Use Container Registry Enterprise Edition instances to build images.

2. Use the following template to create a Deployment named *java-application*.

   When you launch the Mycode program, the heap size is set to 80 MB, and the heap dumps are written to the */mnt/oom/logs* directory. If the heap cannot meet the requirement of the JVM, a HeapDumpOnOutOfMemoryError error is returned.

```
cat << EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: java-application
spec:
  selector:
    matchLabels:
      app: java-application
  template:
    metadata:
      labels:
        app: java-application
    spec:
      containers:
      - name: java-application
        image: registry.cn-hangzhou.aliyuncs.com/acs1/java-oom-test:v1.0  # The image a
ddress of the sample Java application.
        imagePullPolicy: Always
        env:                                # Specify two environment variables. Set the
key of one variable to POD_NAME and the value to metadata.name. Set the key of the othe
r variable to POD_NAMESPACE and the value to metadata.namespace.
        - name: POD_NAME
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.name
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              apiVersion: v1
              fieldPath: metadata.namespace
        args:
        - java                              # Run the Java command.
        - -Xms80m                           # The minimum heap size.
        - -Xmx80m                           # The maximum heap size.
        - -XX:HeapDumpPath=/mnt/oom/logs  # The path in which heap dumps are stored whe
n OOM errors occur.
        - -XX:+HeapDumpOnOutOfMemoryError # Generate heap dumps when OOM errors occur.
        - Mycode                            # Run the Mycode program.
        volumeMounts:
        - name: java-oom-pv
          mountPath: "/mnt/oom/logs"      # Mount the CNFS volume to the /mnt/oom/logs
directory.
          subPathExpr: $(POD_NAMESPACE).$(POD_NAME)   # Create a subdirectory named $(P
OD_NAMESPACE).$(POD_NAME). The subdirectory is used to store heap dumps that are genera
ted due to OOM errors.
      volumes:
      - name: java-oom-pv
        persistentVolumeClaim:
          claimName: cnfs-nas-pvc         # The persistent volume claim (PVC) that is u
sed to mount the CNFS volume. The PVC name is cnfs-nas-pvc.
EOF
```

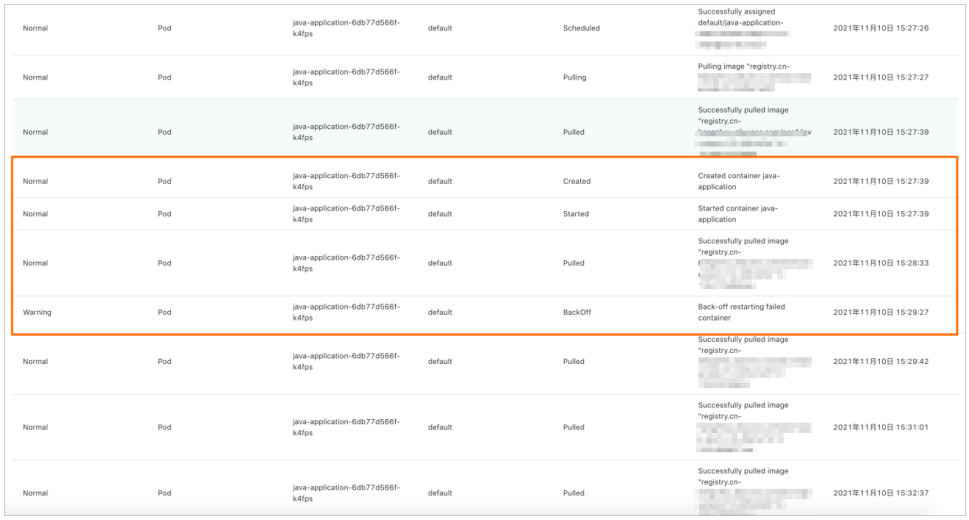
3. Go to the Event Center module of the Container Service for Kubernetes (ACK) console. If a Back-off restarting warning event appears on the page, an OOM error has occurred in the *java-application* application.

   i.

   ii.

   iii.

   iv. In the left-side navigation pane of the cluster details page, choose **Operations > Event Center**.



4. To view, upload, and download files in NAS file systems, you can deploy a File Browser application. This allows you to perform these operations on a web page. Mount the NAS file system to the *root Dir* path of the File Browser application. Then, run the `kubectl port-forward` command to map the container port of the File Browser application to your on-premises machine. This way, you can use your browser to access files in the NAS file system.

   i. Use the following template to create a ConfigMap that is used by File Browser and the File Browser Deployment. By default, port 80 is opened.

```
cat << EOF | kubectl apply -f -
apiVersion: v1
data:
  .filebrowser.json: |
    {
      "port": 80
    }
kind: ConfigMap
metadata:
  labels:
    app.kubernetes.io/instance: filebrowser
    app.kubernetes.io/name: filebrowser
  name: filebrowser
  namespace: default
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app.kubernetes.io/instance: filebrowser
```

```
    app.kubernetes.io/name: filebrowser
  name: filebrowser
  namespace: default
spec:
  progressDeadlineSeconds: 600
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app.kubernetes.io/instance: filebrowser
      app.kubernetes.io/name: filebrowser
  template:
    metadata:
      labels:
        app.kubernetes.io/instance: filebrowser
        app.kubernetes.io/name: filebrowser
    spec:
      containers:
      - image: docker.io/filebrowser/filebrowser:v2.18.0
        imagePullPolicy: IfNotPresent
        name: filebrowser
        ports:
        - containerPort: 80
          name: http
          protocol: TCP
        resources: {}
        securityContext: {}
        terminationMessagePath: /dev/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        - mountPath: /.filebrowser.json
          name: config
          subPath: .filebrowser.json
        - mountPath: /db
          name: rootdir
        - mountPath: /rootdir
          name: rootdir
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
      volumes:
      - configMap:
          defaultMode: 420
          name: filebrowser
        name: config
      - name: rootdir
        persistentVolumeClaim:
          claimName: cnfs-nas-pvc
EOF
```

Expected output:

```
configmap/filebrowser unchanged
deployment.apps/filebrowser configured
```

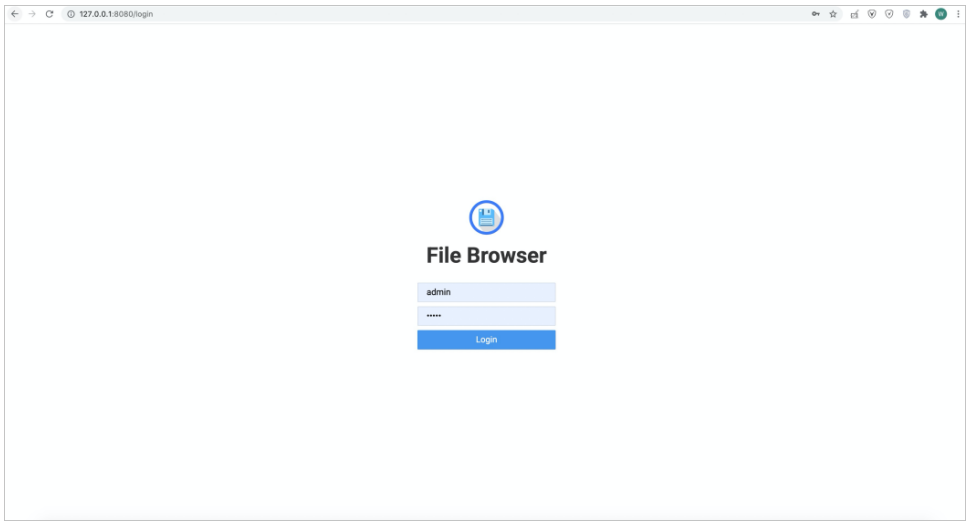
ii. Map port 80 of File Browser to your on-premises machine.

```
kubectl port-forward deployment/filebrowser 8080:80
```

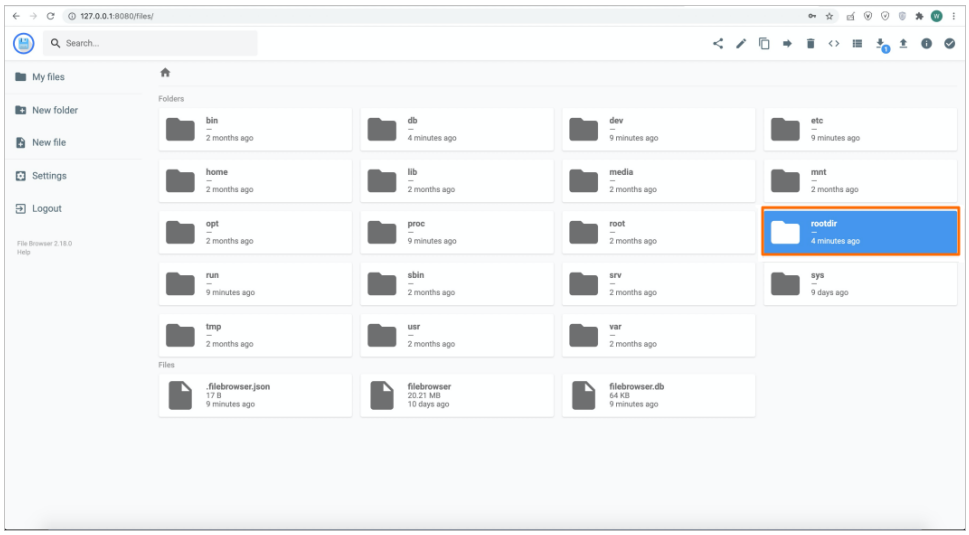
Expected output:

```
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

iii. Open your browser, enter *127.0.0.1:8080* in the address bar, and then press Enter. The File Browser logon page appears. Enter the default username (admin) and password (admin). Then, click Login.

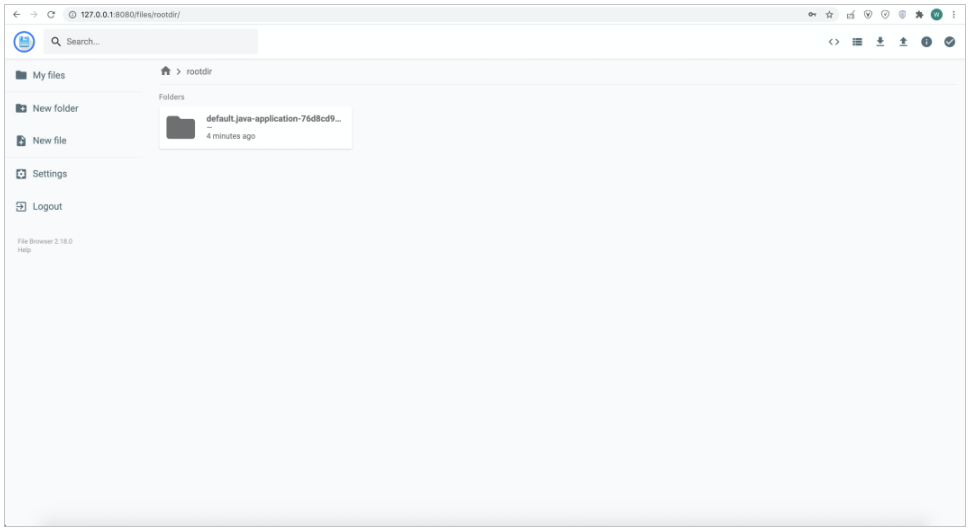


iv. The *cnfs-nas-pvc* PVC is mounted to the *rootDir* directory. Double-click *rootDir* to open the NAS file system.

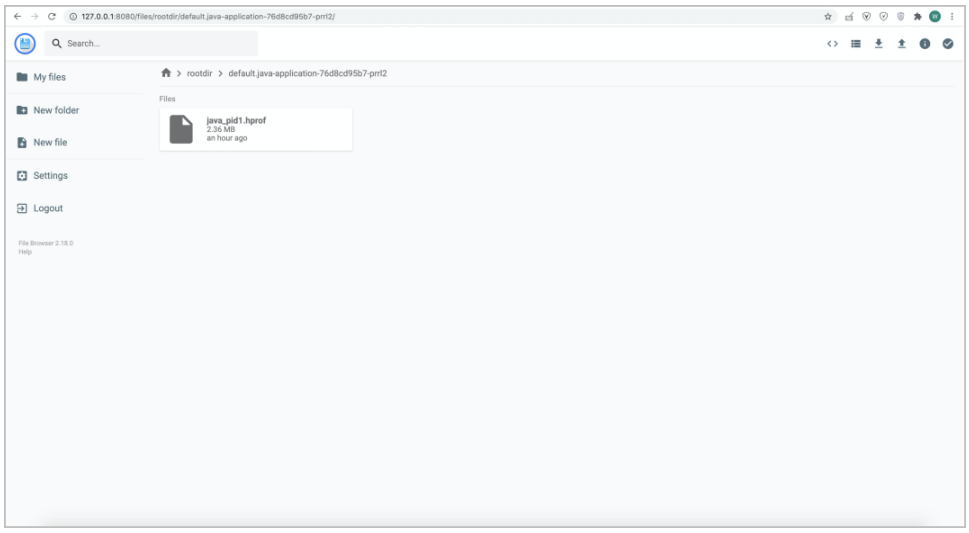


Result

On the File Browser page, find the *default.java-application-76d8cd95b7-prrl2* directory that is created for *java-application* and named based on the `subPathExpr: $(POD_NAMESPACE).$(POD_NAME)` configuration.



Navigate to this directory and find the heap dump file *java_pid1.hprof*. If you want to locate the exact line of code that triggers the OOM error, download *java_pid1.hprof* to your on-premises machine and use Eclipse Memory Analyzer Tool (MAT) to analyze the JVM stacks.



6.Operation and maintenance

6.1. Set up an LDAP authentication source for ACK

An increasing number of enterprise users of Alibaba Cloud Container Service for Kubernetes (ACK) require to use their own account systems in the cloud. However, the migration of these account systems to the cloud may not be a smooth process. The challenge is how to smoothly migrate their accounts to the cloud without registering a large number of new accounts and managing different username/password pairs. This topic describes how to set up a Lightweight Directory Access Protocol (LDAP) authentication source to map external users to Alibaba Cloud Resource Access Management (RAM) users.

Context

The following tools and Alibaba Cloud services are used in this topic:

- ACK.
- Resource Access Management (RAM).
- Identity as a service (IDaaS). Alibaba Cloud provides this service to integrate external account systems with the Alibaba Cloud account system.
- LDAP. LDAP stores the account information and for single sign-on (SSO). OpenLDAP is the open source implementation of LDAP.

How it works

The LDAP authentication source is integrated with the Alibaba Cloud account system through the following steps:

1. Add an LDAP authentication source in the IDaaS console, and synchronize LDAP accounts to IDaaS. The passwords are not synchronized. If you want LDAP users to log on to ACK with their LDAP accounts, you must set a password for each LDAP account in IDaaS. To avoid trouble of setting a password for each account, you can also configure single sign-on.
2. Add an application in the IDaaS console.
   - The application must be linked to a RAM user that has the AliyunRAMFullAccess permission so that the application has the full permissions to manage RAM. Therefore, you must specify the AccessKey ID and AccessKey secret of the RAM user in the application configuration.
   - The application must also be linked to another RAM role or RAM user that is used to grant the LDAP accounts the permissions to manage ACK .
   - You must import the LDAP accounts that have been synchronized to IDaaS to the application, and authorize these accounts to use the application. This enables the LDAP accounts to derive the required permissions from the RAM role or RAM user that is linked to the application.
3. Set up an LDAP authentication source for IDaaS SSO.
4. When LDAP users log on to the Alibaba Cloud Management Console through SSO, they derive the required permissions from the RAM role or RAM user that is linked to the application in IDaaS.
5. Grant the platform as a service (PaaS) permissions to the IDaaS account in the ACK console.

Step 1: Prepare the environment

Perform the following steps to build a staging environment for LDAP:

1. Run the following command to install OpenLDAP.

   You can use OpenLDAP to provision an LDAP data store and configure php-LDAP-admin to manage LDAP.

```
git clone https://github.com/lilongthinker/demo-base-code.git
# Output:
Cloning to 'demo-base-code'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 12 (delta 0), reused 9 (delta 0), pack-reused 0
Unpacking objects: 100% (12/12), done.
```

```
cd demo-base-code/01_ldap
01_ldap git:(master) tree . /
# Output:
./
├── ingress-phpadmin.yaml
├── ldap-deploy.yaml
├── ldap-secret.yaml
├── ldap-service.yaml
├── phpldapadmin-deploy.yaml
└── phpldapadmin-svc.yaml
0 directories, 6 files
################################################
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ldap-ui
  namespace: public-service
spec:
  rules:
  - host: phpldap.c26a7ab225d1544088735677ed906xxxx.cn-beijing.alicontainer.com # Repla
ce the value with the domain name of your cluster.
    http:
      paths:
      - backend:
          serviceName: phpldapadmin
          servicePort: 8080
## Run the vim command to replace the value of the host field with the domain name of y
our cluster.
################################################
```

```
01_ldap git:(master) kubectl create ns public-service
# Output:
namespace/public-service created
```

```
01_ldap git:(master) kubectl apply -f . /
# Output:
ingress.extensions/ldap-ui created
deployment.extensions/ldap created
secret/ldap-secret created
service/ldap-service created
deployment.extensions/phpldapadmin created
service/phpldapadmin created
```

2. Initialize LDAP accounts.
   i. Log on to php-LDAP-admin.
      a. Run the following command to query the domain name and IP address of the Ingress:

```
01_ldap_with_ui git:(master) ✗ kubectl get ing
```

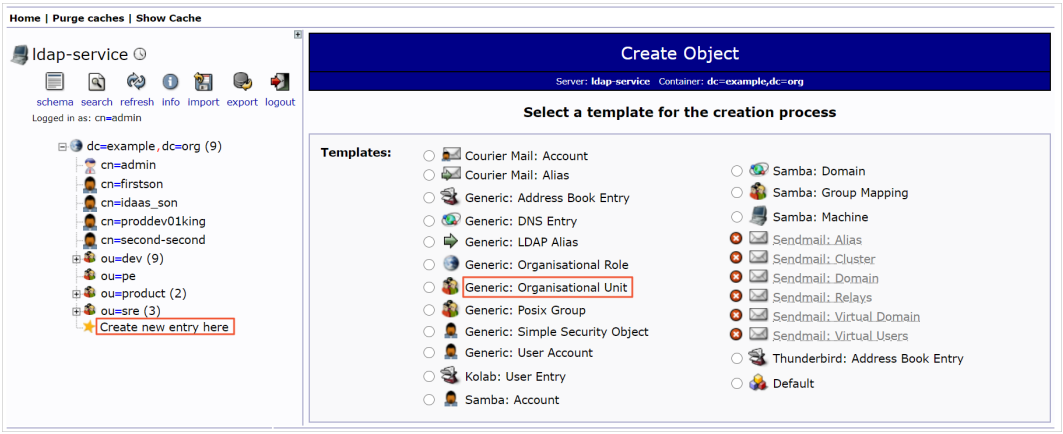
```
NAME      HOSTS
ADDRESS          PORTS   AGE
ldap-ui   phpldap.c26a7ab225d1544088735677ed906xxxx.cn-beijing.alicontainer.com
121.xx.xxx.xxx   80      45s
```

      b. Copy the domain name into the address bar of your browser and press Enter to open the console of php-LDAP-admin. Then, use the default distinguished name (DN) and password to log on to the console.

> **Note**
> - The default DN: cn=admin,dc=example,dc=org.
> - The default password: admin.

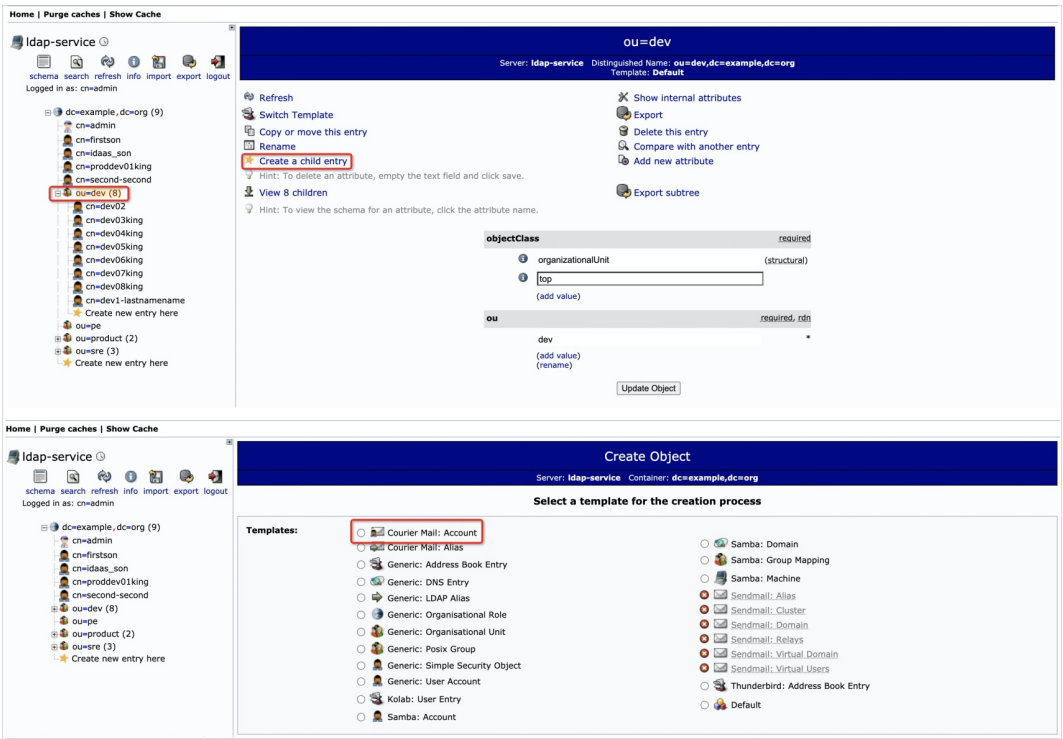
ii. Create organizations and accounts.

a. In the left-side navigation pane, click **Create new entry here**. On the **Create Object** page, click **Generic: Organisational Unit**, and then click **Create Object**.



b. Enter a name for the organization and click **Create Object**. In this example, the organization name is *dev*.

c. In the left-side navigation pane, click the newly created ou=dev. On the ou=dev page, click **Create a child entry**, and then click **Courier Mail: Account**. Set the parameters and click **Create Object**.



> **Notice** The default name of the created account may contain extra space characters. We recommend that you delete the space characters.

Step 2: Configure IDaaS

Perform the following steps to import the organizations and accounts from LDAP to IDaaS.

1. Log on to the IDaaS console.
2. Find and click your IDaaS instance.
3. In the left-side navigation pane, choose **Users > Organizations and Groups**.
4. In the **View Details** section, click **Configure LDAP**.
5. In the **Configure LDAP** pane, click **Create**.
6. Set the parameters that are required to configure LDAP. For more information about the parameters, see LDAP provision configuration.

   The following table describes some of the parameters.

   - Parameters for server connection

Parameter	Description
AD/LDAP Name	Enter a custom name.
Server Address	Enter the IP address of LDAP that is used for external access. Do not specify the domain name of LDAP.
Port Number	Enter the port number of LDAP that is used for external access. The port number is *389*.
Base DN	Enter the LDAP directory to be synchronized to IDaaS.
Administrator DN	Enter an LDAP administrator DN.
Password	Enter the LDAP administrator password.
Select Type	Select **Windows AD** or **OpenLDAP**.
From LDAP to IDaaS	Select **Enable**.
Provision from IDaaS to LDAP	You can select **Enable** or **Disable**.

   - Parameters for field matching rules

Parameter	Description
Username	A keyword of the username in LDAP. You can enter cn.
External ID	■ Enter *objectGUID* if you select the Windows AD type. ■ Enter *uid* if you select the OpenLDAP type.
Password Attribute	■ Enter *unicodePwd* if you select the Windows AD type. ■ Enter *userPassword* if you select the OpenLDAP type.
User Unique Identifier	■ Enter *DistinguishedName* if you select the Windows AD type. ■ Enter *EntryDN* if you select the OpenLDAP type.

Parameter	Description
Email	Enter a keyword, for example, *mail*.
Phone Number	Enter a keyword, for example, *telephoneNumber*.
Alias	Enter an alias for the synchronized account in IDaaS.
Default Password	The default password that is used to synchronize the account from Windows AD or OpenLDAP at a specified time.

> **Note**
>
> Note the following limits when you set the preceding parameters:
>
> - In the **Account** section, set **Administrator DN** and **Password** to the following values:
>   - Administrator DN: cn=admin,dc=example,dc=org
>   - Password: admin
> - After you have set all of the preceding parameters on the Server Connection and Field Matching Rule tabs, click **Test Connection** on both tabs. After the test succeeds, click **Save** on both tabs.
> - If you want to synchronize only specified accounts to Alibaba Cloud, specify **Base DN** as the root directory that you want to synchronize. If you want to synchronize multiple departments in different root directories, configure an LDAP data store for each department.

7. Import organizations and accounts. For more information, see Import organizations and accounts from Windows AD.

Step 3: Configure IDaaS SSO

1. Create an application.
   i. Go to the details page of your IDaaS instance. In the left-side navigation pane, choose **Applications > Add Applications**.
   ii. On the **Add Applications** page, find **plugin_aliyun_role** in the **Application ID** column and click **Add Application** in the Actions column.
   iii. In the pane that appears, click **Add SigningKey**. Set the parameters and click **Submit**.
   iv. Click **Select** in the Actions column to configure LDAP.

   > **Note** Enter the AccessKey ID and AccessKey secret of the RAM user that has the AliyunRAMFullAccess permission.

   v. Click **Submit**.
2. Authorize the application by organization.
   i. Go the details page of your IDaaS instance. In the left-side navigation, choose **Authorization > Application Authorization**.

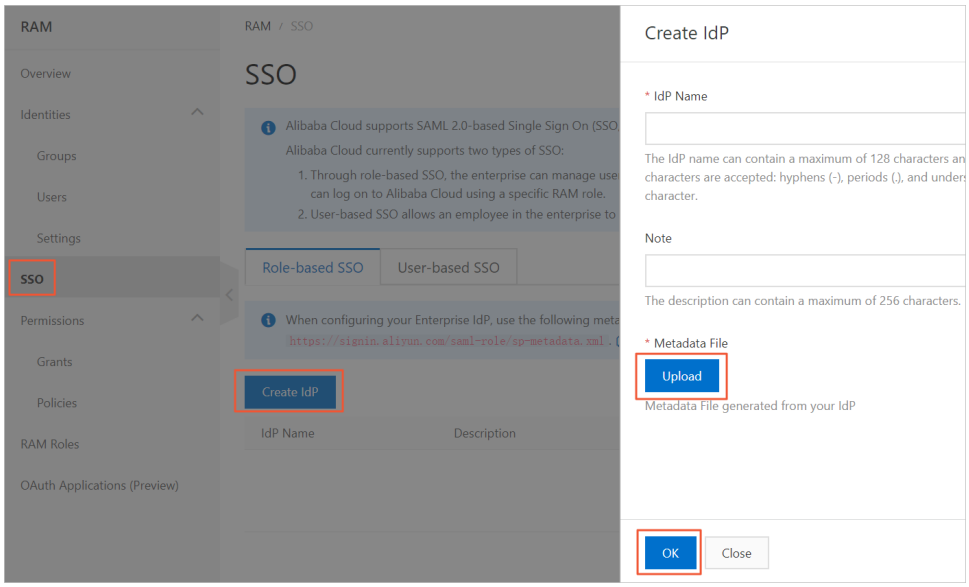
ii. On the **Authorize OUs or Groups by Application** tab, click the created application and organization.

iii. In the left-side navigation pane of the instance details page, choose **Application > Application List**. Find the created application and click **Details** in the Actions column. In the **Application Information** section, click **View Details**.

iv. In the **Application Details** pane, click **Export IDaaS SAML Meta Profile**.

v. Log on to the RAM console. In the left-side navigation pane, click **SSO**. On the **Role-based SSO** tab, click **Create IdP**.

vi. In the **Create IdP** pane, enter the IdP name, click **Upload** to upload the IDaaS SAML meta profile, and then click **OK**.



vii. Create a RAM role, specify **IdP** as the trusted entity of the RAM role, and add the AliyunRAMFullAccess permission. For more information, see Create a RAM role for a trusted IdP.

viii. In the left-side navigation pane of the IDaaS console, choose **Applications > Application List**. Find the application that you have created and click **Details** in the Actions column. In the **Account Information - Account Linking** section, click **View Application Accounts**.

ix. On the **Application Accounts** page, click **Link Accounts**.

x. In the **Application Accounts** pane, enter the IDaaS account and application account, and then click **Save**.

> **Note**
>
> The IDaaS account is the name of the LDAP username that is imported on the **Account** tab of the **Organizations and Groups** page.
>
> You can grant the application account the ACK fullAccess permission by linking the account to the corresponding RAM role or RAM user. If you want to link multiple application accounts to the RAM role or RAM user at a time, use a YAML file.

Step 4: Set up an LDAP authentication source for SSO in the IDaaS console

1. In the left-side navigation pane of the IDaaS console, choose **Authentication > Authentication sources**.
2. In the upper-right corner of the **Authentication sources** page, click **Add Authentication Source**.
3. Find **LDAP** and click **Add Authentication Source** in the Actions column.
4. Set the parameters and click **Submit**.

> **Notice** Select **Display**. If you do not select Display, the authentication source is not displayed in the login dialog box. For more information about the parameters, see LDAP as Authentication Source.

Step 5: Test SSO

Visit the User Login page to test SSO.

1. In the left-side navigation pane of the IDaaS console, click Instances and find your IDaaS instance on the Instances page.
2. Copy the URL in the User login page address column into the address bar of your browser and press Enter. In the login dialog box that appears, click the LDAP icon.
   For more information about setting up an LDAP authentication source, see LDAP as Authentication Source.

7.ESS

7.1. Use ack-autoscaling-placeholder to scale pods in seconds

ack-autoscaling-placeholder provides a buffer for the auto scaling of pods in a Container Service for Kubernetes (ACK) cluster. ack-autoscaling-placeholder is suitable for quickly launching pods for workloads without the need to worry whether node resources are sufficient. This topic describes how to use ack-autoscaling-placeholder to scale pods within seconds.

Prerequisites

Auto Scaling is enabled for your ACK cluster. For more information, see Auto scaling of nodes.

Procedure

1.
2.
3. On the **App Catalog** tab, find and click **ack-autoscaling-placeholder**.
4. On the **ack-autoscaling-placeholder** page, click **Deploy**.
5. In the **Deploy** wizard, select a **cluster** and a **namespace**, and then click **Next**. Select a **chart version**, configure the **parameters**, and then click **OK**.
   After ack-autoscaling-placeholder is deployed, go to the cluster details page. In the left-side navigation, choose **Applications > Helm**. You can find that the application is in the **Deployed** state.
6.
7. On the **Helm** page, find ack-autoscaling-placeholder and click **Update** in the **Actions** column. In the **Update Release** panel, modify the YAML template based on your requirements, and then click **OK**.

```
nameOverride: ""
fullnameOverride: ""
##
priorityClassDefault:
  enabled: true
  name: default-priority-class
  value: -1
##
deployments:
   - name: ack-place-holder
     replicaCount: 1
     containers:
       - name: placeholder
         image: registry-vpc.cn-shenzhen.aliyuncs.com/acs/pause:3.1
         pullPolicy: IfNotPresent
         resources:
           requests:
             cpu: 4                  # Occupy 4 vCPUs and 8 GiB of memory.
             memory: 8
     imagePullSecrets: {}
     annotations: {}
     nodeSelector:                   # Specify rules that are used to select nodes.
       demo: "yes"
     tolerations: []
     affinity: {}
     labels: {}
```

8. Create a PriorityClass for a workload.

   In this example, a PriorityClass that grants a high priority is created.

```
kubectl apply -f priorityClass.yaml
```

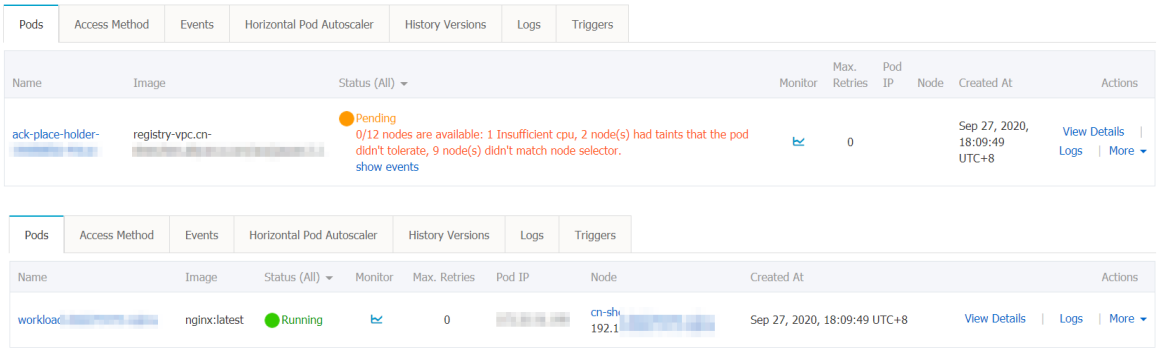
```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000              # Specify the priority.
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

9. Deploy a workload.

```
kubectl apply -f workload.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: placeholder-test
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector:                          # Specify rules that are used to select node
s.
        demo: "yes"
      priorityClassName: high-priority       # Specify the name of the PriorityClass that
you created in Step 8.
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 3                           # Specify the resource request of the worklo
ad.
            memory: 5
```

A PriorityClass that grants a higher priority than other pods is created for the pod of the workload, as shown in the following figure. When node resources are insufficient, the placeholder pod named placeHolder is evicted and changes to the Pending state. After the placeholder pod changes to the Pending state, a scale-out activity is triggered in the cluster because Auto Scaling is enabled for the cluster. Consequently, **a new pod is created within seconds** for the workload.



How it works

A placeholder pod with an extremely low priority (a negative value) is created to occupy a certain amount of computing resources for other pods with higher priorities. If the computing resources are insufficient, the placeholder pod is evicted to release the occupied computing resources for the workload. This way, pods can be launched in seconds. cluster-autoscaler is also used to scale nodes in the cluster.

7.2. Create custom images

Alicloud Image Builder is an image build tool provided by Alibaba Cloud that simplifies and automates image building. You can use OS images created by using Alicloud Image Builder as custom images to create node pools in Container Service for Kubernetes (ACK) clusters. This allows you to quickly add nodes to ACK clusters. This topic describes how to run Alicloud Image Builder as a Job to create custom OS images in ACK clusters.

Prerequisites

- An ACK cluster is created. For more information, see Create an ACK managed cluster.
- A kubectl client is connected to the cluster. For more information, see Connect to ACK clusters by using kubectl.

Context

The node pools in ACK clusters support auto scaling. By default, when you create a node pool, you can select OS images such as CentOS and Alibaba Cloud Linux 2. These OS images can meet the requirements of most scenarios. However, in scenarios that require preinstallation or high performance, these images may be unable to meet your requirements. Alibaba Cloud provides Alicloud Image Builder to help you build custom OS images and facilitate auto scaling in complex scenarios.

To use Alicloud Image Builder to create custom images, you can create a Job or a CronJob to distribute the image build task in the cluster.

Create a Job to quickly build a custom OS image

In this example, a ConfigMap named build-config and a Job named build are created to show how to use Alicloud Image Builder to quickly build a custom OS image.

1. Create a ConfigMap named build-config to specify the parameters for the OS image.
    i. Create a YAML file named *build-config.yaml* and add the following content to the file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: build-config
data:
  ack.json: |-
    {
      "variables": {
        "region": "{{env `REGION`}}",
        "image_name": "ack-custom_image",
        "source_image": "centos_7_02_64_20G_alibase_20170818.vhd",
        "access_key": "{{env `ALICLOUD_ACCESS_KEY`}}",
        "secret_key": "{{env `ALICLOUD_SECRET_KEY`}}"
      },
      "builders": [
        {
          "type": "alicloud-ecs",
          "access_key": "{{user `access_key`}}",
          "secret_key": "{{user `secret_key`}}",
          "region": "{{user `region`}}",
          "image_name": "{{user `image_name`}}",
          "source_image": "{{user `source_image`}}",
          "ssh_username": "root",
          "instance_type": "ecs.g6.large",
          "skip_image_validation": "true",
          "io_optimized": "true"
        }
      ],
      "provisioners": [{
        "type": "shell",
        "inline": [
                    "sleep 30"
        ]
      }]
    }
```

The following table describes the parameters in the YAML file.

Alicloud Image Builder parameters

Parameter	Example	Description
`variables{"<variable1>":"<value>"}`	variables{"access_key":"{{env ALICLOUD_ACCESS_KEY}}"}	The `variables` that are used by Alicloud Image Builder. **Note** If you write sensitive information such as AccessKey pairs (`access_key` and `secret_key`) into the configuration file, such information may be leaked. To ensure data security, you can specify an AccessKey pair as variables. The values of the variables are based on the input values of the runtime.

Parameter	Example	Description
`builders{"type":"<value>"}`	builders{"type":"alicloud-ecs"}	The image `builders`. When type is set to *aliyun-ecs*, a temporary Elastic Compute Service (ECS) instance is created to build the image. The ECS instance is automatically released after the image is built.
`provisioners{"type":"<value>"}`	provisioners{"type":"shell"}	The image `provisioners` that are used to specify the operations that need to be performed on the temporary instance. When type is set to *shell*, a shell provisioner is used. A shell command is automatically run after the Linux instance is connected. For example, you can run the `yum install redis.x86_64 -y` command to install Redis. For more information about how to configure provisioners, see Provisioner configuration.

Image build parameters

Parameter	Example	Description	Importance
access_key	LTAInPyXXXXQ****	The AccessKey ID of your account. For more information, see Obtain an AccessKey pair.	High
secret_key	CM1ycKrrCekQ0dhXXXXXXXXXl7y****	The AccessKey secret of your account.	High
region	cn-beijing	The region where the custom image is to be created.	High
image_name	ack-custom_image	The name of the custom image to be created. The name must be globally unique.	Low
source_image	aliyun_2_1903_x64_20G_alibase_20200904.vhd	The ID of the Alibaba Cloud public image based on which the custom image is created. The created custom image will contain the same operating system as the public image.	High
PRESET_GPU	true	The preinstalled GPU that is used to accelerate startup.	High

ii. Run the following command to deploy Alicloud Image Builder in the cluster:

```
kubectl apply -f build-config.yaml
```

2. Create a Job to build a custom OS image.

i. Create a YAML file named *build.yaml* and add the following content to the file:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: image-builder
  namespace: default
spec:
  template:
    metadata:
      name: image-builder
    spec:
      containers:
        - name: image-builder
          image: registry.cn-hangzhou.aliyuncs.com/acs/image-builder:v3
          env:
            - name: ALICLOUD_ACCESS_KEY
              value: xxxxxxx
            - name: ALICLOUD_SECRET_KEY
              value: xxxxxxx
            - name: REGION
              value: cn-hangzhou
          command: ["packer"]
          args:  ["build","/config/ack.json"]
          volumeMounts:
            - name: config
              mountPath: /config
      volumes:
        - name: config
          configMap:
            name: build-config
            items:
              - key: ack.json
                path: ack.json
      restartPolicy: Never
```

ii. Run the following command to deploy the Job and start building the image:

```
kubectl apply -f build.yaml
```

3. (Optional)Log on to the ACK console and check the image build log.

A log is generated during the image build process. This log records all the image build operations, including checking parameters, creating temporary resources, pre-installing software, creating target resources, and releasing temporary resources. You can check the image build log by performing the following steps:

i.

ii.

iii.

iv.

v. On the Jobs page, find the Job that you created and click **Details** in the **Actions** column.

vi. On the Job details page, click the **Logs** tab to check the image build log.

Provisioner configuration

A provisioner is a component used to install and configure software in a running operating system before the operating system is packaged into an OS image. A provisioner is often used to install software in images in the following scenarios:

- Install software.
- Patch kernels.
- Create users.
- Download application code.

Common operations by using provisioners include:

- Execute shell scripts.

```
"provisioners": [{
    "type": "shell",
    "script": "script.sh"
}]
```

- Execute orchestration scripts by using Ansible.

```
"provisioners": [
  {
  "type": "ansible",
  "playbook_file": "./playbook.yml"
  }
]
```

- Install the Cloud Paralleled File System (CPFS) client.

  The installation of CPFS requires multiple installation packages, some of which involve real-time compilation and may require a long time to install. The use of a custom image can greatly reduce the cost of installing the CPFS client on a large number of nodes. The following code block provides a sample configuration.

```
    {
      "variables": {
        "region": "{{env `REGION`}}",
        "image_name": "ack-custom_image",
        "source_image": "centos_7_04_64_20G_alibase_201701015.vhd",
        "access_key": "{{env `ALICLOUD_ACCESS_KEY`}}",
        "secret_key": "{{env `ALICLOUD_SECRET_KEY`}}"
      },
    "builders": [
        {
          "type": "alicloud-ecs",
          "access_key": "{{user `access_key`}}",
          "secret_key": "{{user `secret_key`}}",
          "region": "{{user `region`}}",
          "image_name": "{{user `image_name`}}",
          "source_image": "{{user `source_image`}}",
          "ssh_username": "root",
          "instance_type": "ecs.g6.large",
          "skip_image_validation": "true",
          "io_optimized": "true"
        }
      ],
   "provisioners": [{
        "type": "shell",
        "inline": [
            "cd $HOME",
            "wget https://cpfs-client.oss-cn-beijing.aliyuncs.com/kernel/kernel-devel-`un
ame -r`.rpm",
            "rpm -ivh --replacefiles kernel-devel-`uname -r`.rpm"
        ]
      }]
```

- Customize the OS image of a GPU-accelerated node.

```
    {
      "variables": {
        "region": "{{env `REGION`}}",
        "image_name": "ack-custom_image",
        "source_image": "aliyun_2_1903_x64_20G_qboot_alibase_20210325.vhd",
        "access_key": "{{env `ALICLOUD_ACCESS_KEY`}}",
        "secret_key": "{{env `ALICLOUD_SECRET_KEY`}}"
      },
    "builders": [
        {
          "type": "alicloud-ecs",
          "access_key": "{{user `access_key`}}",
          "secret_key": "{{user `secret_key`}}",
          "region": "{{user `region`}}",
          "image_name": "{{user `image_name`}}",
          "source_image": "{{user `source_image`}}",
          "ssh_username": "root",
          "instance_type": "ecs.gn6i-c4g1.xlarge",   #Specify the type of the GPU-acceler
ated node for preinstalling the GPU.
          "skip_image_validation": "true",
          "io_optimized": "true"
        }
      ],
   "provisioners": [
        {
            "type": "file",
            "source": "scripts/ack-optimized-os-1.20.sh",
            "destination": "/root/"
        },
        {
            "type": "shell",
            "inline": [
                "export RUNTIME=containerd",
                "export INSTANCE_TYPE=ecs.c6.xlarge",
                "export PRESET_GPU=true",          #Set PRESET_GPU to true for preinstall
ing the GPU.
                "export SKIP_SECURITY_FIX=true",
                "bash /root/ack-optimized-os-1.20.sh"
            ]
        }
      ]
    }
```

What to do next

After a custom image is created by using Alicloud Image Builder, you can create an elastic node pool based on the custom image to quickly add nodes to the cluster. For more information about how to create an elastic node pool, see Auto scaling of nodes.

Related information

- Create a Kubernetes cluster by using a custom image

7.3. Configure auto scaling for cross-zone deployment

You can deploy an application across zones to improve the availability of the application. If an application that is deployed across zones does not have sufficient resources to handle heavy workloads, you may want Container Service for Kubernetes (ACK) to create a specific number of nodes in each zone of the application. This topic describes how to configure auto scaling for cross-zone deployment.

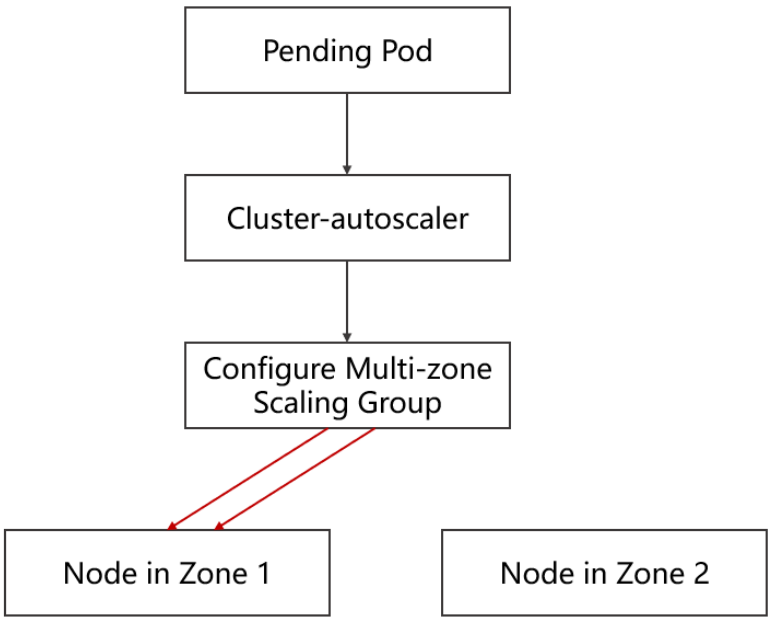
Prerequisites

Multiple zones are selected and vSwitches are created in the zones. You must create at least one vSwitch in the zone where you want to create nodes. For more information about how to select zones and create vSwitches, see Create an ACK managed cluster.

Context

The node auto scaling component provided by ACK checks whether an application can be deployed in a specific scaling group by using prescheduling. Then, the component sends a scale-out request to the scaling group and the scaling group creates the requested number of nodes. The following issues may occur if you configure vSwitches of multiple zones for a scaling group:

If application pods in multiple zones cannot be scheduled due to insufficient cluster resources, the node auto scaling component sends a request to the scaling group to trigger scale-out activities. However, the scaling group may not be able to create nodes in each zone that requires more nodes. Instead, the scaling group may create nodes only in specific zones. This does not meet the requirement of auto scaling for cross-zone deployment.



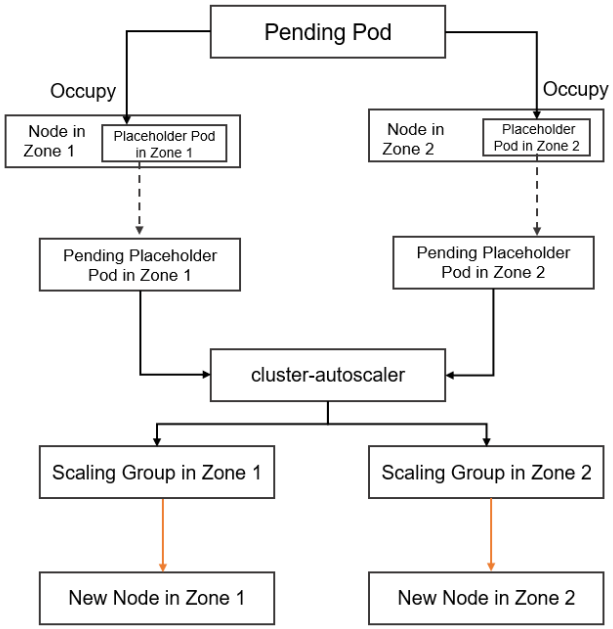
Solution

To meet the requirement of auto scaling for cross-zone deployment, ACK provides the ack-autoscaling-placeholder component. The component resolves this issue by using resource redundancy. The component can scale out node pools in different zones concurrently instead of creating nodes in specific zones. For more information, see Use ack-autoscaling-placeholder to scale pods in seconds.

The following section describes how to configure auto scaling for cross-zone deployment:

1. Create a node pool in each zone and add a label to the node pool. The label specifies the zone in which the node pool is deployed.
2. Configure the nodeSelector to schedule pods based on zone labels. This way, ack-autoscaling-placehodler can schedule a placeholder pod to each zone. By default, the priority of placeholder pods is lower than that of application pods.
3. This allows pending application pods to replace placeholder pods. After the pending application pods replace the placeholder pods that are scheduled by using the nodeSelector, the placeholder pods become pending. The node scheduling policy that is used by the node auto scaling component is changed from anti-affinity to nodeSelector. This way, the node auto scaling component can create nodes in each zone of the application concurrently.

The following figure shows how to create nodes in two zones concurrently based on the existing architecture.



1. Use ack-autoscaling-placeholder as the bridge between the application pods and the node auto scaling component and create a placeholder pod in each zone. The priority of the placeholder pods must be lower than the priority of the application pods.
2. After the application pods change to the Pending state, the application pods replace the placeholder pods and are scheduled to the existing nodes of each zone. The placeholder pods change to the Pending state.
3. The placeholder pods are scheduled by using the nodeSelector. The node auto scaling component must create nodes in each zone to host the placeholder pods.

Step 1: Create a node pool that has auto scaling enabled in each zone and add a custom label to the node pool

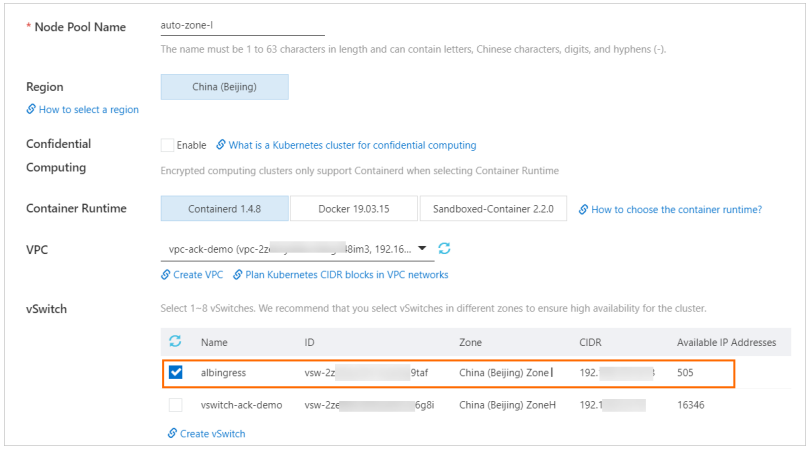
1.

2.
3.
4.
5. In the upper-right corner of the **Node Pools** page, click **Create Node Pool**.

   You can also click **Create Managed Node Pool** in the upper-right corner of the **Node Pools** page.
6. In the **Create Node Pool** dialog box, configure the node pool.

   In this example, the auto-zone-I node pool that has auto scaling enabled is created in Zone I. For more information about the parameters, see Create a node pool.

   i. Select a vSwitch deployed in Zone I.



   ii. Select **Enable Auto Scaling**.

   iii. Click **Show Advanced Options**. In the **Node Label** section, set **Key** to *avaliable_zone* and **Value** to *i*.

   iv. Configure other parameters for the node pool and click **Confirm Order**. In the dialog box that appears, click **Confirm**.
7. On the **Node Pools** page, you can find the node pools that you create. After the status of the auto-zone-I node pool changes to **Active**, the node pool is created.
8. Repeat the operations in Substep 6 to create node pools that have auto scaling enabled in other zones that require auto scaling.

Step 2: Deploy ack-autoscaling-placeholder and placeholder Deployments

1.
2. On the **App Catalog** tab, find and click **ack-autoscaling-placeholder**.
3. On the **ack-autoscaling-placeholder** page, click **Deploy**.
4. In the **Deploy** wizard, select a **cluster** and **namespace**, and then click **Next**. Select a **chart version**, configure the **parameters**, and then click **OK**.

After ack-autoscaling-placeholder is deployed, go to the cluster details page. In the left-side navigation, choose **Applications > Helm**. You can find that the application is in the **Deployed** state.

5.

6. On the **Helm** page, find ack-autoscaling-placeholder-defalut and click **Update** in the **Actions** column.

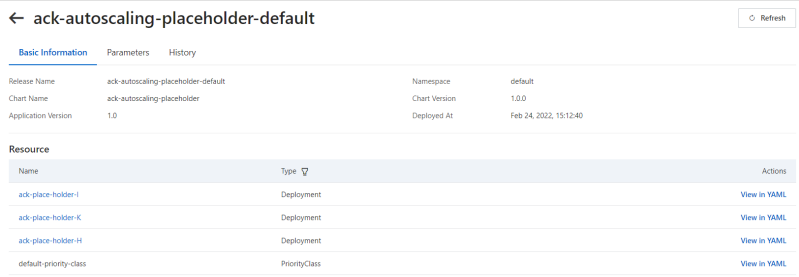
7. In the **Update Release** panel, modify the YAML template based on your requirements and click **OK**. Deploy a placeholder Deployment in each zone.

   The following YAML template provides an example on how to deploy a placeholder Deployment in Zone I, Zone K, and Zone H:

```
deployments:
- affinity: {}
  annotations: {}
  containers:
  - image: registry-vpc.cn-beijing.aliyuncs.com/acs/pause:3.1
    imagePullPolicy: IfNotPresent
    name: placeholder
    resources:
      requests:
        cpu: 3500m     #The CPU request of the placeholder Deployment.
        memory: 6      #The memory request of the placeholder Deployment.
  imagePullSecrets: {}
  labels: {}
  name: ack-place-holder-I             #The name of the placeholder Deployment.
  nodeSelector: {"avaliable_zone":i}   #The zone label. The label must be the same as t
he label that you specified in Step 1 when you created the node pool.
  replicaCount: 10                     #The number of pods that are created in each sca
le-out activity.
  tolerations: []
- affinity: {}
  annotations: {}
  containers:
  - image: registry-vpc.cn-beijing.aliyuncs.com/acs/pause:3.1
    imagePullPolicy: IfNotPresent
    name: placeholder
    resources:
      requests:
        cpu: 3500m    #The CPU request of the placeholder Deployment.
        memory: 6     #The memory request of the placeholder Deployment.
  imagePullSecrets: {}
  labels: {}
  name: ack-place-holder-K            #The name of the placeholder Deployment.
  nodeSelector: {"avaliable_zone":k}  #The zone label. The label must be the same as th
e label that you specified in Step 1 when you created the node pool.
  replicaCount: 10                    #The number of pods that are created in each scal
e-out activity.
  tolerations: []
- affinity: {}
  annotations: {}
  containers:
  - image: registry-vpc.cn-beijing.aliyuncs.com/acs/pause:3.1
    imagePullPolicy: IfNotPresent
```

```
    imagePullPolicy: IfNotPresent
    name: placeholder
    resources:
      requests:
        cpu: 3500m   #The CPU request of the placeholder Deployment.
        memory: 6    #The memory request of the placeholder Deployment.
  imagePullSecrets: {}
  labels: {}
  name: ack-place-holder-H           #The name of the placeholder Deployment.
  nodeSelector: {"avaliable_zone":h} #The zone label. The label must be the same as the
label that you specified in Step 1 when you created the node pool.
  replicaCount: 10                   #The number of pods that are created in each scale
-out activity.
  tolerations: []
fullnameOverride: ""
nameOverride: ""
podSecurityContext: {}
priorityClassDefault:
  enabled: true
  name: default-priority-class
  value: -1
```

After you update the YAML file, a placeholder Deployment is deployed in each zone.



Step 3: Create a PriorityClass for a workload

1. Create a file named priorityClass.yaml by using the following YAML template:

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000              #Specify the priority value. The value must be higher than
the default priority value of the Deployments that you create in Step 2.
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

If you do not want to configure a PriorityClass for a workload, you can configure a global PriorityClass as the default configuration. After you deploy the configurations, application pods can automatically replace placeholder pods.

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: global-high-priority
value: 1                              #Specify the priority value. The value must be hi
gher than the default priority value of the Deployments that you create in Step 2.
globalDefault: true
description: "This priority class should be used for XYZ service pods only."
```

2. Run the following command to deploy the PriorityClass for the workload:

```
kubectl apply -f priorityClass.yaml
```

Expected output:

```
priorityclass.scheduling.k8s.io/high-priority created
```

Step 4: Deploy a workload

In this example, a workload is deployed in Zone I.

1. Create a file named *workload.yaml* by using the following YAML template:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: placeholder-test
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      nodeSelector:                         # Select a node to deploy the workload.
        avaliable_zone: "i"
      priorityClassName: high-priority     #Specify the name of the PriorityClass that
is created in Step 3. If you enable global configurations, this parameter is not requir
ed.
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
        resources:
          requests:
            cpu: 3                          #Specify the resource request of the workloa
d.
            memory: 5
```

2. Run the following command to deploy the workload:

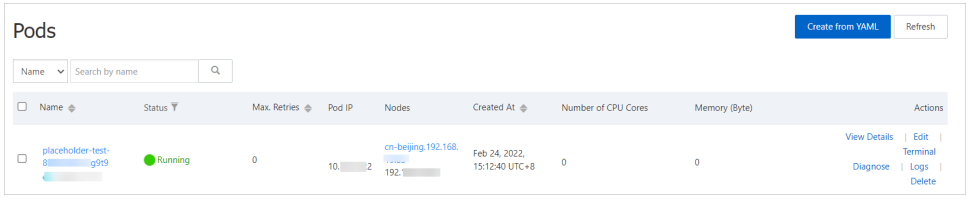
```
kubectl apply -f workload.yaml
```

Expected output:

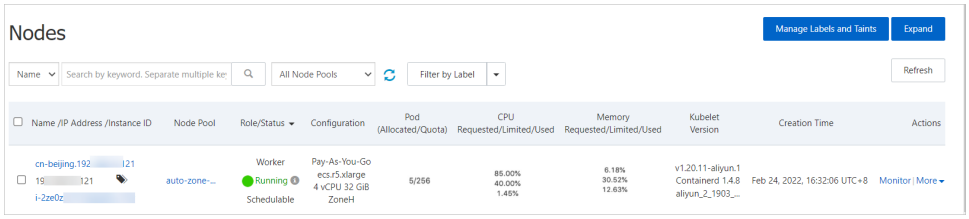
```
deployment.apps/placeholder-test created
```

Verify the result

After you deploy the workload, go to the cluster details page and choose **Workloads > Pods**. On the Pods page, you can find that the PriorityClass of the workload has a higher priority value than that of the placeholder pods. This way, the workload pods can run on the nodes that are added. The placeholder pods trigger the node auto scaling component to create nodes in each zone concurrently and prepare for the next scale-out activity requested by the workload.



Choose **Nodes > Nodes**. On the Nodes page, you can find that the workload pod runs on the node that hosts the placeholder pod.



Related information

- Use ack-autoscaling-placeholder to scale pods in seconds

7.4. Horizontal pod scaling based on ARMS Prometheus metrics

By default, Horizontal Pod Autoscaler (HPA) supports only auto scaling based on the CPU and memory usage. This cannot meet the O&M requirements. Application Real-Time Monitoring Service (ARMS) Prometheus is a fully managed monitoring service that is interfaced with the open source Prometheus ecosystem. ARMS Prometheus monitors a wide array of components and provides multiple ready-to-use dashboards. This topic describes how to convert ARMS Prometheus metrics to metrics that are supported by HPA. This way, HPA can perform auto scaling based on ARMS Prometheus metrics.

Prerequisites

Before you convert ARMS Prometheus metrics to metrics that are supported by HPA, you must install the following components:

1. Install the ARMS Prometheus component. For more information, see Enable ARMS Prometheus.
2. Install the alibaba-cloud-metrics-adapter component. For more information, see Deploy alibaba-cloud-metrics-adapter.

Example

This example shows how to configure alibaba-cloud-metrics-adapter to convert ARMS Prometheus metrics to metrics that are supported by HPA. This example also shows how to configure HPA to perform auto scaling based on ARMS Prometheus metrics.

1. Deploy a workload.
   i.
   ii.
   iii.
   iv. On the **Deployments** page, click **Create from YAML** in the upper-right corner.
   v. On the **Create** page, deploy an application named sample-app and a Service by using a YAML template and click **Create**.

      > **Note** The application pod is used to expose the http_requests_total metric, which indicates the number of requests.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  labels:
    app: sample-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - image: luxas/autoscale-demo:v0.1.2
        name: metrics-provider
        ports:
        - name: http
          containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: sample-app
  namespace: default
  labels:
    app: sample-app
spec:
  ports:
    - port: 8080
      name: http
      protocol: TCP
      targetPort: 8080
  selector:
    app: sample-app
  type: ClusterIP
```

2. Create a ServiceMonitor.
   i. Log on to the ARMS console.
   ii. In the left-side navigation pane, choose **Prometheus Monitoring > Prometheus Instances**.
   iii. In the upper-left corner of the **Prometheus Monitoring** page, select the region in which your Container Service for Kubernetes (ACK) cluster is deployed and click the Prometheus instance that you want to manage. Then, you are redirected to the instance details page.
   iv. In the left-side navigation pane, click **Service Discovery**. Then, click the **Configure** tab.
   v. On the **Configure** tab, click the **ServiceMonitor** tab.

vi. On the **ServiceMonitor** tab, click **Add ServiceMonitor**.

In this example, the following template is used to create a ServiceMonitor.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: sample-app
  namespace: default
spec:
  endpoints:
  - interval: 30s
    port: http
    path: /metrics
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      app: sample-app
```

3. Confirm the status of ARMS Prometheus.
Go to the Service Discovery page and click the **Targets** tab. If the tab displays **default/sample-app/0(1/1 up)**, ARMS Prometheus is monitoring the application that you deployed.
4. Modify the configurations of alibaba-cloud-metrics-adapter.
   i.
   ii. On the **Clusters** page, find the cluster that you want to manage and click its name or click **Details** in the **Actions** column.
   iii. In the left-side navigation pane of the cluster details page, choose **Applications > Helm**.
   iv. On the **Helm** page, find alibaba-cloud-metrics-adapter in the **Release Name** column and click **Update** in the Actions column.
   v. Copy the YAML content in alibaba-cloud-metrics-adapter to the editor and click **OK**.

   The following code block shows some of the YAML content:

```
AlibabaCloudMetricsAdapter:
  affinity: {}
  commonLabels: ""
  env:
  - AccessKeyId: ""
  - AccessKeySecret: ""
  - Region: ""
  fullnameOverride: ""
  image:
    pullPolicy: IfNotPresent
    # The public image address. If you want to specify a private image address, rep
lace the prefix with registry-vpc.{{RegionId}}.aliyuncs.com.
    repository: registry.aliyuncs.com/acs/alibaba-cloud-metrics-adapter-amd64
    tag: v0.2.0-alpha-2f697ee
  listenPort: 443
  nameOverride: ""
  nodeSelector: {}
  podAnnotations: {}
  prometheus:
```

```
    adapter:
      rules:
        custom:
        # Add a conversion rule. Make sure that the labels of the ARMS Prometheus m
etrics are the same as the labels that are specified in the conversion rule. If the
labels are not the same, modify the labels of the ARMS Prometheus metrics.
        - seriesQuery: http_requests_total{namespace!="",pod!=""}
          resources:
            overrides:
              # The resource field specifies an API resource of Kubernetes. You can
run the kubectl api-resources -o wide command to view the Kubernetes API resources.

              # The key field specifies the LabelName of the ARMS Prometheus metric
. Make sure that the ARMS Prometheus metric uses the specified LabelName.
              namespace: {resource: "namespace"}
              pod: {resource: "pod"}
          name:
            matches: ^(.*)_total
            as: ${1}_per_second
          metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.Group
By>>)
        default: false
    #1. Set the enabled field to true to enable the ARMS Prometheus adapter.
    enabled: true
    logLevel: 2
    metricsRelistInterval: 1m
    tls:
      ca: ""
      certificate: ""
      enable: false
      key: ""
    #2. Specify the endpoint of the ARMS Prometheus API.  For more information, see
the Obtain the Prometheus API endpoint section.
    url: http://arms-prometheus-proxy.aliyun.com:9090/api/v1/prometheus/8cba801fff6
5546a3012e9a6843afd/1240538168824185/ce63742a509f948dda8ef18e75e703356/cn-shenzhen
  ramRoleType: restricted
  replicas: 1
  service:
    type: ClusterIP
  serviceAccountName: admin
  tolerations: []
ConfigReloader:
  image:
    # The public image address. If you want to specify a private image address, rep
lace the prefix with registry-vpc.{{RegionId}}.aliyuncs.com.
    repository: registry.aliyuncs.com/acs/configmap-reload
    tag: v0.0.1
```

> **Note**
>
> - `url`: Specify the endpoint of the ARMS Prometheus API. For more information, see Obtain the Prometheus API endpoint.
> - For more information about the configuration file of ack-alibaba-cloud-adapter, see ack-alibaba-cloud-adapter configurations.

5. Deploy HPA.

   HPA allows you to use ARMS Prometheus metrics to expose customer metrics and external metrics. You can configure HPA to perform auto scaling based on customer metrics or external metrics.

   - **Method 1: Configure HPA to perform auto scaling based on custom metrics**

     a. Run the following command to query detailed information about the custom metrics that are supported by HPA:

     ```
     kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/*/
     http_requests_per_second" | jq .
     ```

     Returned result:

     ```
     {
       "kind": "MetricValueList",
       "apiVersion": "custom.metrics.k8s.io/v1beta1",
       "metadata": {
         "selfLink": "/apis/custom.metrics.k8s.io/v1beta1/namespaces/default/pods/%2A/
     http_requests_per_second"
       },
       "items": [
         {
           "describedObject": {
             "kind": "Pod",
             "namespace": "default",
             "name": "sample-app-579bc6774c-rmjfd",
             "apiVersion": "/v1"
           },
           "metricName": "http_requests_per_second",
           "timestamp": "2022-01-28T08:42:58Z",
           "value": "33m",
           "selector": null
         }
       ]
     }
     ```

b. Use the following YAML content to deploy HPA. Then, run the `kubectl apply -f hpa.yaml` command to create an HPA application.

```
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v2beta1
metadata:
  name: sample-app
spec:
#Describe the object that you want HPA to scale. HPA can dynamically change the n
umber of pods that are deployed for the object.
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: sample-app
#Specify the upper limit and lower limit of pods.
  minReplicas: 1
  maxReplicas: 10
#Specify the metrics based on which HPA performs auto scaling. You can specify di
fferent types of metrics at the same time.
  metrics:
  - type: Pods
    pods:
      #Specify the metric pods/http_requests.
      metricName: http_requests_per_second
 # Specify an AverageValue type threshold. You can specify only AverageValue type
thresholds for Pods metrics.
      targetAverageValue: 500m       #If the value contains decimal places and AC
K requires high precision, you can use the m or k unit. For example, 1001m is equ
al to 1.001 and 1k is equal to 1000.
```

c. Perform a stress test after you enable load balancing for the Service.

a. Run the following command to perform a stress test:

```
ab -c 50 -n 2000 LoadBalancer(sample-app):8080/
```

b. Run the following command to query information about HPA:

```
kubectl get hpa sample-app
```

Expected output:

```
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS
AGE
sample-app   Deployment/sample-app   33m/500m   1         10        1
7m
```

- **Method 2: Configure HPA to perform auto scaling based on external metrics**

a. Run the following command to query detailed information about the external metrics that are supported by HPA:

```
kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1/namespaces/default/http_requests_per_second" | jq .
```

Returned result:

```
{
  "kind": "ExternalMetricValueList",
  "apiVersion": "external.metrics.k8s.io/v1beta1",
  "metadata": {},
  "items": [
    {
      "metricName": "http_requests_per_second",
      "metricLabels": {},
      "timestamp": "2022-01-28T08:40:20Z",
      "value": "33m"
    }
  ]
}
```

b. Use the following YAML content to deploy HPA. Then, run the `kubectl apply -f hpa.yaml` command to deploy an HPA application.

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: sample-app
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: sample-app
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: External
      external:
        metric:
          name: http_requests_per_second
          selector:
            matchLabels:
              job: "sample-app"
#You can specify only Value and AverageValue type thresholds for external metrics.
        target:
          type: AverageValue
          averageValue: 500m
```

c. Perform a stress test after you enable load balancing for the Service.

a. Run the following command to perform a stress test:

```
ab -c 50 -n 2000 LoadBalancer(sample-app):8080/
```

b. Run the following command to query information about HPA:

```
kubectl get hpa sample-app
```

Expected output:

```
NAME         REFERENCE                TARGETS    MINPODS   MAXPODS   REPLICAS
AGE
sample-app   Deployment/sample-app   33m/500m   1         10        1
7m
```

ack-alibaba-cloud-adapter configurations

ack-alibaba-cloud-adapter performs the following steps to convert ARMS Prometheus metrics to metrics that are supported by HPA:

1. Discovery: discovers ARMS Prometheus metrics that can be used by HPA.
2. Association: associates the metrics with Kubernetes resources, such as pods, nodes, and namespaces.
3. Naming: defines the names of the converted metrics for HPA.
4. Querying: defines the template of the requests that are sent to the ARMS Prometheus API.

In the preceding example, the `http_requests_total` metric that is exposed by the sample-app pod is converted to the `http_requests_per_second` metric for HPA. The following code block shows the configurations of ack-alibaba-cloud-adapter that is used in the example:

```
- seriesQuery: http_requests_total{namespace!="",pod!=""}
  resources:
    overrides:
      namespace: {resource: "namespace"}
      pod: {resource: "pod"}
  name:
    matches: ^(.*)_total
    as: ${1}_per_second
  metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)
```

> **Note**
>
> - `seriesQuery`: the Prometheus Query Language (PromQL) query data.
> - `metricsQuery`: aggregates the PromQL query data in `seriesQuery`.
> - `resources`: maps the labels in the PromQL query data to resources. The resources field specifies Kubernetes API resources such as pods, namespaces, and nodes. You can run the `kubectl api-resources -o wide` command to query Kubernetes API resources. The `key` field specifies the `LabelName` of the ARMS Prometheus metric. Make sure that the ARMS Prometheus metric uses the specified LabelName.
> - `name`: uses a regular expression to convert the name of the ARMS Prometheus metric to a name that is easy to identify. In this example, the metric name is converted from `http_request_total` to `http_request_per_second`.

- Discovery

  Specify the ARMS Prometheus metric that you want to convert. You can use seriesFilters to filter metrics. You can use seriesQuery to search metrics by label, as shown in the following code block:

  ```
  seriesQuery: http_requests_total{namespace!="",pod!=""}
  seriesFilters:
      - isNot: "^container_.*_seconds_total"
  ```

  > **Note** seriesFilters is optional. You can use seriesFilters to filter metrics:
  >
  > - is: matches metrics that contain <regex>.
  > - isNot: matches metrics that do not contain <regex>.

- Association

  Map the labels of ARMS Prometheus metrics to Kubernetes resources The labels of the `http_requests_total` metric are `kubernetes_namespace!=""` and `kubernetes_pod_name!=""`.

  ```
  - seriesQuery: http_requests_total{namespace!="",pod!=""}
    resources:
      overrides:
        namespace: {resource: "namespace"}
        pod: {resource: "pod"}
  ```

- Naming

  Name the HPA metrics that are converted from the ARMS Prometheus metrics. The names of the ARMS Prometheus metrics remain unchanged. You do not need to configure the naming settings if you directly use the ARMS Prometheus metrics.

  > **Note** You can run the `kubectl get --raw "/apis/custom.metrics.k8s.io/v1beta1"` command to query the metrics that are supported by HPA.

```
- seriesQuery: http_requests_total{namespace!="",pod!=""}
  resources:
    overrides:
      namespace: {resource: "namespace"}
      pod: {resource: "pod"}
  name:
    matches: "^(.*)_total"
    as: "${1}_per_second"
```

- Querying

  The template of requests that are sent to the ARMS Prometheus API. ack-alibaba-cloud-adapter passes parameters in HPA to the request template, sends a request to the ARMS Prometheus API based on the template, and then sends the returned parameter values to HPA for auto scaling.

```
- seriesQuery: http_requests_total{namespace!="",pod!=""}
  resources:
    overrides:
      namespace: {resource: "namespace"}
      pod: {resource: "pod"}
  name:
    matches: ^(.*)_total
    as: ${1}_per_second
  metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m])) by (<<.GroupBy>>)
```

Obtain the Prometheus API endpoint

For ARMS Prometheus

1. Log on to the ARMS console.
2. In the left-side navigation pane, choose **Prometheus Monitoring > Prometheus Instances**.
3. In the upper-left corner of the **Prometheus Monitoring** page, select the region in which your ACK cluster is deployed and click the Prometheus instance that you want to manage. Then, you are redirected to the instance details page.
4. In the left-side navigation pane, click **Settings**. On the page that appears, click the **Settings** tab.
5. On the **Settings** tab, record the HTTP API endpoint. This endpoint is used to import monitoring data to Grafana.

   We recommend that you call the ARMS Prometheus API over an internal network. You can call the API over the Internet if no internal network is available.

For open source Prometheus

1.
2. Check **service** and **namespace** in the **Labels** column.

   If ServiceName is ack-prometheus-operator-prometheus and ServiceNamespace is monitoring, the endpoint of the open source Prometheus API is:

   ```
   http://ack-prometheus-operator-prometheus.monitoring.svc.cluster.local:9090
   ```

Related information

- Enable ARMS Prometheus

- HPA

7.5. Configure horizontal pod autoscaling for multiple applications based on the metrics of the NGINX Ingress controller

You can deploy an application in multiple pods to improve application stability. However, this method increases the resource cost and causes resource waste during off-peak hours. You can also manually scale the pods of your application. However, this method increases your O&M workload and pods cannot be scaled in real time. To resolve the preceding issues, you can configure horizontal pod autoscaling for multiple applications based on the metrics of the NGINX Ingress controller. This method improves the stability of applications and reduces the O&M cost. This topic describes how to configure horizontal pod autoscaling for multiple applications based on the metrics of the NGINX Ingress controller.

Prerequisites

To configure horizontal pod autoscaling for multiple applications based on the metrics of the NGINX Ingress controller, you must convert the Prometheus metric to a metric that is supported by the Horizontal Pod Autoscaler (HPA) and deploy the required components.

- Install the ARMS Prometheus component. For more information, see Enable ARMS Prometheus.
- Install the alibaba-cloud-metrics-adapter component. For more information, see Deploy alibaba-cloud-metrics-adapter.
- The stress testing tool Apache Benchmark is installed. For more information, see Apache Benchmark.

Context

To automatically scale the pods of an application based on the number of requests in a production environment, you can use the http_requests_total metric to collect the number of requests. We recommend that you configure horizontal pod autoscaling based on the metrics of the NGINX Ingress controller.

An Ingress is a Kubernetes API object. An Ingress forwards client requests to Services based on the hosts and URL paths of the requests. Then, the Services route the requests to the backend pods.

The NGINX Ingress controller is deployed in a Container Service for Kubernetes (ACK) cluster to control the Ingresses in the cluster. The NGINX Ingress controller provides high-performance and custom traffic management. The NGINX Ingress controller provided by ACK is developed based on the open source version and is integrated with various features of Alibaba Cloud services to provide a simplified user experience.

Procedure

In this topic, two ClusterIP Services are created to forward the external requests received by the NGINX Ingress controller. In the following example, the HPA is configured to automatically scale pods based on the `nginx_ingress_controller_requests` metric, which indicates the traffic loads.

1. Use the following YAML template to create a Deployment and a Service.

i. Create a file named *nginx1.yaml* based on the following content. Then, run the `kubectl apply -f nginx1.yaml` command to create an application named test-app and a Service also named test-app.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: test-app
  labels:
    app: test-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: test-app
  template:
    metadata:
      labels:
        app: test-app
    spec:
      containers:
      - image: skto/sample-app:v2
        name: metrics-provider
        ports:
        - name: http
          containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: test-app
  namespace: default
  labels:
    app: test-app
spec:
  ports:
    - port: 8080
      name: http
      protocol: TCP
      targetPort: 8080
  selector:
    app: test-app
  type: ClusterIP
```

ii. Create a file named *nginx2.yaml* based on the following content. Then, run the `kubectl apply -f nginx2.yaml` command to create an application named sample-app and a Service also named sample-app.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  labels:
    app: sample-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      containers:
      - image: skto/sample-app:v2
        name: metrics-provider
        ports:
        - name: http
          containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: sample-app
  namespace: default
  labels:
    app: sample-app
spec:
  ports:
    - port: 80
      name: http
      protocol: TCP
      targetPort: 8080
  selector:
    app: sample-app
  type: ClusterIP
```

2. Create a file named *ingress.yaml* based on the following content. Then, run the `kubectl apply -f ingress.yaml` command to create an Ingress.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: test-ingress
  namespace: default
spec:
  ingressClassName: nginx
  rules:
    - host: test.cf6828835dbba4a368e67f0b9925d****.cn-qingdao.alicontainer.com
      http:
        paths:
          - backend:
              service:
                name: sample-app
                port:
                  number: 80
            path: /
            pathType: ImplementationSpecific
          - backend:
              service:
                name: test-app
                port:
                  number: 8080
            path: /home
            pathType: ImplementationSpecific
```

- host: the domain name that is used to access the applications. In this example, the default domain name of your cluster is used. The default domain name is in the `*.[cluster-id].[region-id].alicontainer.com` format. Replace `cluster-id` and `region-id` with the actual values.
- path: the URL paths that are used to match requests. The requests received by the Ingress are matched against the Ingress rules and forwarded to the corresponding Service. Then, the Service routes the requests to the backend pods.
- backend: the name and port of the Service. The information specifies the Service to which the requests that match the `path` parameter are forwarded.

3. Run the following command to query the Ingress:

```
kubectl get ingress -o wide
```

Expected output:

```
NAME           CLASS   HOSTS
ADDRESS        PORTS   AGE
test-ingress   nginx   test.cf6828835dbba4a368e67f0b9925d****.cn-qingdao.alicontainer.c
om   10.10.10.10   80      55s
```

4. After you deploy the preceding resource objects, you can send requests to the `/` and `/home` URL paths to access the specified host. The NGINX Ingress controller automatically routes your requests to the test-app and sample-app applications based on the URL paths of the requests. You can obtain information about the requests to each application from the Prometheus metric `nginx_ingress_controller_requests`, as shown in the following figure.

```
nginx_ingress_controller_requests{canary="",controller_class="k8s.io/ingress-nginx",controller_namespace="kube-system",controller_pod="nginx-ingress-
controller-58ffffbccd-hnvg6",host="test.cf6828835dbba4a368e67[redacted].cn-qingdao.alicontainer.com",ingress="test-
ingress",method="GET",namespace="default",path="/",service="sample-app",status="200"} 3
nginx_ingress_controller_requests{canary="",controller_class="k8s.io/ingress-nginx",controller_namespace="kube-system",controller_pod="nginx-ingress-
controller-58ffffbccd-hnvg6",host="test.cf6828835dbba4a368e67f0b9[redacted].cn-qingdao.alicontainer.com",ingress="test-
ingress",method="GET",namespace="default",path="/home",service="test-app",status="200"} 1
```

5. Modify the *adapter.config* file of the alibaba-cloud-metrics-adapter component to convert the Prometheus metric to a metric that is supported by the HPA.

   > **Note** Before you modify the adapter.config file, make sure that the alibaba-cloud-metrics-adapter component is installed in your cluster.

   i.
   ii.
   iii.
   iv. In the left-side navigation pane, choose **Applications > Helm**.
   v. Click the release named **ack-alibaba-cloud-metrics-adapter**.
   vi. In the **Resource** section, click the **adapter-config** ConfigMap.
   vii. On the **adapter-config** page, click **Edit** in the upper-right corner.
   viii. Replace the content in the **Value** column with the following content. Then, click **OK**.

   For more information about how to configure the ConfigMap, see ack-alibaba-cloud-adapter configurations.

```
rules:
- metricsQuery: sum(rate(<<.Series>>{<<.LabelMatchers>>}[2m]))
  name:
    as: ${1}_per_second
    matches: ^(.*)_requests
  resources:
    namespaced: false
    overrides:
      controller_namespace:
        resource: namespace
  seriesQuery: nginx_ingress_controller_requests
```

6. Run the following command to query a metric:

```
kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1/namespaces/*/nginx_ingress_con
troller_per_second" | jq .
```

   Expected output:

```
{
  "kind": "ExternalMetricValueList",
  "apiVersion": "external.metrics.k8s.io/v1beta1",
  "metadata": {},
  "items": [
    {
      "metricName": "nginx_ingress_controller_per_second",
      "metricLabels": {},
      "timestamp": "2022-03-31T10:11:37Z",
      "value": "0"
    }
  ]
}
```

7. Create a file named *hpa.yaml* based on the following content. Then, run the `kubectl apply -f hpa.yaml` command to configure the HPA for both the sample-app and test-app applications.

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: sample-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: sample-app
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: External
      external:
        metric:
          name: nginx_ingress_controller_per_second
          selector:
            matchLabels:
# You can configure this parameter to filter metrics. The value of this parameter is pa
ssed to the <<.LabelMatchers>> field in the adapter.config file.
              service: sample-app
# You can specify only Value and AverageValue type thresholds for external metrics.
        target:
          type: AverageValue
          averageValue: 30
----------
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: test-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: test-app
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: External
      external:
        metric:
          name: nginx_ingress_controller_per_second
          selector:
            matchLabels:
# You can configure this parameter to filter metrics. The value of this parameter is pa
ssed to the <<.LabelMatchers>> field in the adapter.config file.
              service: test-app
# You can specify only Value and AverageValue type thresholds for external metrics.
        target:
          type: AverageValue
          averageValue: 30
```

8. Run the following command to query the HPA information:

```
kubectl get hpa
```

Expected output:

```
NAME          REFERENCE                TARGETS       MINPODS   MAXPODS   REPLICAS   AGE
sample-hpa    Deployment/sample-app    0/30 (avg)    1         10        1          74s
test-hpa      Deployment/test-app      0/30 (avg)    1         10        1          59m
```

9. After you configure the HPA, perform stress tests to check whether the pods of the applications are automatically scaled out when the number of requests increases.
   i. Run the following command to perform stress tests on the `/home` URL path of the host:

   ```
   ab -c 50 -n 5000 test.cf6828835dbba4a368e67f0b9925d****.cn-qingdao.alicontainer.com
   /home
   ```

   ii. Run the following command to query the HPA information:

   ```
   kubectl get hpa
   ```

   Expected output:

   ```
   NAME          REFERENCE                TARGETS           MINPODS   MAXPODS   REPLICAS
   AGE
   sample-hpa    Deployment/sample-app    0/30 (avg)        1         10        1
   22m
   test-hpa      Deployment/test-app      22096m/30 (avg)   1         10        3
   80m
   ```

   iii. Run the following command to perform stress tests on the root path of the host:

   ```
   ab -c 50 -n 5000 test.cf6828835dbba4a368e67f0b9925d****.cn-qingdao.alicontainer.com
   /
   ```

   iv. Run the following command to query the HPA information:

   ```
   kubectl get hpa
   ```

   Expected output:

   ```
   NAME          REFERENCE                TARGETS           MINPODS   MAXPODS   REPLICAS
   AGE
   sample-hpa    Deployment/sample-app    27778m/30 (avg)   1         10        2
   38m
   test-hpa      Deployment/test-app      0/30 (avg)        1         10        1
   96m
   ```

   The output shows that the pods of the applications are automatically scaled out when the number of requests exceeds the scaling threshold.

8.Istio

8.1. Use ASM to manage applications in ECI pods running on virtual nodes of ACK clusters

Container Service provides various serverless containers based on virtual nodes and Elastic Container Instance (ECI). For example, virtual nodes can be deployed in Container Service for Kubernetes (ACK) clusters to enable seamless integration of Kubernetes and ECI. You can create ECI pods as needed to avoid the planning of cluster capacity. This topic describes how to use Alibaba Cloud Service Mesh (ASM) to manage applications in ECI pods that run on the virtual nodes of ACK clusters.

Prerequisites

- The version of your ASM instance is v1.7.5.41-ge61a01c3-aliyun or later.
- The ack-virtual-node component is deployed and properly runs in your ACK cluster. For more information, see Step 1: Deploy ack-virtual-node in ACK clusters.
- The ACK cluster is added to your ASM instance. For more information, see Add a cluster to an ASM instance.

Enable automatic sidecar injection

After you enable automatic sidecar injection for a namespace in the ASM console, an Envoy proxy is automatically injected as a sidecar into each pod that is created in the namespace. These Envoy proxies comprise the data plane of the ASM instance.

1.
2.
3.
4.
5. On the **Namespaces** page, find the namespace for which you want to enable automatic sidecar injection and click **Enable Automatic Sidecar Injection** in the **Automatic Sidecar Injection** column. In this example, automatic sidecar injection is enabled for the namespaces that are named default and vk. If no namespaces are available, you must create at least two namespaces.
6. In the **Submit** message, click **OK**.

Create an application that runs in an ECI pod

> **Note** After an application is created and run in an ECI pod, ASM can manage the application on the data plane by using sidecars.

Use a pod label to create an application that runs in an ECI pod

You can add the `label alibabacloud.com/eci=true` label to a pod to make the pod an ECI pod that runs on a virtual node.

1. Run the following command to check whether the `istio-injection=enabled` label is added to

the namespace that is named default:

```
kubectl get ns default --show-labels
```

The following output is expected:

```
NAME      STATUS   AGE   LABELS
default   Active   84d   istio-injection=enabled,provider=asm
```

2. Run the following command to create an NGINX application:

```
kubectl run nginx -n default --image nginx -l alibabacloud.com/eci=true
```

3. Run the following command to view the information about the pods that run on virtual nodes:

```
kubectl get pod -n default -o wide|grep virtual-kubelet
```

Use a namespace label to create an application that runs in an ECI pod

You can add the `label alibabacloud.com/eci=true` label to the namespace of a pod to make the pod an ECI pod that runs on a virtual node.

1. Run the following command to check whether the `istio-injection=enabled` label is added to the vk namespace:

```
kubectl get ns default --show-labels
```

The following output is expected:

```
NAME      STATUS   AGE   LABELS
default   Active   84d   istio-injection=enabled,provider=asm
```

2. Run the following command to add a label to the vk namespace:

```
kubectl label namespace vk alibabacloud.com/eci=true
```

3. Run the following command to create an NGINX application:

```
kubectl -n vk run nginx --image nginx
```

4. Run the following command to view the information about the pods that run on virtual nodes:

```
kubectl -n vk get pod -o wide|grep virtual-kubelet
```

8.2. Use ASM to manage applications in registered external Kubernetes clusters

Alibaba Cloud Service Mesh (ASM) allows you to manage applications in external Kubernetes clusters that are registered in the Container Service console.

Prerequisites

- An external Kubernetes cluster that can access the Internet is registered in the Container Service

console. For more information, see Register an external Kubernetes cluster.

- ASM is activated. For more information, see Create an ASM instance.

Procedures

1.
2.
3. On the **Mesh Management** page, click **Create ASM Instance**.
4. In the **Create ASM Instance** panel, enter an instance name, and select a region, a virtual private cloud (VPC), and a vSwitch.

   > **Note**
   > - Select the region where the registered external Kubernetes cluster resides or a region that is nearest to the cluster.
   > - Select the VPC where the registered external Kubernetes cluster resides.
   > - Select a vSwitch from the vSwitch drop-down list as required. If no vSwitch is available, click **Create vSwitch** to create one. For more information, see Work with vSwitches.

5. Specify whether to allow Internet access to the API server.

   > **Note** An ASM instance runs on Kubernetes runtime. You can use the API server to define various mesh resources, such as virtual services, destination rules, and Istio gateways.
   > - If you allow Internet access to the API server, an elastic IP address (EIP) is created and bound to a Server Load Balancer (SLB) instance on the private network. Port 6443 of the API server is exposed. You can use the kubeconfig file of the cluster to connect to and manage the registered cluster to define mesh resources over the Internet.
   > - If you do not allow Internet access to the API server, no EIP is created. You can use the kubeconfig file to connect to and manage the registered cluster to define mesh resource only through the VPC where the cluster resides.

6. Select **Expose Istio Pilot** in the Internet Access section.

   > **Note** If you do not select Expose Istio Pilot, the pod in the registered external cluster cannot connect to Istio Pilot, and applications in the pod cannot work as expected.

7. Keep the default settings for other parameters. Click **OK** to create the ASM instance.

   > **Note** It takes 2 to 3 minutes to create an ASM instance.

8.
9.
10. In the **Add Cluster** panel, select an external cluster as required and click **OK**.

> **Note** After you add a cluster to an ASM instance, the status of the ASM instance becomes **Updating**. Wait a few seconds and click **Refresh** in the upper-right corner. If the cluster is added to the instance, the status of the instance will become **Running**. The waiting duration may vary with the network speed. On the **Kubernetes Clusters** page, you can view the information about the added cluster.

11.

12. In the Deploy Ingress Gateway panel, set the parameters as required.
    i. Select the cluster where you want to deploy an ingress gateway service from the **Cluster** drop-down list.
    ii. Select Internet Access or Internal Access for the SLB Instance Type parameter.

    > **Note** Different external clusters may support different types of SLB instances. For example, specific external clusters do not support internal SLB instances. Select the SLB instance type as required. If the registered external cluster does not support SLB instances, select Internet Access for SLB Instance Type. After the ingress gateway service is defined, edit the YAML file of the ingress gateway service to specify the service type, such as Nodeport or ClusterIP.
    >
    > You can only create SLB instances instead of using existing ones for external clusters.

    iii. Configure port mappings.

    > **Note**
    > - We recommend that you use the same port for the container and the service in a mapping and enable the port on the Istio gateway.
    > - ASM provides four default ports that are commonly used by Istio. You can keep or delete the default ports, or add new ports as required.

13. Click **OK** to deploy the ingress gateway service.

    After you deploy the ingress gateway service, log on to the external cluster to view the details of the ingress gateway service.

Deploy applications in the external cluster

Deploy applications in the external cluster by running commands on the kubectl client or using the external cluster console. For more information, see Deploy an application in an ASM instance.

Define Istio resources

Define Istio resources in the ASM console. For more information, see 使用Istio资源实现版本流量路由.

8.3. Use ASM to deploy an application in multiple clusters in the same VPC

Alibaba Cloud Service Mesh (ASM) allows you to deploy the microservices of an application in multiple clusters in the same Virtual Private Cloud (VPC). This topic uses the Bookinfo application as an example to describe how to deploy an application in two clusters that share the same VPC and are added to the same ASM instance.

Prerequisites

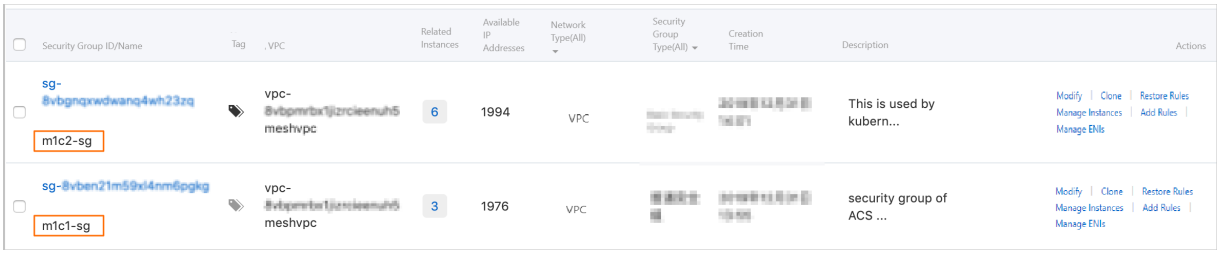
- Two Container Service for Kubernetes (ACK) clusters are created in the same VPC. For more information, see Create an ACK dedicated cluster. In this topic, the two clusters are m1c1 and m1c2.
- An ASM instance is created. For more information, see Create an ASM instance. In this topic, the ASM instance is mesh1.

Step 1: Change the security group names for the two clusters

Change the security group names for the two clusters. Make sure that users can deduce the corresponding clusters from the new security group names. In this example, change the security group names to m1c1-sg and m1c2-sg.

1.
2.
3.
4. On the **Security Groups** page, find the security group that you want to modify and click **Modify** in the **Actions** column.
5. In the Modify Security Group dialog box, modify **Security Group Name** and **Description**.
6. Click **OK**.

The following figure shows the new security group names.



Step 2: Set security group rules to allow mutual access between the two clusters

To enable the two clusters to access each other, you must set rules for accessing the security groups of the two clusters.

1. On the configuration page of the m1c1-sg group, create a rule to allow the access from m1c2-sg. For more information, see Add a security group rule.
2. On the configuration page of the m1c2-sg group, create a rule to allow the access from m1c1-sg.

Step 3: Add the two clusters to the ASM instance and deploy an ingress gateway

The two clusters can access each other. After you add the two clusters to the ASM instance, you only need to deploy an ingress gateway for one of the two clusters.

1. Add the two clusters to the ASM instance. For more information, see Add a cluster to an ASM instance.

2. Deploy an ingress gateway for the m1c1 cluster. For more information, see Deploy an ingress gateway service.

Step 4: Deploy the Bookinfo application

ASM allows you to deploy an application across clusters. You can deploy the microservices of the Bookinfo application in the two clusters.

1. Deploy the Bookinfo application excluding the v3 version of the reviews microservice in the m1c2 cluster. For more information, see Deploy an application in an ASM instance.

   > **Note** The v3 version of the reviews microservice displays ratings as red stars.

   The following code shows the content of the YAML file:

```
##################################################################################################
# Details service
##################################################################################################
apiVersion: v1
kind: Service
metadata:
  name: details
  labels:
    app: details
    service: details
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: details
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-details
  labels:
    account: details
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: details-v1
  labels:
    app: details
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: details
      version: v1
  template:
```

```
  template:
    metadata:
      labels:
        app: details
        version: v1
    spec:
      serviceAccountName: bookinfo-details
      containers:
      - name: details
        image: docker.io/istio/examples-bookinfo-details-v1:1.15.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
##################################################################################################
# Ratings service
##################################################################################################
apiVersion: v1
kind: Service
metadata:
  name: ratings
  labels:
    app: ratings
    service: ratings
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: ratings
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-ratings
  labels:
    account: ratings
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  labels:
    app: ratings
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ratings
      version: v1
  template:
    metadata:
```

```
      labels:
        app: ratings
        version: v1
    spec:
      serviceAccountName: bookinfo-ratings
      containers:
      - name: ratings
        image: docker.io/istio/examples-bookinfo-ratings-v1:1.15.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
##################################################################################################
# Reviews service
##################################################################################################
apiVersion: v1
kind: Service
metadata:
  name: reviews
  labels:
    app: reviews
    service: reviews
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: reviews
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-reviews
  labels:
    account: reviews
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reviews-v1
  labels:
    app: reviews
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reviews
      version: v1
  template:
    metadata:
      labels:
```

```
        app: reviews
        version: v1
    spec:
      serviceAccountName: bookinfo-reviews
      containers:
      - name: reviews
        image: docker.io/istio/examples-bookinfo-reviews-v1:1.15.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reviews-v2
  labels:
    app: reviews
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reviews
      version: v2
  template:
    metadata:
      labels:
        app: reviews
        version: v2
    spec:
      serviceAccountName: bookinfo-reviews
      containers:
      - name: reviews
        image: docker.io/istio/examples-bookinfo-reviews-v2:1.15.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
# ---
# apiVersion: apps/v1
# kind: Deployment
# metadata:
#   name: reviews-v3
#   labels:
#     app: reviews
#     version: v3
# spec:
#   replicas: 1
#   selector:
#     matchLabels:
#       app: reviews
#       version: v3
#   template:
#     metadata:
#       labels:
```

```
#         app: reviews
#         version: v3
#     spec:
#       serviceAccountName: bookinfo-reviews
#       containers:
#       - name: reviews
#         image: docker.io/istio/examples-bookinfo-reviews-v3:1.15.0
#         imagePullPolicy: IfNotPresent
#         ports:
#         - containerPort: 9080
---
##################################################################################################
# Productpage services
##################################################################################################
apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
    service: productpage
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: productpage
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-productpage
  labels:
    account: productpage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: productpage-v1
  labels:
    app: productpage
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: productpage
      version: v1
  template:
    metadata:
      labels:
        app: productpage
```

```
        version: v1
    spec:
      serviceAccountName: bookinfo-productpage
      containers:
      - name: productpage
        image: docker.io/istio/examples-bookinfo-productpage-v1:1.15.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
```

2. Deploy the v3 version of the reviews microservice and the rating microservice on which the reviews microservice depends in the m1c1 cluster.

   The following code shows the content of the YAML file:

```
##################################################################################################
#############
# Reviews service
##################################################################################################
#############
apiVersion: v1
kind: Service
metadata:
  name: reviews
  labels:
    app: reviews
    service: reviews
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: reviews
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-reviews
  labels:
    account: reviews
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reviews-v3
  labels:
    app: reviews
    version: v3
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reviews
      version: v3
```

```
  template:
    metadata:
      labels:
        app: reviews
        version: v3
    spec:
      serviceAccountName: bookinfo-reviews
      containers:
      - name: reviews
        image: docker.io/istio/examples-bookinfo-reviews-v3:1.15.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
##################################################################################################
#############
# Ratings service
##################################################################################################
#############
apiVersion: v1
kind: Service
metadata:
  name: ratings
  labels:
    app: ratings
    service: ratings
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: ratings
```

Step 5: Define a virtual service and an Istio gateway

1. In the namespace that is named default of the ASM instance, define a virtual service that is named bookinfo. For more information, see 使用Istio资源实现版本流量路由.

   The following code shows the content of the YAML file:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        prefix: /static
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
          number: 9080
```

2. In the namespace that is named default of the ASM instance, define an Istio gateway that is named bookinfo-gateway. For more information, see 使用Istio资源实现版本流量路由.

   The following code shows the content of the YAML file:

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
```

You can refresh the product page to view the effect of the three versions of the reviews microservice in turn. The v3 version of the reviews microservice can take effect normally though it does not reside in the same cluster as other microservices.

Step 6: Make the v3 version of the reviews microservice take effect all the time (Optional)

You can define a destination rule and a virtual service to set a policy for deploying the microservices of the Bookinfo application. The following example specifies that the v3 version of the reviews microservice always takes effect.

1. In the namespace that is named default of the ASM instance, define a destination rule that is named reviews.

   The following code shows the content of the YAML file:

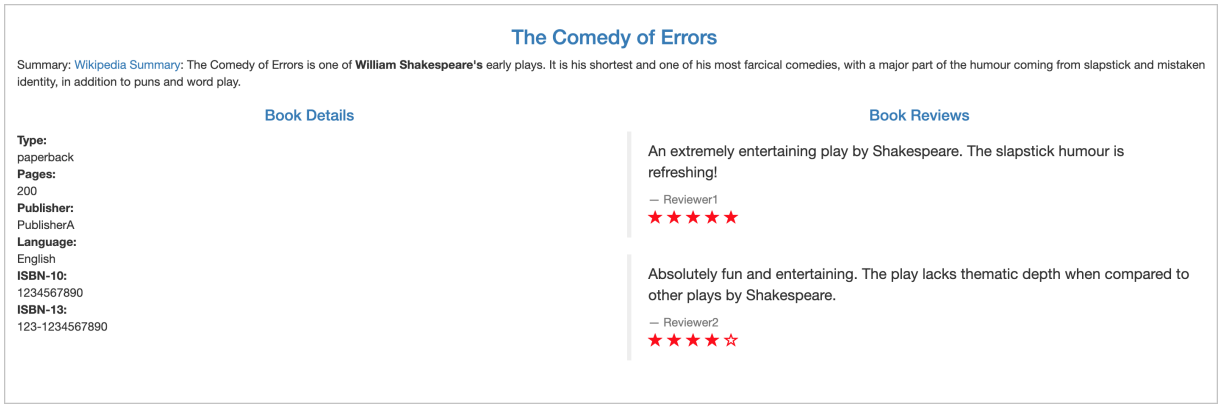
```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  - name: v3
    labels:
      version: v3
```

2. In the namespace that is named default of the ASM instance, define a virtual service that is named reviews.

   The following code shows the content of the YAML file:

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v3
```

When you access the product page, the v3 version of the reviews microservice takes effect all the time. In this case, ratings are displayed as red stars.



8.4. Use ASM to implement cross-region disaster recovery and load balancing

Alibaba Cloud Service Mesh (ASM) provides cross-region traffic distribution and failover capabilities for applications. The cross-region traffic distribution feature implements cross-region load balancing by routing traffic to multiple clusters based on their weights. The cross-region failover feature implements cross-region disaster recovery by transferring traffic from a faulty region to another region. This topic describes how to use cross-region failover and traffic distribution features to implement cross-region disaster recovery and load balancing. The Bookinfo application is used as an example.

Prerequisites

An ASM instance is created. For more information, see Create an ASM instance.

Plan a network

Before you use ASM, you must plan a network for ASM. This involves the CIDR blocks and names of the vSwitches, virtual private clouds (VPCs), and clusters. In this example, create a network based on the following plan:

- Network plan for the vSwitches and VPCs
  - vSwitches

> **Notice** To prevent route conflicts when you use Cloud Enterprise Network (CEN) to connect to a VPC, use different CIDR blocks for each vSwitch.

vSwitch	VPC	IPv4 CIDR block
vpc-hangzhou-switch-1	vpc-hangzhou	192.168.0.0/24
vpc-shanghai-switch-1	vpc-shanghai	192.168.2.0/24

- VPCs

VPC	Region	IPv4 CIDR block
vpc-hangzhou	cn-hangzhou	192.168.0.0/16
vpc-shanghai	cn-shanghai	192.168.0.0/16

- Network plan for the clusters

Cluster	Region	VPC	Pod CIDR	Service CIDR
ack-hangzhou	cn-hangzhou	vpc-hangzhou	10.45.0.0/16	172.16.0.0/16
ack-shanghai	cn-shanghai	vpc-shanghai	10.47.0.0/16	172.18.0.0/16

Step 1: Create clusters in different regions

1. Create two vSwitches in the China (Hangzhou) and China (Shanghai) regions based on the preceding plan, and then create VPCs that are associated with the vSwitches. For more information, see Create a vSwitch and Create a VPC.
2. Use the VPCs that you created and the preceding network plan to create clusters in the China (Hangzhou) and China (Shanghai) regions. For more information, see Create an ACK managed cluster.

Step 2: Use CEN to implement cross-region VPC communication

1. Create a CEN instance.
   i. Log on to the CEN console.
   ii. On the **Instances** page, click **Create CEN Instance**.
   iii. In the **Create CEN Instance** panel, set the parameters and click **OK**.

Parameter	Description
Name	The name of the CEN instance. The name must be 2 to 128 characters in length and can contain digits, underscores (_), and hyphens (-). It must start with a letter.
Description	The description of the instance.
Network Type	The type of the network. In this example, VPC is used.
Region	The region where the instance resides. In this example, China (Hangzhou) is used.
Networks	The instance that you want to attach. In this example, the VPC that you created in the China (Hangzhou) region is used.

2. Attach a network instance.

i. On the **Instances** page, find the CEN instance that you created and click its ID.

ii. Click the **Networks** tab and then click **Attach Network**.

iii. In the **Attach Network** panel, set the parameters on the **Your Account** tab and click **OK**.

Parameter	Description
Network Type	The type of the network. In this example, VPC is used.
Region	The region where the instance resides. In this example, China (Shanghai) is used.
Networks	The instance that you want to attach. In this example, the VPC that you created in the China (Shanghai) region is used.

3. Purchase the bandwidth plan. For more information, see Purchase a bandwidth plan.
4. Set the cross-region connection bandwidth.
   i. On the **Instances** page, find the CEN instance that you created and click its ID.
   ii. Click the **Region Connections** tab and then click **Set Region Connection**.
   iii. In the **Set Region Connection** panel, select the bandwidth plan that you purchased from the **Bandwidth Plans** drop-down list, set the **Connected Regions** to China (Shanghai) and China (Hangzhou), and then click **OK**.
5. Add rules to the security groups.

   Add the pod network CIDR of the ack-shanghai cluster to the security group of the ack-hangzhou cluster and vice versa. This allows the pod CIDR of a cluster to access the localhost of another cluster.

   i.
   ii. On the **Clusters** page, find the ack-shanghai cluster and click **Details** in the **Actions** column.
   iii. On the **Cluster Information** page, click the **Basic Information** tab.

   View the pod network CIDR of the ack-shanghai cluster and go back to the **Clusters** page.

   iv. On the **Clusters** page, find the ack-hangzhou cluster and click **Details** in the **Actions** column.
   v. On the **Cluster Information** page, click the **Cluster Resources** tab. Then, click the security group ID next to **Security Group**.
   vi. On the **Security Group Rules** page, click **Add Rule** on the **Inbound** tab.
   vii. Set the **Protocol Type** parameter to **All** and the **Source** parameter to the pod network CIDR of the ack-shanghai cluster. Then, click **Save** in the **Actions** column.
   viii. Repeat the preceding substeps to view the pod network CIDR of the ack-hangzhou cluster and add the pod network CIDR to the security group of the ack-shanghai cluster.

Step 3: Publish the pod route information to CEN

1.
2. On the **Clusters** page, find the ack-hangzhou cluster and click **Details** in the **Actions** column.
3. On the cluster details page, click the **Cluster Resources** tab and click the VPC ID next to **VPC**.

4. On the Information tab, view the router ID in the **vRouter Basic Information** section.
5. In the left-side navigation pane of the VPC console, click **Route Tables**.
6. On the **Route Tables** page, find the router ID and click the name of the route instance.
7. On the **Route Entry List** tab, click **Custom**.
8. Click **Publish** next to the pod CIDR block. This topic uses a CIDR block of 10.45.0.0/16.
9. In the **Publish Route Entry** dialog box, click **OK**.
10. Repeat the preceding substeps to publish the pod route information about the ack-shanghai cluster to CEN.
11. Verify whether the pod route information about the ack-hangzhou and ack-shanghai clusters is published to CEN.

    On the details page of the route table in the China (Hangzhou) region, click **Dynamic** on the **Routes** tab. You can view the route information of the pod CIDR block in the ack-shanghai cluster and the route information about the IPv4 CIDR block of vpc-shanghai-switch-1. On the details page of the route table in the China (Shanghai) region, you can also view the route information of the pod CIDR block in the ack-hangzhou cluster and the route information about the IPv4 CIDR block of vpc-hangzhou-switch-1. This indicates that the pod route information about the ack-hangzhou and ack-shanghai clusters is published to CEN.

Step 4: Add clusters to an ASM instance

Add the ack-hangzhou and ack-shanghai clusters that you created to an ASM instance.

1.
2.
3.
4.
5. In the **Add Cluster** panel, select the ack-hangzhou cluster and click **OK**.
6. In the **Note** dialog box, click **OK**.
7. Repeat the preceding substeps to add the ack-shanghai cluster to the same ASM instance.

Step 5: Configure an ingress gateway in ASM

1. View the ID of the ack-shanghai cluster.
   i.
   ii. On the **Clusters** page, find the ack-shanghai cluster and click **Details** in the **Actions** column.
   iii. On the Cluster Information page, click the **Basic Information** tab.

   In the **Basic Information** section, view the ID of the ack-shanghai cluster.
2.
3.
4.
5.
6. On the **ASM Gateways** page, click **Create**.
7. On the **Create** page, select the ack-hangzhou cluster from the **Cluster** drop-down list, set the **SLB Instance Type** parameter to **Internet Access**, and then select an SLB instance from the

Create SLB Instance drop-down list. Keep the default values of other parameters. Click **Create**.

8. On the **ASM Gateways** page, find the gateway named ingressgateway and click **YAML** in the **Actions** column.
9. In the **Edit** panel, enter the ID of the ack-shanghai cluster and click **OK**.

```
spec:
  clusterIds:
    - ack-hangzhou cluster-id
    - ack-shanghai cluster-id
```

Step 6: Deploy the Bookinfo application

1. Use kubectl to connect to the ack-hangzhou cluster. For more information, see Connect to ACK clusters by using kubectl.
2. Create an *ack-hangzhou-k8s.yaml* file that contains the following content:

   View the content of the YAML file.

```
# Details service
apiVersion: v1
kind: Service
metadata:
  name: details
  labels:
    app: details
    service: details
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: details
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-details
  labels:
    account: details
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: details-v1
  labels:
    app: details
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: details
      version: v1
  template:
```

```
    metadata:
      labels:
        app: details
        version: v1
    spec:
      serviceAccountName: bookinfo-details
      containers:
      - name: details
        image: docker.io/istio/examples-bookinfo-details-v1:1.16.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
        securityContext:
          runAsUser: 1000
---
# Ratings service
apiVersion: v1
kind: Service
metadata:
  name: ratings
  labels:
    app: ratings
    service: ratings
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: ratings
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-ratings
  labels:
    account: ratings
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  labels:
    app: ratings
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ratings
      version: v1
  template:
    metadata:
      labels:
        app: ratings
```

```
        version: v1
    spec:
      serviceAccountName: bookinfo-ratings
      containers:
      - name: ratings
        image: docker.io/istio/examples-bookinfo-ratings-v1:1.16.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
        securityContext:
          runAsUser: 1000
---
# Reviews service
apiVersion: v1
kind: Service
metadata:
  name: reviews
  labels:
    app: reviews
    service: reviews
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: reviews
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-reviews
  labels:
    account: reviews
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reviews-v1
  labels:
    app: reviews
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reviews
      version: v1
  template:
    metadata:
      labels:
        app: reviews
        version: v1
    spec:
      serviceAccountName: bookinfo-reviews
```

```
      containers:
      - name: reviews
        image: docker.io/istio/examples-bookinfo-reviews-v1:1.16.2
        imagePullPolicy: IfNotPresent
        env:
        - name: LOG_DIR
          value: "/tmp/logs"
        ports:
        - containerPort: 9080
        volumeMounts:
        - name: tmp
          mountPath: /tmp
        - name: wlp-output
          mountPath: /opt/ibm/wlp/output
        securityContext:
          runAsUser: 1000
      volumes:
      - name: wlp-output
        emptyDir: {}
      - name: tmp
        emptyDir: {}
---
# Productpage services
apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
    service: productpage
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: productpage
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-productpage
  labels:
    account: productpage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: productpage-v1
  labels:
    app: productpage
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
    matchLabels:
      app: productpage
      version: v1
  template:
    metadata:
      labels:
        app: productpage
        version: v1
    spec:
      serviceAccountName: bookinfo-productpage
      containers:
      - name: productpage
        image: docker.io/istio/examples-bookinfo-productpage-v1:1.16.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
        volumeMounts:
        - name: tmp
          mountPath: /tmp
        securityContext:
          runAsUser: 1000
      volumes:
      - name: tmp
        emptyDir: {}
---
```

3. Run the following command to deploy the BookInfo application in the ack-hangzhou cluster:

```
kubectl apply -f ack-hangzhou-k8s.yaml
```

4. Use kubectl to connect to the ack-shanghai cluster. For more information, see Connect to ACK clusters by using kubectl.

> **Note** When you use kubectl to connect to the ack-shanghai cluster, you must switch the kubeconfig of the ack-hangzhou cluster to that of the ack-shanghai cluster.

5. Create an *ack-shanghai.yaml* file that contains the following content:

   View the content of the YAML file.

```
# Details service
apiVersion: v1
kind: Service
metadata:
  name: details
  labels:
    app: details
    service: details
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: details
```

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-details
  labels:
    account: details
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: details-v1
  labels:
    app: details
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: details
      version: v1
  template:
    metadata:
      labels:
        app: details
        version: v1
    spec:
      serviceAccountName: bookinfo-details
      containers:
      - name: details
        image: docker.io/istio/examples-bookinfo-details-v1:1.16.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
        securityContext:
          runAsUser: 1000
---
# Ratings service
apiVersion: v1
kind: Service
metadata:
  name: ratings
  labels:
    app: ratings
    service: ratings
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: ratings
---
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: bookinfo-ratings
  labels:
    account: ratings
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  labels:
    app: ratings
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ratings
      version: v1
  template:
    metadata:
      labels:
        app: ratings
        version: v1
    spec:
      serviceAccountName: bookinfo-ratings
      containers:
      - name: ratings
        image: docker.io/istio/examples-bookinfo-ratings-v1:1.16.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
        securityContext:
          runAsUser: 1000
---
# Reviews service
apiVersion: v1
kind: Service
metadata:
  name: reviews
  labels:
    app: reviews
    service: reviews
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: reviews
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-reviews
  labels:
    account: reviews
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reviews-v2
  labels:
    app: reviews
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reviews
      version: v2
  template:
    metadata:
      labels:
        app: reviews
        version: v2
    spec:
      serviceAccountName: bookinfo-reviews
      containers:
      - name: reviews
        image: docker.io/istio/examples-bookinfo-reviews-v2:1.16.2
        imagePullPolicy: IfNotPresent
        env:
        - name: LOG_DIR
          value: "/tmp/logs"
        ports:
        - containerPort: 9080
        volumeMounts:
        - name: tmp
          mountPath: /tmp
        - name: wlp-output
          mountPath: /opt/ibm/wlp/output
        securityContext:
          runAsUser: 1000
      volumes:
      - name: wlp-output
        emptyDir: {}
      - name: tmp
        emptyDir: {}
---
# Productpage services
apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
    service: productpage
spec:
  ports:
  - port: 9080
```

```
    name: http
  selector:
    app: productpage
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-productpage
  labels:
    account: productpage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: productpage-v1
  labels:
    app: productpage
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: productpage
      version: v1
  template:
    metadata:
      labels:
        app: productpage
        version: v1
    spec:
      serviceAccountName: bookinfo-productpage
      containers:
      - name: productpage
        image: docker.io/istio/examples-bookinfo-productpage-v1:1.16.2
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
        volumeMounts:
        - name: tmp
          mountPath: /tmp
        securityContext:
          runAsUser: 1000
      volumes:
      - name: tmp
        emptyDir: {}
---
```

6. Run the following command to deploy the BookInfo application in the ack-shanghai cluster:

```
kubectl apply -f ack-shanghai.yaml
```

7. Use kubectl to connect to the ASM instance. For more information, see Use kubectl to connect to an ASM instance.

> **Note** When you use kubectl to connect to the ASM instance, you must switch the kubeconfig of the ack-shanghai cluster to that of the ASM instance.

8. Create an *asm.yaml* file that contains the following content:

   View the content of the YAML file.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        prefix: /static
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
          number: 9080
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: productpage
spec:
```

```
  host: productpage
  subsets:
  - name: v1
    labels:
      version: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  - name: v3
    labels:
      version: v3
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ratings
spec:
  host: ratings
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  - name: v2-mysql
    labels:
      version: v2-mysql
  - name: v2-mysql-vm
    labels:
      version: v2-mysql-vm
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: details
spec:
  host: details
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
```

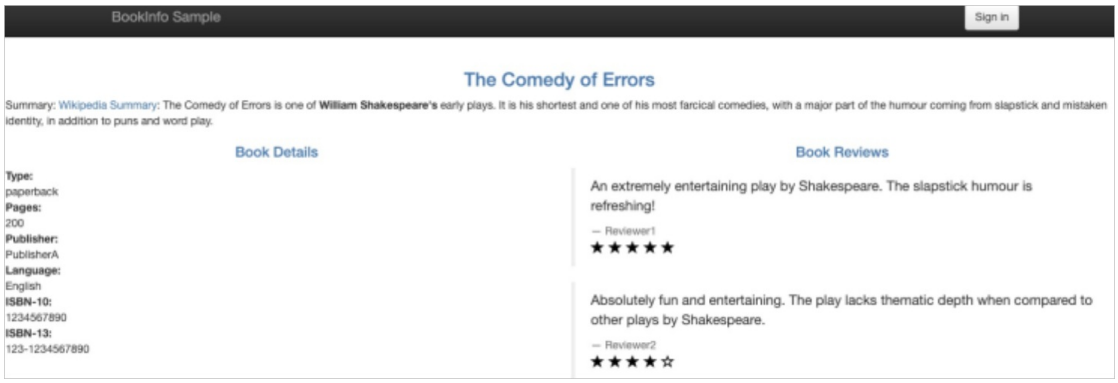
```
    labels:
      version: v2
---
```

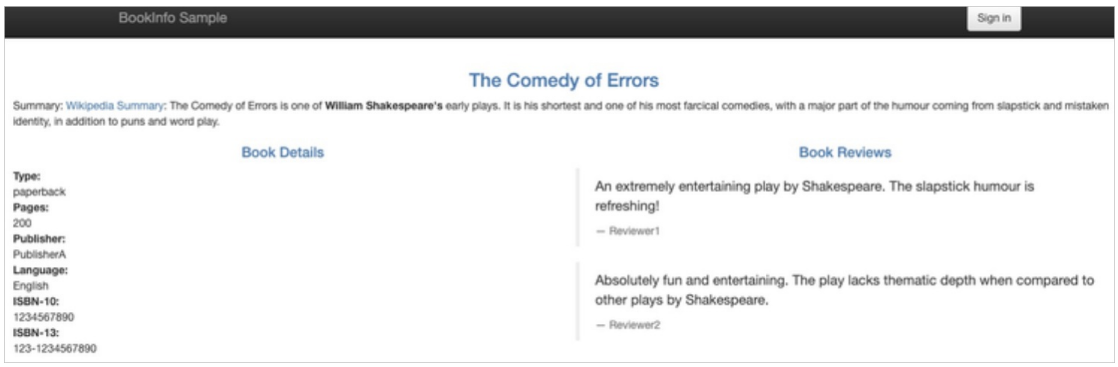
9. Run the following command to create a routing rule in the ASM instance:

```
kubectl apply -f asm.yaml
```

10. Verify whether the Bookinfo application is deployed.
    i.
    ii.
    iii. On the **Clusters** page, find the ack-hangzhou cluster and click **Details** in the **Actions** column.
    iv.
    v. At the top of the **Services** page, select istio-system from the **Namespace** drop-down list. Find the ingress gateway that is named istio-ingressgateway and view the IP address whose port is 80 in the **External Endpoint** column.
    vi. Enter the *<IP address of the ingress gateway>/productpage* in the address bar of your browser.

    Refresh the page multiple times. The following images alternately appear on the screen.





Step 7: Use the cross-region traffic distribution and failover features

Configure cross-region traffic distribution

1.
2.
3.

4. In the **Basic Information** section, click **Enable locality traffic distribution** on the right of **Locality-Failover**.

   > **Note** If you have enabled cross-region failover, you must disable cross-region failover before you can enable cross-region traffic distribution.

5. In the **Locality-Traffic-Distribution** dialog box, set the **Policy** parameter to cn-hangzhou and click **New Policy**.

6. Click the icon and the icon. Set the **To** parameter to cn-hangzhou and the **Weight** parameter to 90%.

7. Click the icon, set the **To** parameter to cn-shanghai and the **Weight** parameter to 10%, and then click **Submit**.

8. Run the following command to request the BookInfo application 10 times to verify whether the cross-region traffic distribution is successful:

   ```
   for ((i=1;i<=10;i++));do curl http://<Ingress gateway endpoint of port 80 in the ack-hangzhou cluster>/productpage 2>&1|grep full.stars;done
   ```

   Expected output:

   ```
   <!-- full stars: -->
   <!-- full stars: -->
   ```

   You can find that 10 access requests are made and two rows of `full stars` output are returned. This indicates that 9 of the 10 requests are routed to the v1 reviews service in the ack-hangzhou cluster and 1 request is routed to the v2 reviews service in the ack-shanghai cluster. Traffic is routed to different clusters based on the weights of the clusters.

Configure cross-region failover

1. Disables the reviews service in the ack-hangzhou cluster.
   i.
   ii.
   iii.
   iv. On the **Deployments** page, set the **Namespace** parameter to default, find reviews-v1, and then click **Scale** in the **Actions** column.
   v. In the **Scale** dialog box, set the **Desired Number of Pods** parameter to 0 and click **OK**.

2. Configure a destination rule.

   Configure a destination rule. If the reviews service cannot be requested within 1 second, the reviews service will be ejected for 1 minute.
   i.
   ii.
   iii.
   iv.
   v. On the **DestinationRule** page, find the reviews service and click **YAML** in the **Actions** column.

vi. In the **Edit** panel, copy the following content to the code editor and click **OK**.

```
spec:
  ......
  trafficPolicy:
    connectionPool:
      http:
        maxRequestsPerConnection: 1
    outlierDetection:
      baseEjectionTime: 1m
      consecutive5xxErrors: 1
      interval: 1s
```

- maxRequestsPerConnection: specifies the maximum number of requests per connection.
- baseEjectionTime: specifies the minimum ejection duration.
- consecutive5xxErrors: specifies the number of consecutive errors.
- interval: specifies the time interval for ejection analysis.

3. Enable cross-region failover.

i. In the **Basic Information** section, click **Enable Locality-Failover** on the right of **Locality-Failover**.

> **Note** If you have enabled cross-region traffic distribution, you must disable cross-region traffic distribution before you can enable cross-region failover.

ii. In the **Locality-Failover** dialog box, set the Failover parameter to cn-hangzhou if the **From** parameter is set to cn-shanghai. Set the Failover parameter to cn-shanghai if the **From** parameter is set to cn-hangzhou. Then, click **Submit**.

4. Run the following command to request the BookInfo application 10 times and record the number of successful routes to the v2 reviews service:

```
for ((i=1;i<=10;i++));do curl http://<Ingress gateway endpoint of port 80 in the ack-hangzhou cluster>/productpage 2>&1|grep full.stars;done|wc -l
```

Expected output:

```
20
```

You can find that 10 access requests are made and 20 rows of results are returned. This is because a two-row result that contains `full stars` is returned each time a route to the v2 reviews service succeeds. This indicates that all 10 requests are routed to the v2 reviews service in the ack-shanghai cluster, and the cross-region failover is successful.

8.5. Implement auto scaling for workloads by using ASM metrics

Alibaba Cloud Service Mesh (ASM) collects telemetry data for Container Service for Kubernetes (ACK) clusters in a non-intrusive manner, which makes the service communication in the clusters observable. This telemetry feature makes service behaviors observable and helps O&M staff troubleshoot, maintain, and optimize applications without increasing maintenance costs. Based on the four key monitoring metrics, including latency, traffic, errors, and saturation, ASM generates a series of metrics for the services that it manages. This topic shows you how to implement auto scaling for workloads by using ASM metrics.

Prerequisites

- An ACK cluster is created. For more information, see Create an ACK managed cluster.
- An ASM instance is created. For more information, see Create an ASM instance.
- A Prometheus instance and a Grafana instance are deployed in the ACK cluster. For more information, see Use Prometheus to monitor an ACK cluster.
- A Prometheus instance is deployed to monitor the ASM instance. For more information, see Deploy a self-managed Prometheus instance to monitor ASM instances.

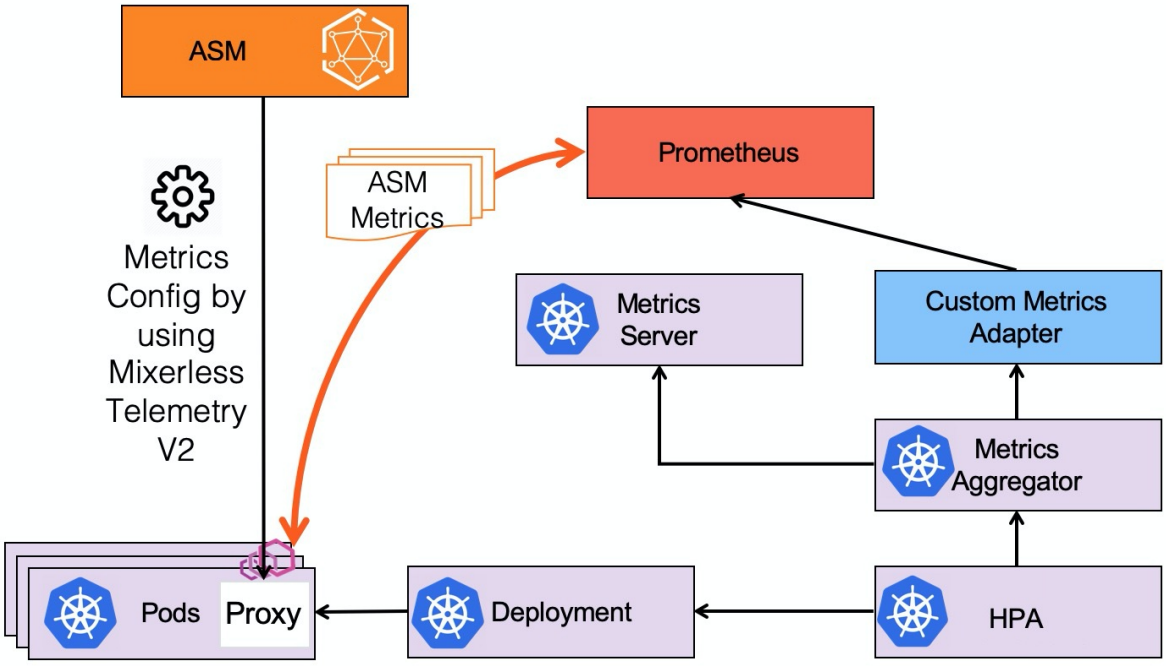
Context

ASM generates a series of metrics for the services that it manages. For more information, visit Istio Standard Metrics.

Auto scaling is an approach that is used to automatically scale up or down workloads based on the resource usage. In Kubernetes, two autoscalers are used to implement auto scaling.

- Cluster Autoscaler (CA): CAs are used to increase or decrease the number of nodes in a cluster.
- Horizontal Pod Autoscaler (HPA): HPAs are used to increase or decrease the number of pods that are used to deploy applications.

The aggregation layer of Kubernetes allows third-party applications to extend the Kubernetes API by registering themselves as API add-ons. These add-ons can be used to implement the custom metrics API and allow HPAs to query any metrics. HPAs periodically query core metrics such as CPU utilization and memory usage by using the resource metrics API. In addition, HPAs use the custom metrics API to query specific application metrics, such as the observability metrics that are provided by ASM.



Step 1: Enable Prometheus monitoring for the ASM instance

1.
2.
3.
4. On the management page of the ASM instance, click **Settings** in the upper-right corner.

   > **Note** Make sure that the Istio version of the ASM instance is 1.6.8.4 or later.

5. In the **Settings Update** panel, select **Enable Prometheus**. Then, click **OK**.

   After that, ASM automatically configures the Envoy filters that are required for Prometheus.

Step 2: Deploy the adapter for the custom metrics API

1. Download the installation package of the adapter. For more information, visit kube-metrics-adapter. Then, install and deploy the adapter for the custom metrics API in the ACK cluster.

   ```
   ## Use Helm 3.
   helm -n kube-system install asm-custom-metrics ./kube-metrics-adapter  --set prometheus
   .url=http://prometheus.istio-system.svc:9090
   ```

2. After the installation is completed, run the following commands to check whether kube-metrics-adapter is enabled.
   - Check whether the autoscaling/v2beta API group exists.

```
kubectl api-versions |grep "autoscaling/v2beta"
```

The following output is expected:

```
autoscaling/v2beta
```

- Check the status of the pod of kube-metrics-adapter.

```
kubectl get po -n kube-system |grep metrics-adapter
```

The following output is expected:

```
asm-custom-metrics-kube-metrics-adapter-85c6d5d865-2cm57          1/1     Running   0
19s
```

- Query the custom metrics that are provided by kube-metrics-adapter.

```
kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1" | jq .
```

The following output is expected:

```
{
  "kind": "APIResourceList",
  "apiVersion": "v1",
  "groupVersion": "external.metrics.k8s.io/v1beta1",
  "resources": []
}
```

Step 3: Deploy a sample application

1. Create a namespace named test. For more information, see Manage namespaces.
2. Enable automatic sidecar injection. For more information, see Install a sidecar proxy.
3. Deploy a sample application.
   i. Create a file named *podinfo.yaml*.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: podinfo
  namespace: test
  labels:
    app: podinfo
spec:
  minReadySeconds: 5
  strategy:
    rollingUpdate:
      maxUnavailable: 0
    type: RollingUpdate
  selector:
    matchLabels:
      app: podinfo
  template:
    metadata:
      annotations:
```

```
        prometheus.io/scrape: "true"
      labels:
        app: podinfo
    spec:
      containers:
      - name: podinfod
        image: stefanprodan/podinfo:latest
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9898
          name: http
          protocol: TCP
        command:
        - ./podinfo
        - --port=9898
        - --level=info
        livenessProbe:
          exec:
            command:
            - podcli
            - check
            - http
            - localhost:9898/healthz
          initialDelaySeconds: 5
          timeoutSeconds: 5
        readinessProbe:
          exec:
            command:
            - podcli
            - check
            - http
            - localhost:9898/readyz
          initialDelaySeconds: 5
          timeoutSeconds: 5
        resources:
          limits:
            cpu: 2000m
            memory: 512Mi
          requests:
            cpu: 100m
            memory: 64Mi
---
apiVersion: v1
kind: Service
metadata:
  name: podinfo
  namespace: test
  labels:
    app: podinfo
spec:
  type: ClusterIP
  ports:
    - name: http
      port: 9898
      targetPort: 9898
```

```
      targetPort: 9898
      protocol: TCP
  selector:
    app: podinfo
```

ii. Deploy the podinfo application.

```
kubectl apply -n test -f podinfo.yaml
```

4. To trigger auto scaling, you must deploy a load testing service in the test namespace for triggering requests.

i. Create a file named *loadtester.yaml*.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: loadtester
  namespace: test
  labels:
    app: loadtester
spec:
  selector:
    matchLabels:
      app: loadtester
  template:
    metadata:
      labels:
        app: loadtester
      annotations:
        prometheus.io/scrape: "true"
    spec:
      containers:
        - name: loadtester
          image: weaveworks/flagger-loadtester:0.18.0
          imagePullPolicy: IfNotPresent
          ports:
            - name: http
              containerPort: 8080
          command:
            - ./loadtester
            - -port=8080
            - -log-level=info
            - -timeout=1h
          livenessProbe:
            exec:
              command:
                - wget
                - --quiet
                - --tries=1
                - --timeout=4
                - --spider
                - http://localhost:8080/healthz
            timeoutSeconds: 5
          readinessProbe:
            exec:
```

```
              command:
                - wget
                - --quiet
                - --tries=1
                - --timeout=4
                - --spider
                - http://localhost:8080/healthz
            timeoutSeconds: 5
          resources:
            limits:
              memory: "512Mi"
              cpu: "1000m"
            requests:
              memory: "32Mi"
              cpu: "10m"
          securityContext:
            readOnlyRootFilesystem: true
            runAsUser: 10001
---
apiVersion: v1
kind: Service
metadata:
  name: loadtester
  namespace: test
  labels:
    app: loadtester
spec:
  type: ClusterIP
  selector:
    app: loadtester
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
```

ii. Deploy the load testing service.

```
kubectl apply -n test -f loadtester.yaml
```

5. Check whether the sample application and the load testing service are deployed.

i. Check the pod status.

```
kubectl get pod -n test
```

The following output is expected:

```
NAME                          READY   STATUS    RESTARTS   AGE
loadtester-64df4846b9-nxhvv   2/2     Running   0          2m8s
podinfo-6d845cc8fc-26xbq      2/2     Running   0          11m
```

ii. Log on to the container for load testing and run the hey command to generate loads.

```
export loadtester=$(kubectl -n test get pod -l "app=loadtester" -o jsonpath='{.item
s[0].metadata.name}')
kubectl -n test exec -it ${loadtester} -- hey -z 5s -c 10 -q 2 http://podinfo.test:
9898
```

A load is generated, which indicates that the sample application and the load testing service are deployed.

Step 4: Configure an HPA by using ASM metrics

Define an HPA to scale the workloads of the Podinfo application based on the number of requests that the Podinfo application receives per second. When more than 10 requests are received per second on average, the HPA increases the number of replicas.

1. Create a file named *hpa.yaml*.

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: podinfo
  namespace: test
  annotations:
    metric-config.external.prometheus-query.prometheus/processed-requests-per-second: |
      sum(
          rate(
              istio_requests_total{
                destination_workload="podinfo",
                destination_workload_namespace="test",
                reporter="destination"
              }[1m]
          )
      )
spec:
  maxReplicas: 10
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: podinfo
  metrics:
    - type: External
      external:
        metric:
          name: prometheus-query
          selector:
            matchLabels:
              query-name: processed-requests-per-second
        target:
          type: AverageValue
          averageValue: "10"
```

2. Deploy the HPA.

```
kubectl apply -f hpa.yaml
```

3. Check whether the HPA is deployed.

   Query the custom metrics that are provided by kube-metrics-adapter.

   ```
   kubectl get --raw "/apis/external.metrics.k8s.io/v1beta1" | jq .
   ```

   The following output is expected:

   ```
   {
     "kind": "APIResourceList",
     "apiVersion": "v1",
     "groupVersion": "external.metrics.k8s.io/v1beta1",
     "resources": [
       {
         "name": "prometheus-query",
         "singularName": "",
         "namespaced": true,
         "kind": "ExternalMetricValueList",
         "verbs": [
           "get"
         ]
       }
     ]
   }
   ```

   The output contains the resource list of custom ASM metrics, which indicates that the HPA is deployed.

Verify auto scaling

1. Log on to the container for load testing and run the hey command to generate loads.

   ```
   kubectl -n test exec -it ${loadtester} -- sh
   ~ $ hey -z 5m -c 10 -q 5 http://podinfo.test:9898
   ```

2. View the effect of auto scaling.

   > **Note** Metrics are synchronized every 30 seconds by default. The container can be scaled only once in every 3 to 5 minutes. This way, the HPA can reserve time for automatic scaling before the conflict strategy is executed.

   ```
   watch kubectl -n test get hpa/podinfo
   ```

   The following output is expected:

   ```
   NAME      REFERENCE             TARGETS           MINPODS   MAXPODS   REPLICAS   AGE
   podinfo   Deployment/podinfo   8308m/10 (avg)   1         10        6          124m
   ```

   The HPA starts to scale up workloads in 1 minute until the number of requests per second decreases under the specified threshold. After the load testing is completed, the number of requests per second decreases to zero. Then, the HPA starts to decrease the number of pods. A few minutes later, the number of replicas decreases from the value in the preceding output to one.

8.6. Use an ingress gateway to access a gRPC service in an ASM instance

You can route traffic to gRPC services in an Alibaba Cloud Service Mesh (ASM) instance by using an ingress gateway. This topic describes how to use an ingress gateway to access a gRPC service in an ASM instance. This topic also describes how to switch traffic between two versions of a gRPC service.

Prerequisites

Step 1: Deploy the two versions of a gRPC service

Deploy version 1 and version 2 of a gRPC service: istio-grpc-server-v1 and istio-grpc-server-v2.

1.
2.
3.
4.
5. At the top of the **Deployments** page, select a namespace from the **Namespace** drop-down list. Click **Create from YAML**.

   > **Note** Select a namespace that has the istio-system=enabled tag, which indicates that automatic sidecar injection is enabled. For more information, see Upgrade sidecar proxies.

6. Copy the following YAML code to the Template editor. Then, click **Create**.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: istio-grpc-server-v1
  labels:
    app: istio-grpc-server
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: istio-grpc-server
      version: v1
  template:
    metadata:
      labels:
        app: istio-grpc-server
        version: v1
    spec:
      containers:
      - args:
        - --address=0.0.0.0:8080
        image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-server
        imagePullPolicy: Always
        livenessProbe:
          exec:
```

```
          exec:
            command:
            - /bin/grpc_health_probe
            - -addr=:8080
          initialDelaySeconds: 2
        name: istio-grpc-server
        ports:
        - containerPort: 8080
        readinessProbe:
          exec:
            command:
            - /bin/grpc_health_probe
            - -addr=:8080
          initialDelaySeconds: 2
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: istio-grpc-server-v2
  labels:
    app: istio-grpc-server
    version: v2
spec:
  replicas: 1
  selector:
    matchLabels:
      app: istio-grpc-server
      version: v2
  template:
    metadata:
      labels:
        app: istio-grpc-server
        version: v2
    spec:
      containers:
        - args:
            - --address=0.0.0.0:8080
          image: registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-server
          imagePullPolicy: Always
          livenessProbe:
            exec:
              command:
                - /bin/grpc_health_probe
                - -addr=:8080
            initialDelaySeconds: 2
          name: istio-grpc-server
          ports:
            - containerPort: 8080
          readinessProbe:
            exec:
              command:
                - /bin/grpc_health_probe
                - -addr=:8080
            initialDelaySeconds: 2
---
```

```
apiVersion: v1
kind: Service
metadata:
  name: istio-grpc-server
  labels:
    app: istio-grpc-server
spec:
  ports:
  - name: grpc-backend
    port: 8080
    protocol: TCP
  selector:
    app: istio-grpc-server
  type: ClusterIP
---
```

Step 2: Set routing rules for the ASM instance

Create an Istio gateway, a virtual service, and a destination rule for the ASM instance to route all inbound traffic to istio-grpc-server-v1.

1.
2.
3.
4. Create an .
   i.
   ii. In the **Create** panel, select **default** from the **Namespace** drop-down list and copy the following YAML code to the code editor. Then, click **Create**.

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: grpc-gateway
spec:
  selector:
    istio: ingressgateway # use Istio default gateway implementation
  servers:
  - port:
      number: 8080
      name: grpc
      protocol: GRPC
    hosts:
    - "*"
```

5. Create a .
   i.

ii. In the **Create** panel, select **default** from the **Namespace** drop-down list and copy the following YAML code to the code editor. Then, click **Create**.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: dr-istio-grpc-server
spec:
  host: istio-grpc-server
  trafficPolicy:
    loadBalancer:
      simple: ROUND_ROBIN
  subsets:
    - name: v1
      labels:
        version: "v1"
    - name: v2
      labels:
        version: "v2"
```

6. Create a .

i.

ii. In the **Create** panel, select **default** from the **Namespace** drop-down list and copy the following YAML code to the code editor. Then, click **Create**.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: grpc-vs
spec:
  hosts:
  - "*"
  gateways:
  - grpc-gateway
  http:
    - match:
        - port: 8080
      route:
        - destination:
            host: istio-grpc-server
            subset: v1
          weight: 100
        - destination:
            host: istio-grpc-server
            subset: v2
          weight: 0
```

Step 3: Deploy an ingress gateway service

Enable port 8080 in the ingress gateway service and point the port to port 8080 of the Istio gateway.

1.

2.

3.
4.
5. On the **ASM Gateways** page, click **Create**. In the **Create** panel, set the parameters as required.

Parameter	Description
Cluster	Select the cluster in which you want to deploy an ingress gateway service.
SLB Instance Type	Select **Internet Access**.
Use Existing SLB Instance or Create SLB Instance	Configure a Server Load Balancer (SLB) instance. ○ Use Existing SLB Instance: Select an SLB instance from the drop-down list. ○ Create SLB Instance: Click **Create SLB Instance** and select an SLB instance type from the drop-down list. **Note** We recommend that you assign a dedicated SLB instance to each Kubernetes service in the cluster. If multiple Kubernetes services share the same SLB instance, the following risks and limits exist: ○ If you assign a Kubernetes service with an SLB instance that is used by another Kubernetes service, the existing listeners of the SLB instance are forcibly overwritten. This may interrupt the original Kubernetes service and make your application unavailable. ○ If you create an SLB instance when you create a Kubernetes service, the SLB instance cannot be shared among Kubernetes services. Only SLB instances that you create in the SLB console or by calling API operations can be shared. ○ Kubernetes services that share the same SLB instance must use different frontend listening ports. Otherwise, port conflicts may occur. ○ If multiple Kubernetes services share the same SLB instance, you must use the listener names and the VServer group names as unique identifiers in Kubernetes. Do not modify the names of listeners or VServer groups. ○ You cannot share SLB instances across clusters.

Parameter	Description
Port Mapping	Click **Add Port** and set the **Protocol** parameter to TCP, the **Service Port** parameter to 8080, and the **Container Port** parameter to 8080. **Note** The service port exposes the ASM instance to external access. Inbound traffic accesses the ASM instance by using the service port and is routed to a port on the Istio gateway, which serves as the container port. Make sure that the container port that you enter is the same as the specified port on the Istio gateway.

6. Click **Create**.

Step 4: Start the gRPC client

1. Run the following command to start the gRPC client:

```
docker run -d --name grpc-client registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-grpc-client 365d
```

2. Run the following command to log on to the default container of the pod where the gRPC client resides:

```
docker exec -it grpc-client sh
```

3. Run the following command to access the gRPC service that you deployed in the ASM instance:

```
/bin/greeter-client --insecure=true --address=<IP address of the ingress gateway service>:8080 --repeat=100
```

The command output indicates that all requests are routed to istio-grpc-server-v1.

```
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:18:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
```

Step 5: Transfer a specific ratio of the traffic to istio-grpc-server-v2

Route 40% of the traffic to istio-grpc-server-v2 and 60% of the traffic to istio-grpc-server-v1.

1.

2.
3.
4.
5. On the page, click **YAML** in the **Actions** column of the grpc-vs virtual service.
6. In the **Edit** panel, copy the following YAML code to the code editor. Then, click **OK**.

```
....
      route:
        - destination:
            host: istio-grpc-server
            subset: v1
          weight: 60
        - destination:
            host: istio-grpc-server
            subset: v2
          weight: 40
```

7. Log on to the default container of the pod where the gRPC client resides. Run the following command to access the gRPC service that you deployed in the ASM instance:

```
/bin/greeter-client --insecure=true --address=<IP address of the ingress gateway servic
e>:8080 --repeat=100
```

The command output indicates that 40% of the traffic is routed to istio-grpc-server-v2.

> **Note** If you send 100 requests in a test, the traffic may not always be routed to istio-grpc-server-v1 and istio-grpc-server-v2 at an exact ratio of 60 to 40, but will be close.

```
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v1-dbdd97cc-n85lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
2020/09/11 03:34:51 Hello world from istio-grpc-server-v2-665c4cf57d-h74lw
```

9.DevOps

9.1. Set up Jenkins to build an application delivery pipeline

This topic describes how to set up a Jenkins continuous integration environment in a Container Service for Kubernetes (ACK) cluster. This topic also provides an example on how to build an application delivery pipeline that covers source code compilation, image building and pushing, and application deployment.

Prerequisites

- An ACK cluster is created. For more information, see Create an ACK managed cluster.
- a kubectl client is connected to the cluster. For more information, see Connect to ACK clusters by using kubectl.

Deploy Jenkins

We recommend that you perform the following steps to install ack-jenkins in the cluster and run a build job named **demo-pipeline**. After the build job is completed, you can customize the configuration of the build job based on your requirements.

1.
2.
3. On the **Marketplace** page, click the **App Catalog** tab. Then, find and click **ack-jenkins**.
4. On the **ack-jenkins** page, click **Deploy**.
5. In the **Deploy** wizard, select a cluster and a namespace, and then click **Next**.

   > **Note** We recommend that you select a custom namespace or the **default** namespace. In this example, a custom namespace named **ci** is selected.

6. On the **Parameters** wizard page, modify the value of the `AdminPassword` field and click **OK**.

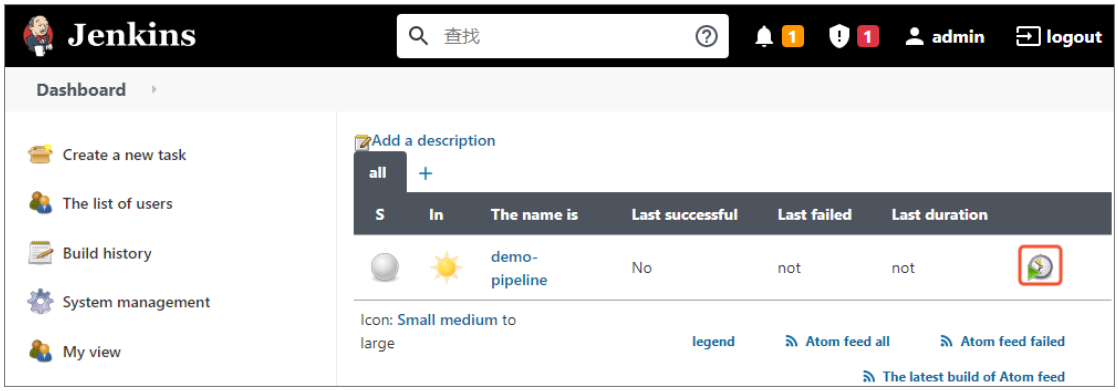
   If you do not specify a password in the `AdminPassword` field, the system automatically generates a password after Jenkins is deployed. The default password is `admin`.

7. Log on to Jenkins.
   i.
   ii.
   iii.
   iv. Select the namespace in which Jenkins is deployed. Find the **ack-jenkins-default** Service and then click the endpoint in the **External Endpoint** column to log on to Jenkins.
8. Configure the settings on the **Getting Started** page.
   i. On the **Getting Started** page, click **Install suggested plugins**.
   ii. After the plug-ins are installed, click **Save and Finish** in the **Instance Configuration** section of the **Getting Started** page.

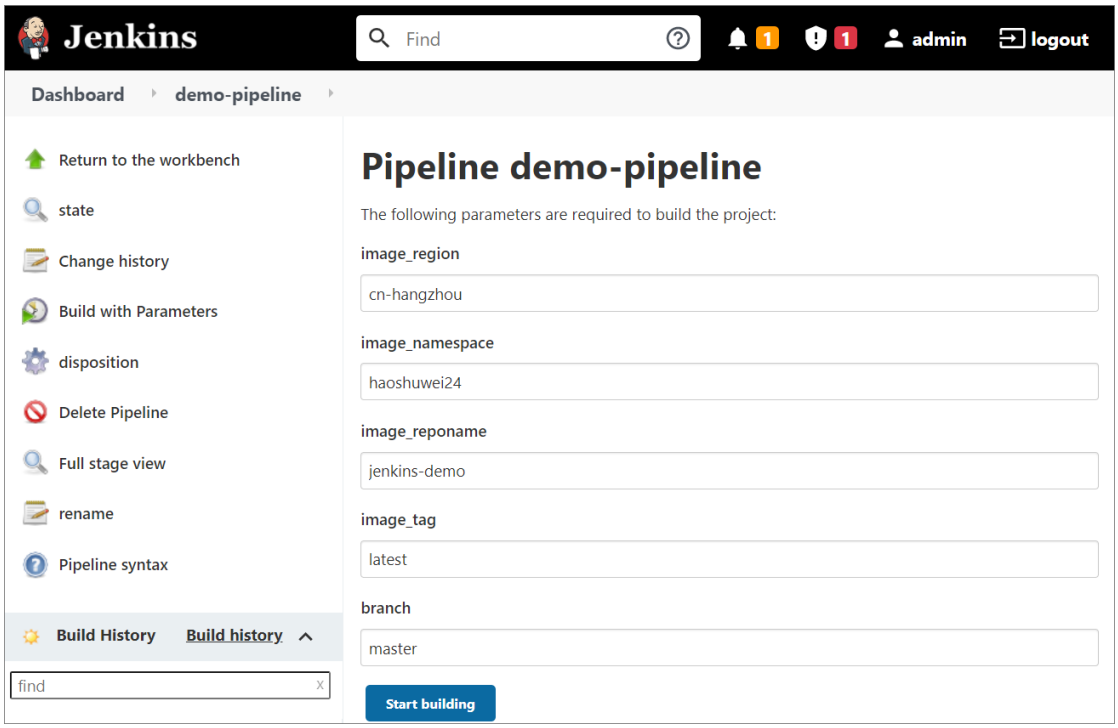
iii. Click **Start using Jenkins**.

9. Build a pipeline named **demo-pipeline** and access the pipeline.

i. On the Jenkins homepage, click the icon to the right side of **demo-pipeline**.



ii. Modify the build parameters based on information about your image repository. In this example, the branch of the source code repository is `master` and the image is `registry.cn-beijing.aliyuncs.com/ack-cicd/ack-jenkins-demo:latest`.

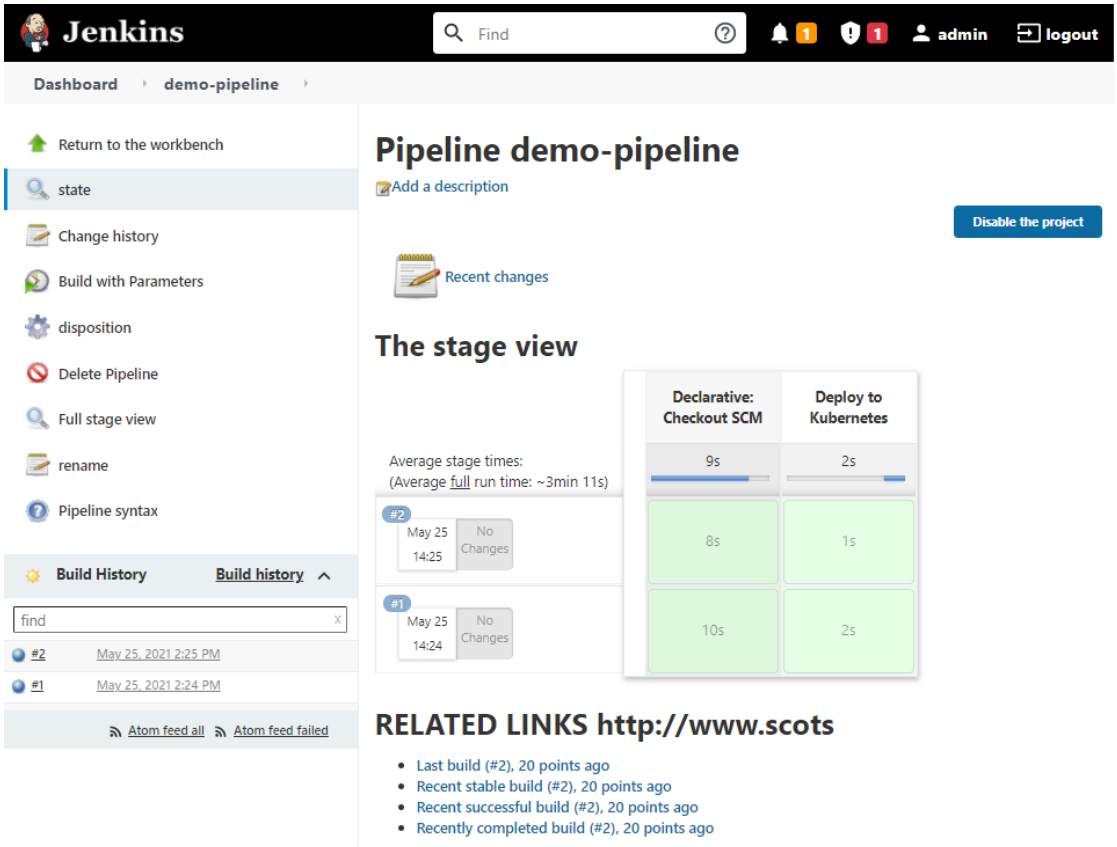


iii. Click **Start building**.

Test the connectivity between the Jenkins master and the Jenkins slave pod that is dynamically allocated by the Kubernetes cluster.

After you click Build, Jenkins dynamically creates a slave pod in the Kubernetes cluster to run the build job. For more information about the sample application code, see jenkins-demo-GitHub or jenkins-demo-haoshuwei.

iv. In the left-side navigation pane, click **Status**. If the build job is successful, Jenkins runs as normal on Kubernetes.



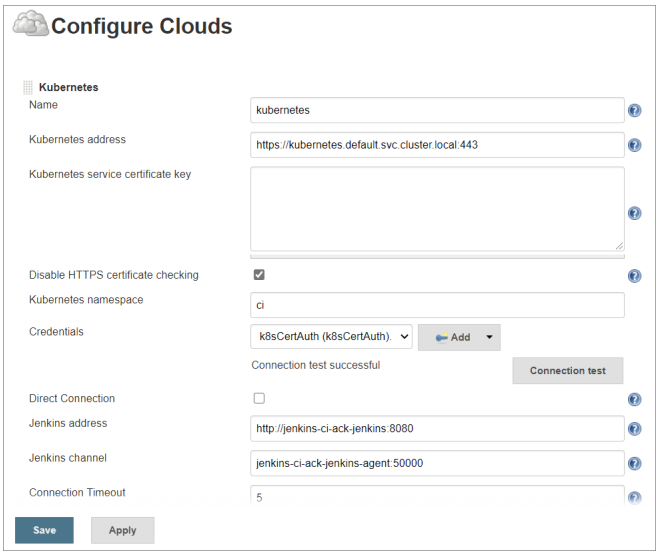
Configure Kubernetes Cloud

Jenkins uses a plug-in named Kubernetes to connect to Kubernetes clusters, and then dynamically creates and releases slave pods.

1. In the left-side navigation pane, click **Back to Dashboard**.
2. In the left-side navigation pane of the Jenkins dashboard, click **Manage Jenkins**.
3. On the **Manage Jenkins** page, click **Manage Nodes and Clouds** in the **System Configuration** section.
4. In the left-side navigation pane of the **Nodes** page, click **Configure Clouds**.
5. On the **Configure Clouds** page, click **Kubernetes Cloud details...**.

   Kubernetes Cloud is automatically configured when ack-jenkins is deployed. The following table describes the parameters.

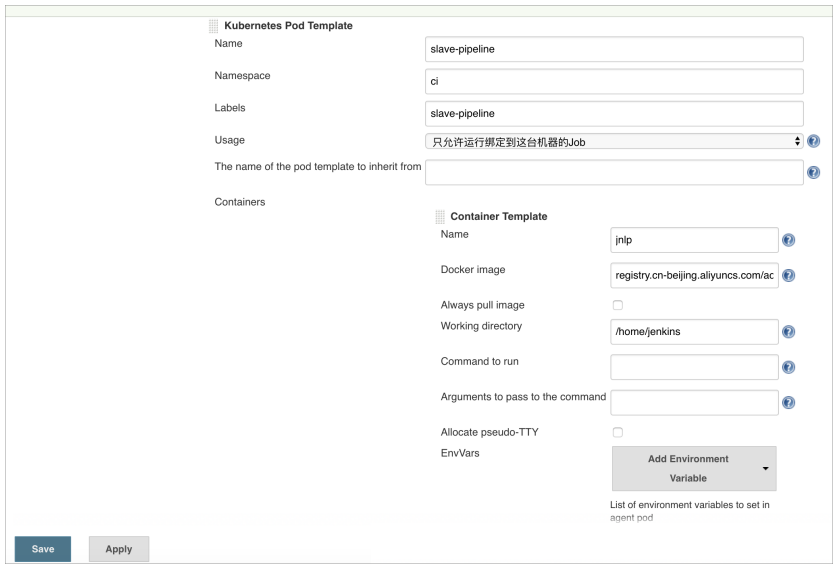
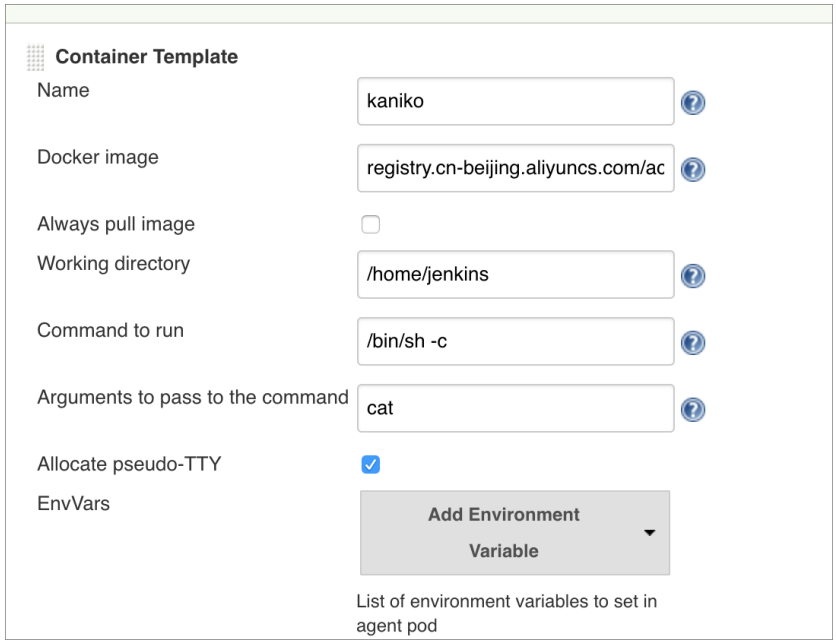
Parameter	Description

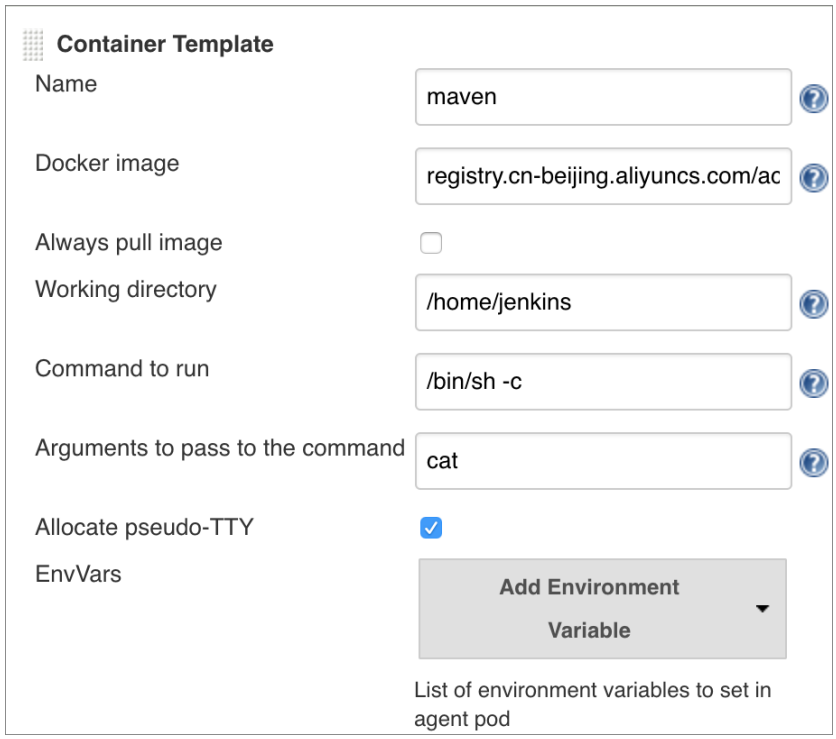
Parameter	Description
Name	Default value: `kubernetes`. 
Kubernetes address	Default value: `https://kubernetes.default.svc.cluster.local:443`. This indicates that Jenkins is installed in the cluster and can access the Kubernetes API server by using an internal endpoint.
Kubernetes namespace	Default value: `default`. This indicates that slave pods are dynamically created and destroyed in the `default` namespace.
Jenkins address	Default value: `http://ack-jenkins-default:8080`. This indicates the endpoint that the slave pods use to connect to the Jenkins master.
Jenkins channel	Default value: `ack-jenkins-default-agent:50000`. Slave pods use Java Network Launch Protocol (JNLP) to connect to the Jenkins master.

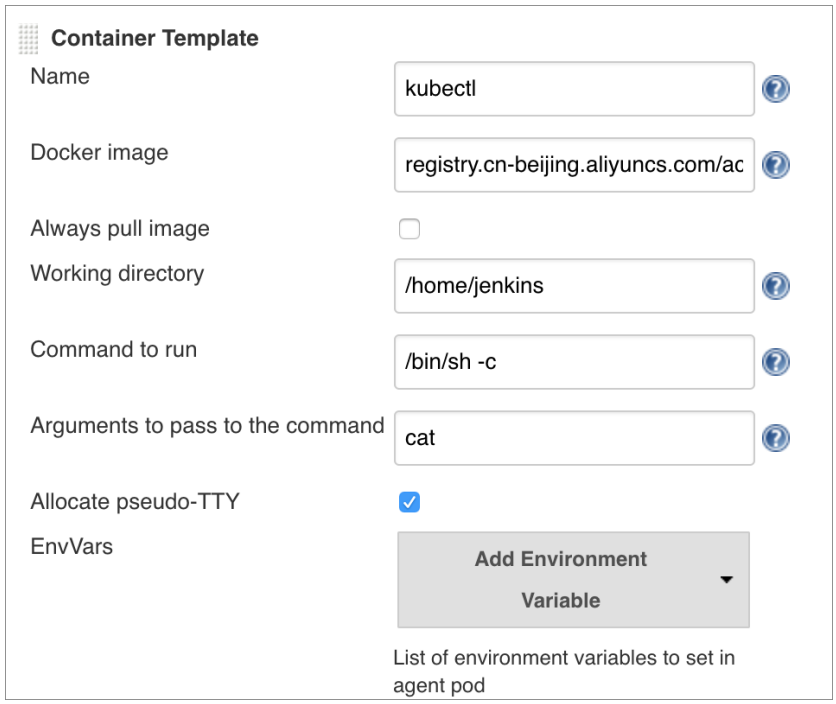
6. On the **Configure Clouds** page, click **Pod Templates...** and then click **Pod Templates details...**.

   A pod template is automatically configured when ack-jenkins is deployed. The following table describes the parameters.

Parameter	Description

Parameter	Description
Name	The name of the slave pod. Default value: `slave-pipeline`. 
Namespace	Default value: `jenkins`.
Labels	Build jobs select pod templates with matching labels to create slave pods. Default value: `slave-pipeline`.
	`jnlp` is used to connect to the Jenkins master.
	`kaniko` is used to build and push container images. 

Parameter	Description
Containers	`maven` is used to build and package applications. 

Parameter	Description
	`kubectl` is used to deploy applications. 

Configure the Maven cache for slave pods

Slave pods are dynamically created in the Kubernetes cluster and are scheduled to a random worker node. To ensure that each slave pod uses the **Maven cache**, a shared persistent volume (PV) is required.

1. Create Apsara File Storage NAS (NAS) volumes to provide shared persistent storage. For more information, see Mount a dynamically provisioned NAS volume.
   Run the following command to query the NAS volumes that are created in the **jenkins** namespace:

   ```
   kubectl -n jenkins get pvc
   ```

   Expected output:

   In the expected output, the persistent volume claim (PVC) named `ack-jenkins-default` provides persistent storage for the */var/jenkins_home* directory of the Jenkins master. The PVC named `nas-csi-pvc` provides shared persistent storage for the Maven cache.

   ```
   NAME                  STATUS   VOLUME                                     CAPACITY   AC
   CESS MODES   STORAGECLASS              AGE
   ack-jenkins-default   Bound    d-2ze8xhgzuw1t278j****                     20Gi       RW
   O            alicloud-disk-efficiency   3h16m
   nas-csi-pvc           Bound    nas-c6c3e703-58a9-484e-8ce5-630486ea****   20Gi       RW
   X            alicloud-nas-subpath       4m17s
   ```

2. (Optional)Add a ConfigMap volume in the pod template.

   To mount a custom settings file to a slave pod in Jenkins, you can create a ConfigMap and then specify the ConfigMap as a volume in the pod template.

i. Run the following command to create a ConfigMap:

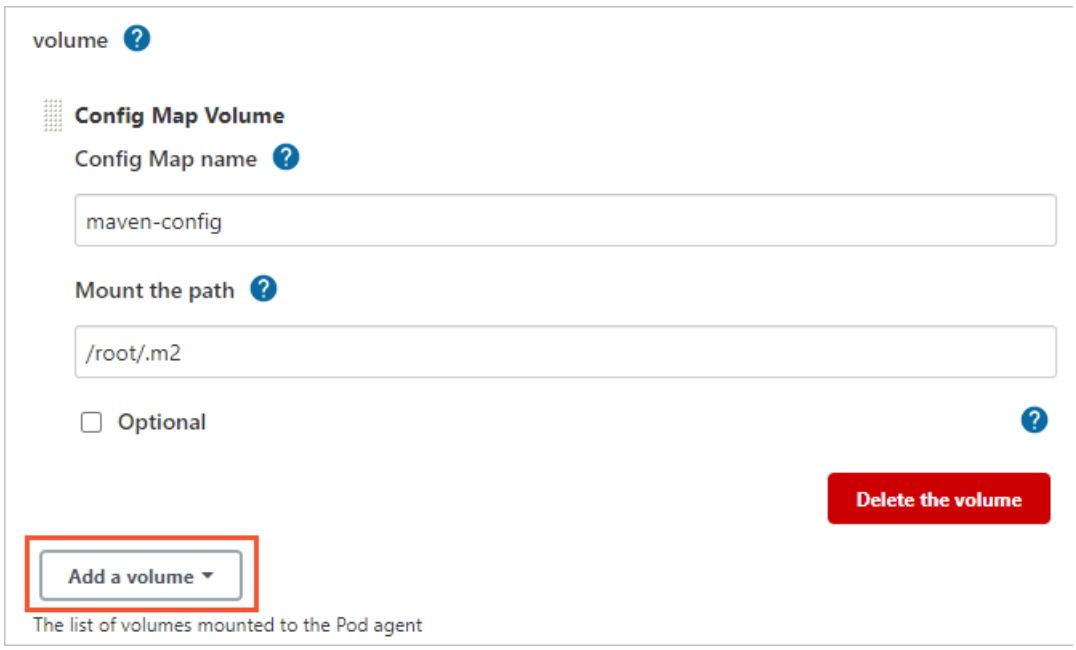
To modify the settings.xml file, you must first create a ConfigMap.

```
kubectl -n jenkins create configmap maven-config --from-file=settings.xml
```

ii. In the left-side navigation pane of the Jenkins dashboard, click **Manage Jenkins**. On the **Manage Jenkins** page, click **Manage Nodes and Clouds** in the **System Configuration** section. In the left-side navigation pane of the **Nodes** page, click **Configure Clouds**.

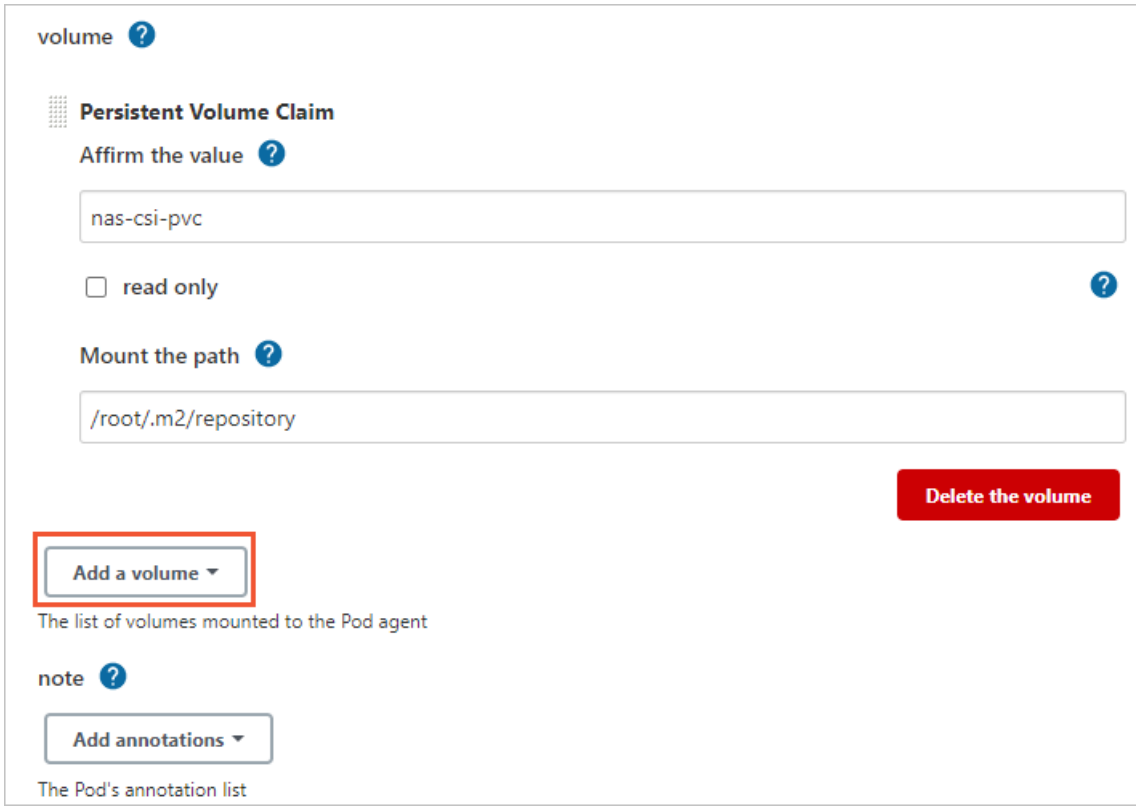
iii. On the **Configure Clouds** page, click **Pod Template** and then click **Pod Template details**.

iv. In the **Volumes** field, click **Add Volume** and then select **Config Map Volume**.



v. Click **Save**.

3. Add a PVC in the pod template. For more information, see Step 2.



Verify the result

i. On the Jenkins homepage, click **demo-pipeline**.

ii. In the left-side navigation pane of the Jenkins dashboard, click **Configure**.

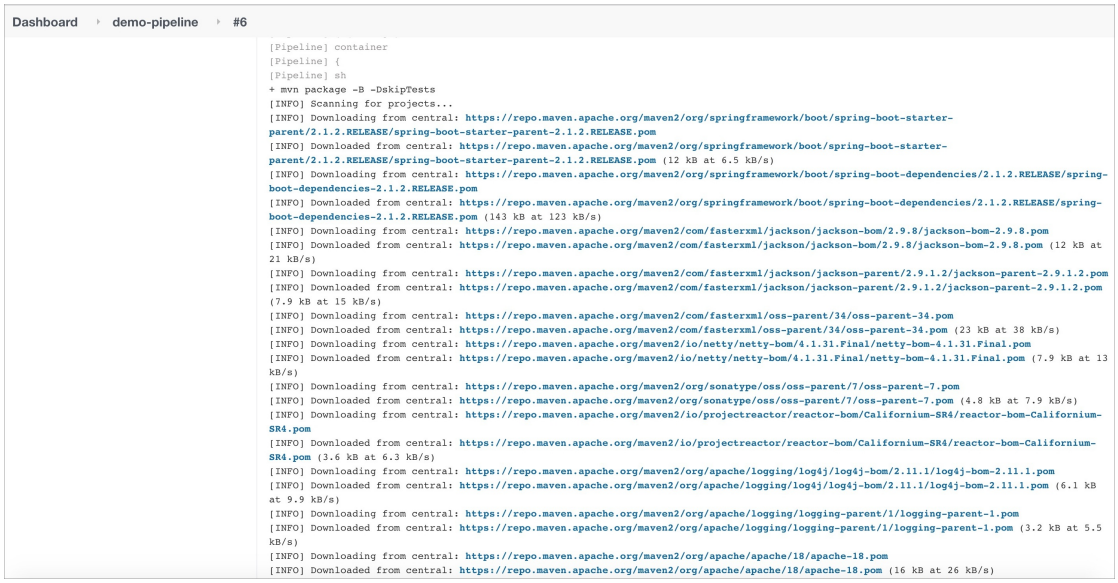
iii. Click the **Pipeline** tab.

iv. In the **Script Path** field, enter *Jenkinsfile.maven*.

v. Click **Save**.

vi. On the Jenkins homepage, click the icon to the right side of **demo-pipeline**. Then, click **Start building**.

When you build the **Maven** project for the first time, the system requires some time to download all dependencies.



When you build the **Maven** project again, the system can reference dependencies from the cache and quickly package the source code.

Use kaniko to build and push container images

When you use **kaniko** to push images, you must configure credentials that are used to access the image repository.

In this example, you must generate a *config.json* file on a Linux machine. This file is used to access the image repository. Do not generate the file on a macOS machine. The following example shows how to build and push the `registry.cn-hangzhou.aliyuncs.com/haoshuwei24/jenkins-demo:20200428` image.

1. Run the following command to log on to the image repository.

   The *config.json* file is automatically generated when you log on to the image repository.

   ```
   docker login -u <username> -p <password> registry.cn-hangzhou.aliyuncs.com
   ```

   Expected output:

   ```
   Login Succeeded
   ```

2. Create a Secret named *jenkins-docker-cfg* in the **jenkins** namespace by using the *config.json* file.

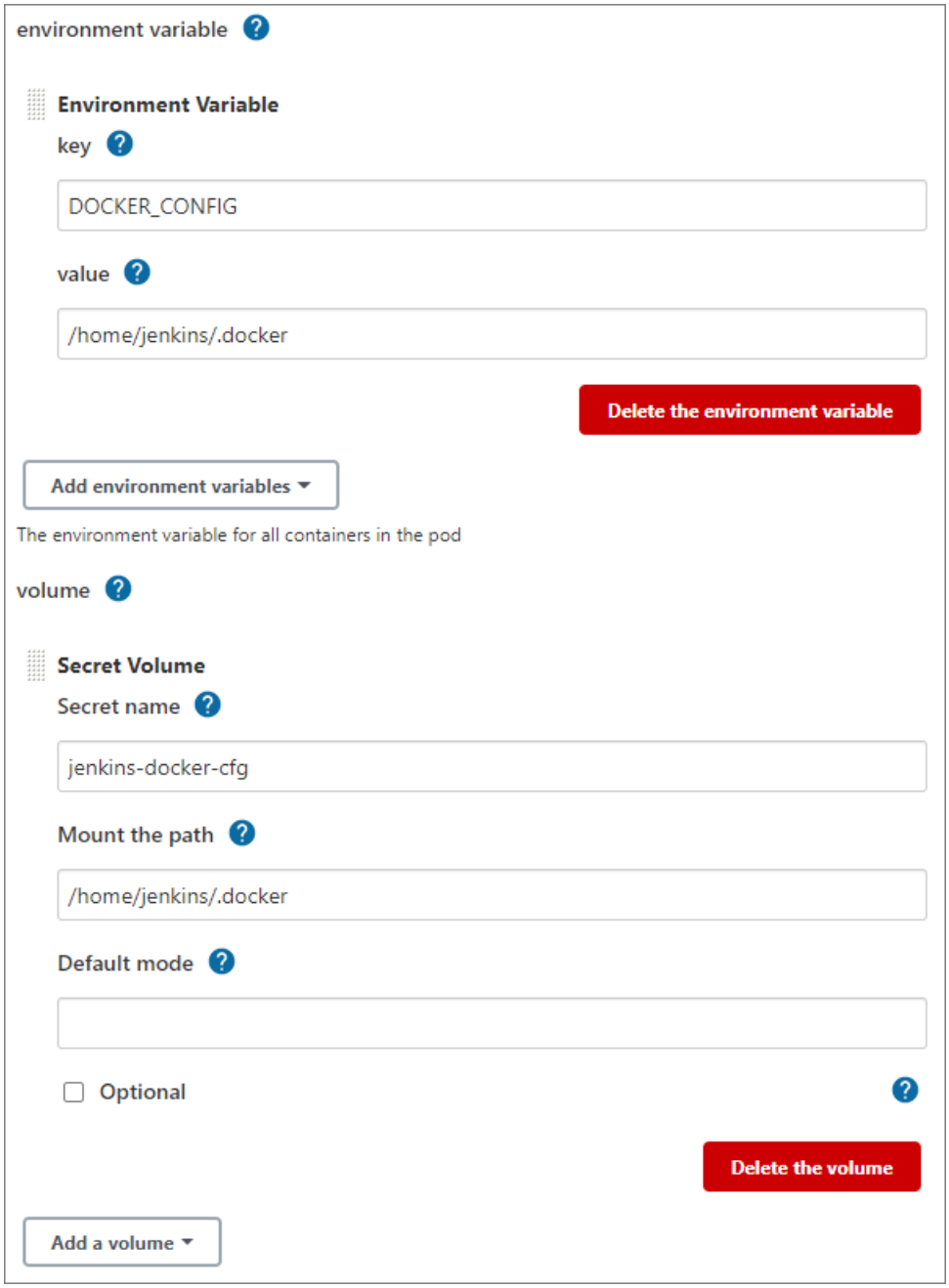
   ```
   kubectl create secret generic jenkins-docker-cfg -n jenkins --from-file=/root/.docker/config.json
   ```

3. Add an environment variable and a Secret volume in the pod template.

   i. In the left-side navigation pane of the Jenkins dashboard, click **Manage Jenkins**. On the **Manage Jenkins** page, click **Manage Nodes and Clouds** in the **System Configuration** section. In the left-side navigation pane of the **Nodes** page, click **Configure Clouds**.

   ii. On the **Configure Clouds** page, click **Pod Template** and then click **Pod Template details**.

iii. In the **Environment Variables** field, click **Add Environment Variable** and then select **Environment Variable**.



iv. In the **Volumes** field, click **Add Volume** and then select **Secret Volume**.

v. Click **Save**.

Verify the result

i. On the Jenkins homepage, click **demo-pipeline**.

ii. In the left-side navigation pane of the Jenkins dashboard, click **Configure**.

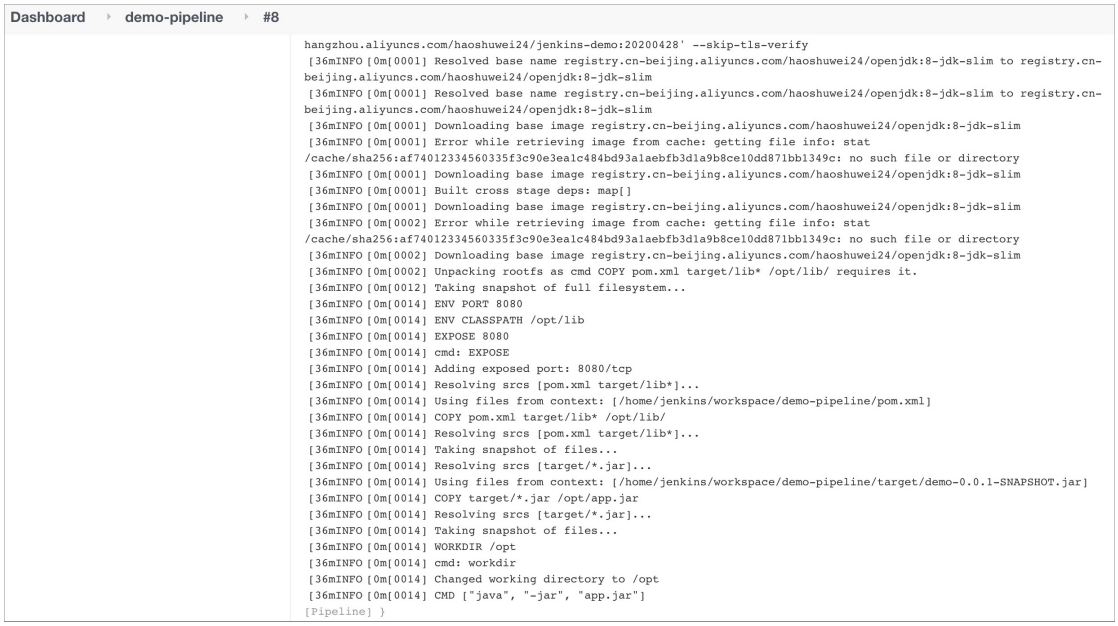
iii. Click the **Pipeline** tab.

iv. In the **Script Path** field, enter *Jenkinsfile.kaniko*.

v. Click **Save**.

vi. On the Jenkins homepage, click the icon to the right side of **demo-pipeline**. Modify the build parameters based on information about your image repository. Then, click **Start building**.

View the **kaniko** build log.



vii. Check whether the image is successfully pushed to the image repository.

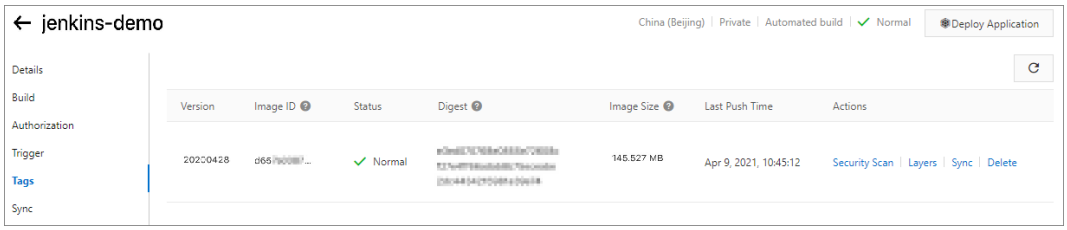
a.

b.

c. On the management page of the Container Registry Personal Edition instance, choose **Repository > Repositories** in the left-side navigation pane.

d. On the **Repositories** page, find the repository that you want to manage and click **Manage** in the Actions column.

e. In the left-side navigation pane, click **Tags** to check whether the image is successfully pushed to the repository.



Upgrade Jenkins

If you want to update the jenkins-master image in your Jenkins environment, you must create a snapshot of the disk volume that is mounted to the */var/jenkins_home* directory. This prevents data loss because the automation script of the new Jenkins master automatically writes data to the */var/jenkins_home* directory. For more information, see Use volume snapshots created from disks.

Related information

- Source code repository

- ACK
- kaniko

9.2. Use GitLab CI to run a GitLab runner and run a pipeline on Kubernetes

This topic describes how to install and register GitLab runners in a Kubernetes cluster and add a Kubernetes executor to build an application. The topic also provides step-by-step examples to implement a continuous integration (CI)/continuous delivery (CD) pipeline that includes stages, such as source code compilation, image build and push, and application deployment.

Background information

In the following example, a Java project is built and deployed in a Container Service for Kubernetes (ACK) cluster. This example shows how to use GitLab CI to run a GitLab runner, set a Kubernetes executor, and execute a CI/CD pipeline.

Create a GitLab project and upload sample code

1. Create a GitLab project.

   Address of the created GitLab project in this example:

   ```
   http://xx.xx.xx.xx/demo/gitlab-java-demo.git
   ```

2. Run the following commands to create a copy of the sample code and upload the copy to GitLab:

   ```
   git clone https://github.com/haoshuwei/gitlab-ci-k8s-demo.git
   git remote add gitlab http://xx.xx.xx.xx/demo/gitlab-java-demo.git
   git push gitlab master
   ```

Install GitLab Runner in an ACK cluster

1. Obtain registration information about GitLab runners.

i. Obtain registration information about the runners for this project.

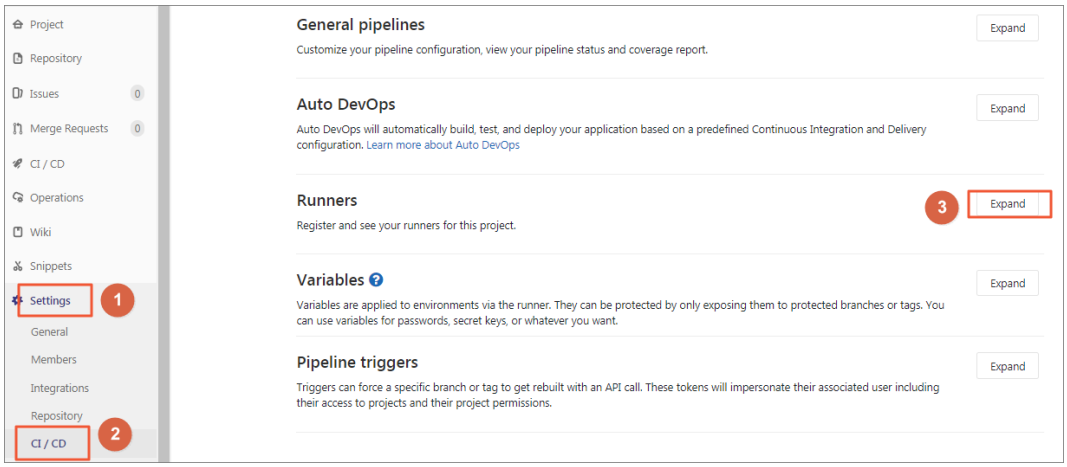
a. Log on to the GitLab console.

b. In the top navigation bar of the GitLab console, choose **Projects > Your projects**.

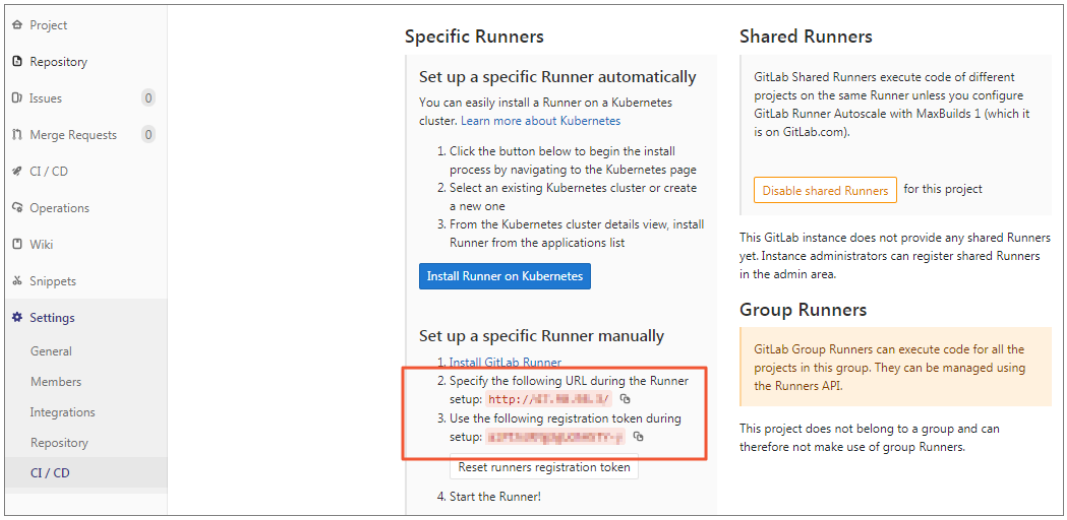
c. On the **Your projects** tab, select the project that you want to use.

d. In the left-side navigation pane, choose **Settings > CI / CD**.

e. Click **Expand** on the right side of **Runners**.



f. Copy the URL and registration token.



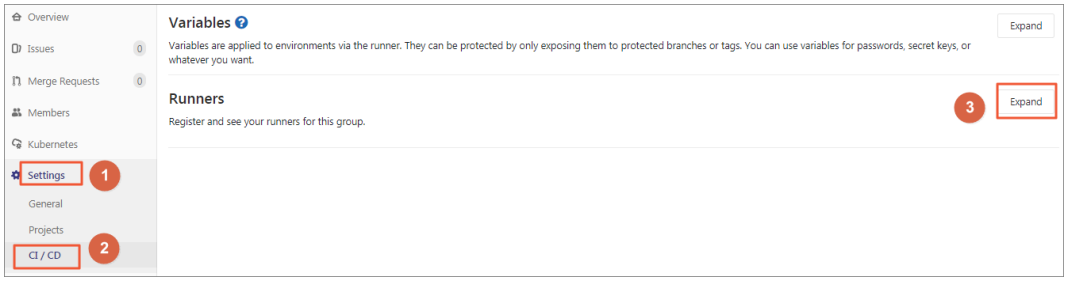
ii. Obtain registration information about group runners.

a. In the top navigation bar, choose **Groups > Your groups**.

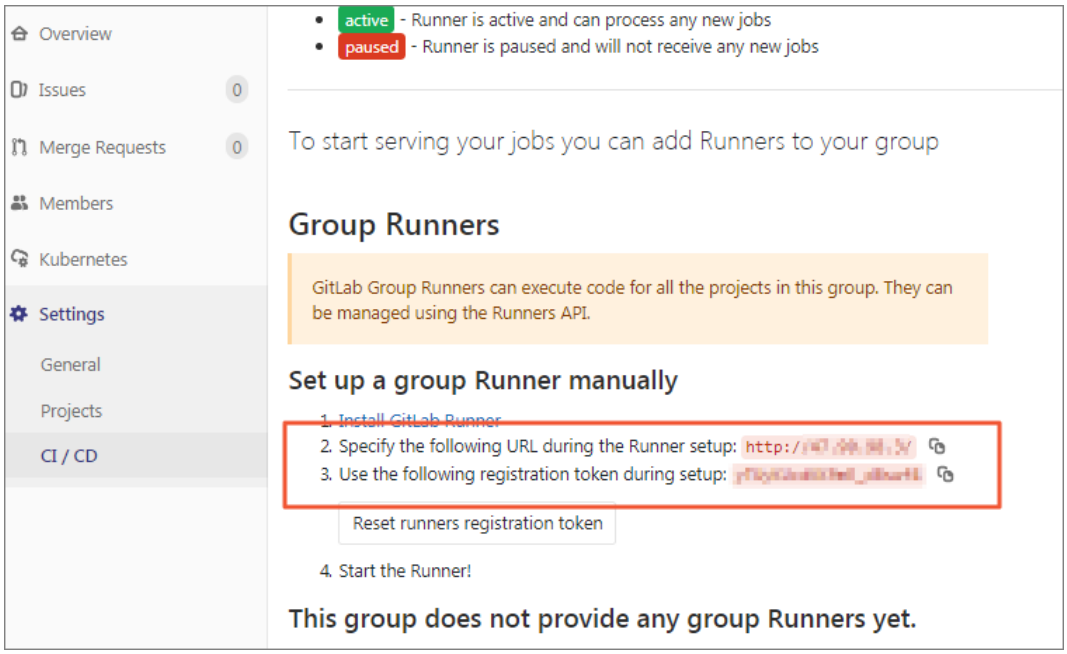
b. On the **Your groups** tab, select the required group.

c. In the left-side navigation pane, choose **Settings > CI / CD**.

d. Click **Expand** on the right side of **Runners**.



e. Copy the URL and registration token.



iii. Obtain registration information about shared runners.

> **Note** Only the administrator has the permission to perform this step.

a. In the top navigation bar, click

to go to the **Admin Area** page.

b. In the left-side navigation pane, choose **Overview > Runners**.

c. Copy the URL and registration token.

2. Run the following command to obtain a copy of the Helm chart of GitLab Runner:

```
git clone https://github.com/haoshuwei/gitlab-runner.git
```

Replace the gitlabUrl and runnerRegistrationToken fields as shown in the following code:

```
## GitLab Runner Image
##
image: gitlab/gitlab-runner:alpine-v11.4.0
## Specify a imagePullPolicy
##
imagePullPolicy: IfNotPresent
## Default container image to use for initcontainer
init:
  image: busybox
  tag: latest
## The GitLab Server URL (with protocol) that want to register the runner against
##
gitlabUrl: http://xx.xx.xx.xx/
## The Registration Token for adding new Runners to the GitLab Server. This must
## be retreived from your GitLab Instance.
##
runnerRegistrationToken: "AMvEWrBTBu-d8czE****"
## Unregister all runners before termination
##
unregisterRunners: true
## Configure the maximum number of concurrent jobs
##
concurrent: 10
## Defines in seconds how often to check GitLab for a new builds
##
checkInterval: 30
## For RBAC support:
##
rbac:
```

```
  create: true
  clusterWideAccess: false
## Configure integrated Prometheus metrics exporter
##
metrics:
  enabled: true
## Configuration for the Pods that that the runner launches for each new job
##
runners:
  ## Default container image to use for builds when none is specified
  ##
  image: ubuntu:16.04
  ## Specify the tags associated with the runner. Comma-separated list of tags.
  ##
  tags: "k8s-runner"
  ## Run all containers with the privileged flag enabled
  ## This will allow the docker:dind image to run if you need to run Docker
  ## commands. Please read the docs before turning this on:
  ##
  privileged: true
  ## Namespace to run Kubernetes jobs in (defaults to the same namespace of this releas
e)
  ##
  namespace: gitlab
  cachePath: "/opt/cache"
  cache: {}
  builds: {}
  services: {}
  helpers: {}
resources: {}
```

3. Run the following commands to install GitLab Runner:
   - Package the Helm chart of GitLab Runner.

     ```
     helm package .
     ```

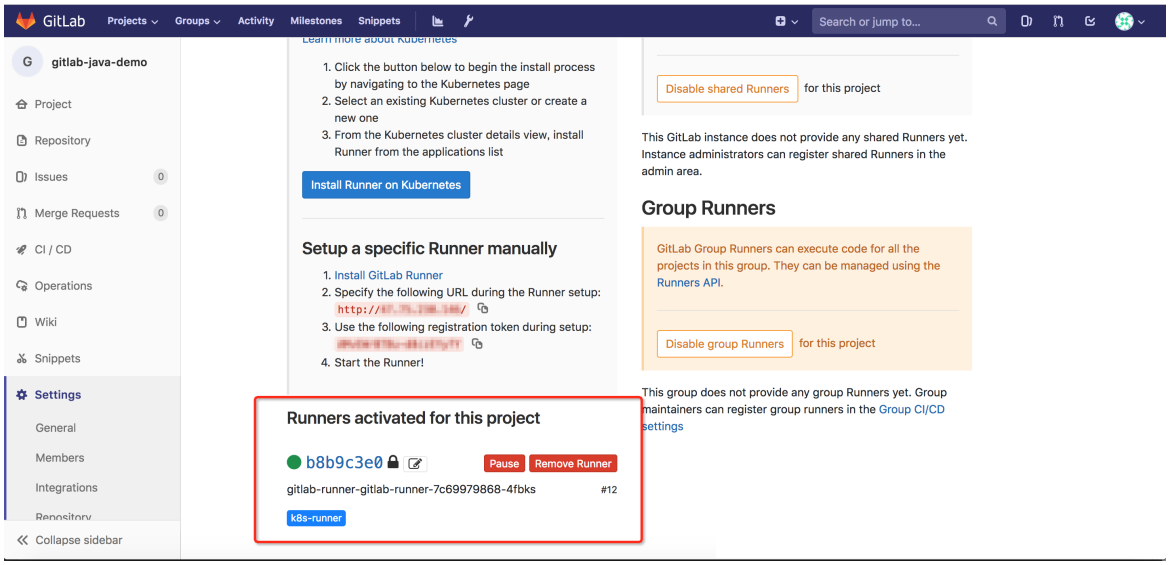
     Expected output:

     ```
     Successfully packaged chart and saved it to: /root/gitlab/gitlab-runner/gitlab-runner
     -0.1.37.tgz
     ```

   - Install GitLab Runner.

     ```
     helm install --namespace gitlab gitlab-runner *.tgz
     ```

Check whether the related Deployment or pods have been started. If the related Deployment or pods have been started, the registered GitLab runners will be displayed in GitLab, as shown in the following figure.



Cache settings

GitLab Runner has a limited cache capacity. You must mount volumes to store cache data. In the preceding example, the */opt/cache* directory is used as the default cache directory after GitLab Runner is installed. You can modify the `runners.cachePath` field in the *values.yaml* file to change the cache directory.

For example, to create a cache for Apache Maven, add the `MAVEN_OPTS` variable to the `variables` field and specify a local cache directory. The following code is an example:

```
variables:
  KUBECONFIG: /etc/deploy/config
  MAVEN_OPTS: "-Dmaven.repo.local=/opt/cache/.m2/repository"
```

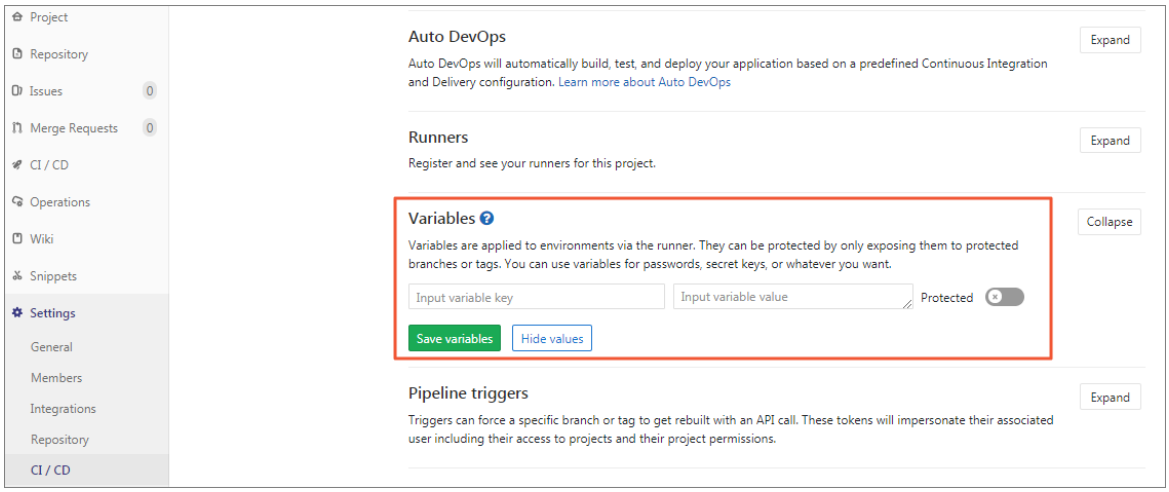
Modify the following fields in the *templates/configmap.yaml* file to mount docker.sock and the volume to store cache data:

```
cat >>/home/gitlab-runner/.gitlab-runner/config.toml <<EOF
            [[runners.kubernetes.volumes.pvc]]
              name = "gitlab-runner-cache"
              mount_path = "{{ .Values.runners.cachePath }}"
            [[runners.kubernetes.volumes.host_path]]
              name = "docker"
              mount_path = "/var/run/docker.sock"
              read_only = true
              host_path = "/var/run/docker.sock"
    EOF
```

Set global variables

1. In the top navigation bar, choose **Projects > Your projects**.
2. On the **Your projects** tab, select the project that you want to use.

3. In the left-side navigation pane, choose **Settings > CI / CD**.
4. Click **Expand** on the right side of **Variables**. Add environment variables for GitLab Runner. In this example, add the following variables:



   - REGISTRY_USERNAME: The username for logging on to the image repository.
   - REGISTRY_PASSWORD: The password for logging on to the image repository.
   - kube_config: The KubeConfig as an encoded string.

   Run the following command to convert KubeConfig into an encoded string:

```
echo $(cat ~/.kube/config | base64) | tr -d " "
```

Edit the .gitlab-ci.yml file

Use the *.gitlab-ci.yml* file to compile the source code, build an image, push the image, and deploy the application for the Java demo project. For more information, refer to .gitlab-ci.yml.example of the gitlabci-java-demo project.

The following template is an example of the *.gitlab-ci.yml* file:

```
image: docker:stable
stages:
  - package
  - docker_build
  - deploy_k8s
variables:
  KUBECONFIG: /etc/deploy/config
  MAVEN_OPTS: "-Dmaven.repo.local=/opt/cache/.m2/repository"
mvn_build_job:
  image: maven:3.6.2-jdk-14
  stage: package
  tags:
    - k8s-runner
  script:
    - mvn package -B -DskipTests
    - cp target/demo.war /opt/cache
docker_build_job:
  image: docker:latest
  stage: docker_build
  tags:
    - k8s-runner
  script:
    - docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD registry.cn-beijing.aliyuncs
.com
    - mkdir target
    - cp /opt/cache/demo.war target/demo.war
    - docker build -t registry.cn-beijing.aliyuncs.com/haoshuwei24/gitlabci-java-demo:$CI_P
IPELINE_ID .
    - docker push registry.cn-beijing.aliyuncs.com/haoshuwei24/gitlabci-java-demo:$CI_PIPEL
INE_ID
deploy_k8s_job:
  image: registry.cn-hangzhou.aliyuncs.com/haoshuwei24/kubectl:1.16.6
  stage: deploy_k8s
  tags:
    - k8s-runner
  script:
    - mkdir -p /etc/deploy
    - echo $kube_config |base64 -d > $KUBECONFIG
    - sed -i "s/IMAGE_TAG/$CI_PIPELINE_ID/g" deployment.yaml
    - cat deployment.yaml
    - kubectl apply -f deployment.yaml
```

The *.gitlab-ci.yml* file defines a pipeline that consists of the following stages:

```
image: docker:stable  # By default, if no image is specified for each stage of the pipeline
, the docker:stable image is used.
stages:
  - package                # Package the source code.
  - docker_build         # Build, package, and push the image.
  - deploy_k8s           # Deploy the application.
variables:
  KUBECONFIG: /etc/deploy/config   # Define the global variable KUBECONFIG.
```

- Package the source code by using Apache Maven.

```
mvn_build_job:     # The job name.
  image: maven:3.6.2-jdk-14  # The image that is used in this stage.
  stage: package      # The name of the stage.
  tags:                     # The tag of the GitLab Runner image.
    - k8s-runner
  script:
    - mvn package -B -DskipTests  # Run the build script.
    - cp target/demo.war /opt/cache  # Save the build output to cache.
```

- Build, package, and push the image.

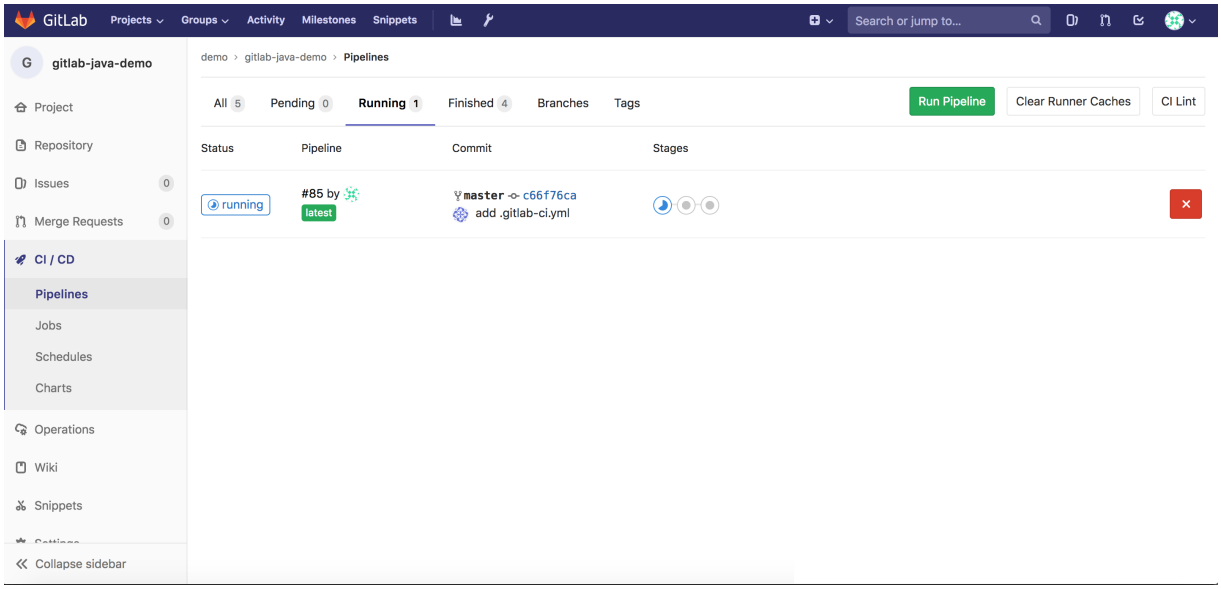
```
docker_build_job:  # The job name.
  image: docker:latest # The image that is used in this stage.
  stage: docker_build   # The name of the stage.
  tags:                      # The tag of the GitLab Runner image.
    - k8s-runner
  script:
    - docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD registry.cn-beijing.aliyun
cs.com   # Log on to the image repository.
    - mkdir target
    - cp /opt/cache/demo.war target/demo.war
    - docker build -t registry.cn-beijing.aliyuncs.com/haoshuwei24/gitlabci-java-demo:$CI
_PIPELINE_ID .     # Package the Docker image. Use the pipeline ID as its tag.
    - docker push registry.cn-beijing.aliyuncs.com/haoshuwei24/gitlabci-java-demo:$CI_PIP
ELINE_ID      # Push the Docker image.
```

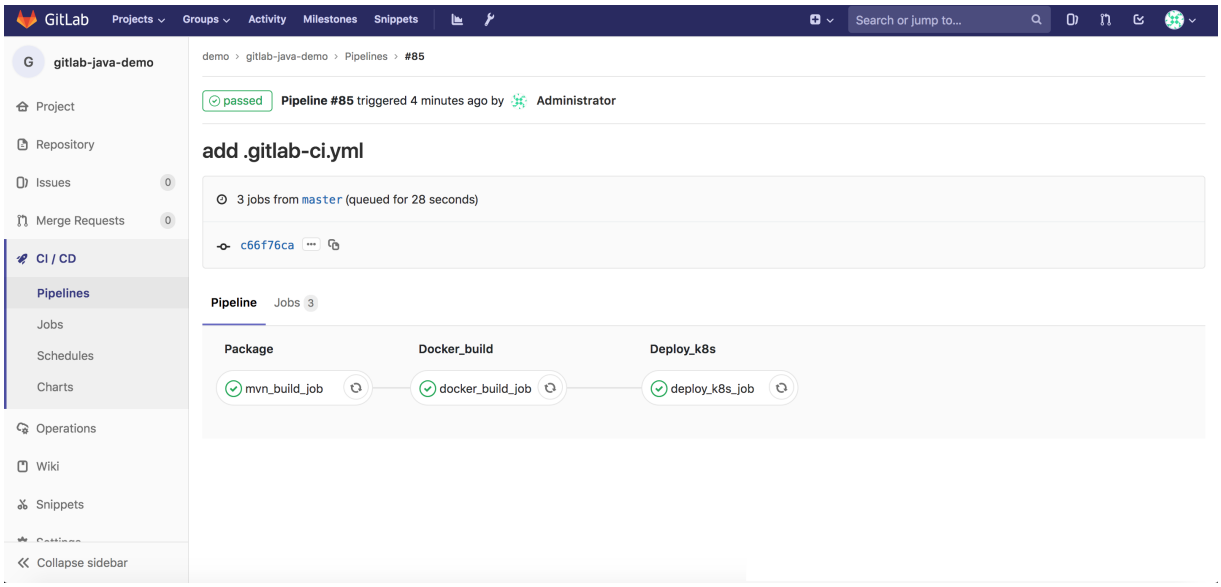
- Deploy the application.

```
deploy_k8s_job:   # The job name.
  image: registry.cn-hangzhou.aliyuncs.com/haoshuwei24/kubectl:1.16.6   # The image that
is used in this stage.
  stage: deploy_k8s  # The name of the stage.
  tags:                      # The tag of the GitLab Runner image.
    - k8s-runner
  script:
    - mkdir -p /etc/deploy
    - echo $kube_config |base64 -d > $KUBECONFIG   # Configure the config file that is us
ed to connect to the Kubernetes cluster.
    - sed -i "s/IMAGE_TAG/$CI_PIPELINE_ID/g" deployment.yaml  # Dynamically replace the i
mage tag in the deployment file.
    - cat deployment.yaml
    - kubectl apply -f deployment.yaml
```

Execute the pipeline

After you submit the *.gitlab-ci.yml* file, the gitlab-java-demo project will automatically detect this file and execute the pipeline, as shown in the following figures.





Access the application

By default, if no namespace is specified in the deployment file, the application is deployed in the GitLab namespace.

```
kubectl -n gitlab get svc
```

Expected output:

```
NAME        TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)        AGE
java-demo   LoadBalancer   172.19.9.252   xx.xx.xx.xx   80:32349/TCP   1m
```

Visit xx.xx.xx.xx/demo by using your browser to access the application.

For more information about ACK, see Container Service for Kubernetes.

For more information about GitLab CI, see GitLab CI.

9.3. Elastic and cost-effective CI/CD based on ASK

This topic describes the scenario and benefits of an elastic and cost-effective continuous integration and continuous delivery (CI/CD) solution that is based on lightweight serverless Kubernetes (ASK). This topic also describes the problems that are solved by this solution, and provides the architecture diagram of the solution and links to the relevant references.

Scenario

Elastic Container Instance-based lightweight ASK and Apsara File Storage NAS can be combined to achieve an automated CI/CD solution that features high service availability, elastic scaling, high scalability, and cost-effectiveness.

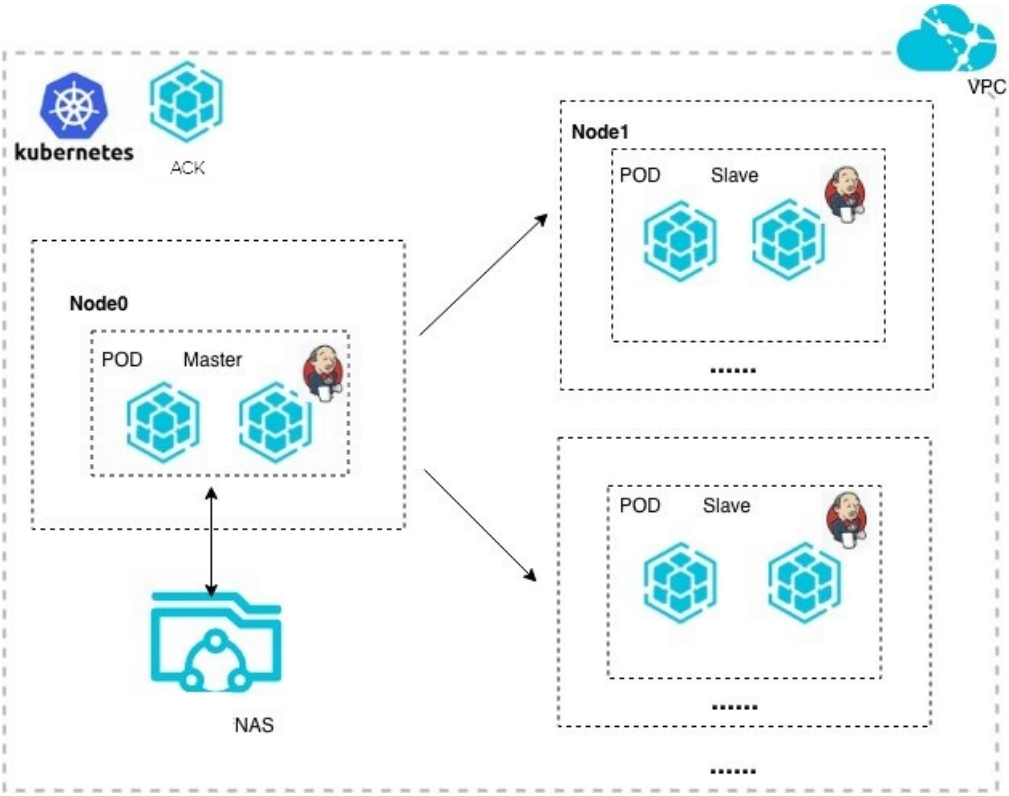
Benefits

- Services are highly available.
- Resources can be scaled to maintain high utilization.
- Resources are highly scalable.

Problems solved

- Single points of failure (SPOF)s of the master nodes in clusters.
- Low utilization rates of cluster resources.
- Low scalability of cluster resources.

Architecture



References

For more information, see best practice An elastic and cost-effective CI/CD solution based on ASK.

9.4. Use Bamboo to deploy a Bamboo agent in an ACK cluster and run a build plan

This topic describes how to use Bamboo to deploy a remote agent in a Container Service for Kubernetes (ACK) cluster and run a build plan for an application. In this topic, a sample application compiled in Java is created and deployed in an ACK cluster.

Prerequisites

- An ACK cluster is created. For more information, see Create an ACK managed cluster.
- A Bamboo server is created.

Source code

Address of the GitHub project in this example:

```
https://github.com/AliyunContainerService/jenkins-demo.git
```

The source code under the `bamboo` branch is used.

Step 1: Deploy a remote agent in an ACK cluster

1. Create a Secret named kaniko-docker-cfg.

   The kaniko-docker-cfg Secret is used to authenticate access to an image registry when you run a build plan and push an image to the image registry.

   i. Log on to your Linux server as a **root** user and run the following command to generate the */root/.docker/config.json* file:

   ```
   docker login registry.cn-hangzhou.aliyuncs.com
   ```

   ii. Use Cloud Shell to connect to the created ACK cluster. Run the following command to create the kaniko-docker-cfg Secret:

   ```
   kubectl -n bamboo create secret generic kaniko-docker-cfg  --from-file=/root/.docker/config.json
   ```

2. Deploy the Bamboo agent.

   Create a service account and ClusterRoleBinding to configure permissions of the Bamboo agent in the ACK cluster.

   i. Create a file named *bamboo-agent.yaml* and copy the following content to the file. Run the `kubectl -n bamboo apply -f bamboo-agent.yaml` command to deploy the Bamboo agent.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  namespace: bamboo
  name: bamboo
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: bamboo-cluster-admin
subjects:
  - kind: ServiceAccount
    name: bamboo
    namespace: bamboo
roleRef:
  kind: ClusterRole
  name: cluster-admin
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: bamboo-agent
spec:
  replicas: 1
  selector:
    matchLabels:
      app: bamboo-agent
  template:
    metadata:
      labels:
        app: bamboo-agent
    spec:
      serviceAccountName: bamboo
      containers:
      - name: bamboo-agent
        env:
        - name: BAMBOO_SERVER_URL
          value: http://xx.xx.xx.xx:8085
        image: registry.cn-hangzhou.aliyuncs.com/haoshuwei/docker-bamboo-agent:v1
        imagePullPolicy: Always
        volumeMounts:
          - mountPath: /root/.docker/
            name: kaniko-docker-cfg
      volumes:
        - name: kaniko-docker-cfg
          secret:
            secretName: kaniko-docker-cfg
```

ii. After the Bamboo agent is deployed, run the following command to query the log data of the agent:

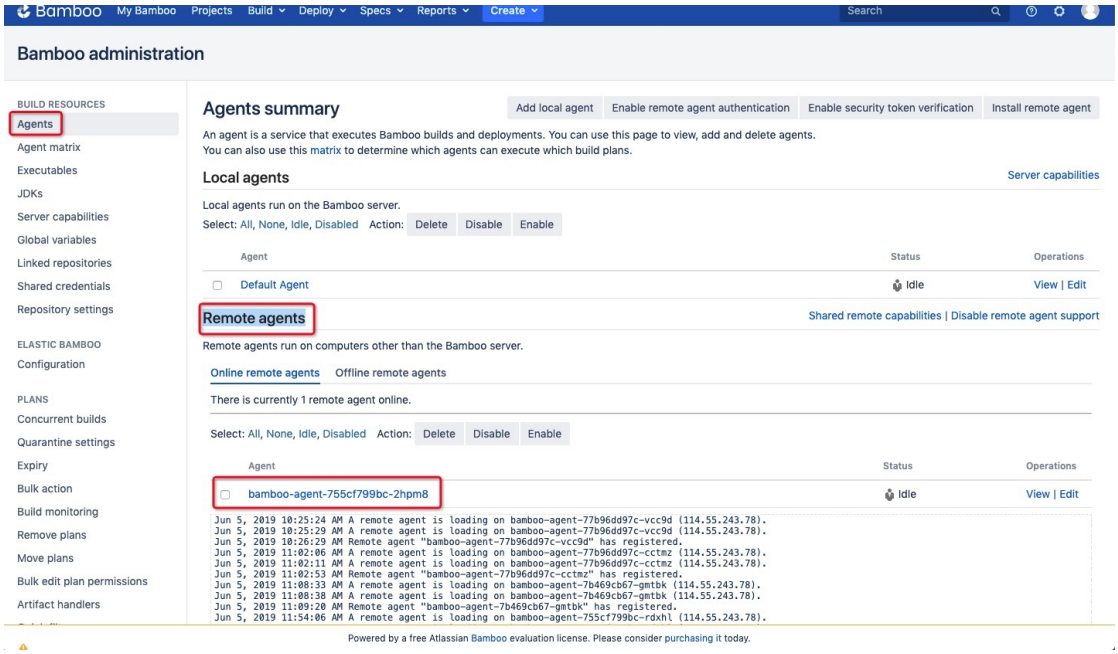
```
kubectl -n bamboo logs -f <bamboo agent pod name>
```

> **Note** Replace *bamboo agent pod name* with the actual pod name.

iii. After the remote agent is deployed, log on to the Bamboo console. In the upper-right corner of the console, click

and select **Agent** from the drop-down list.

On the **Bamboo administration** page, you can find the deployed agent.



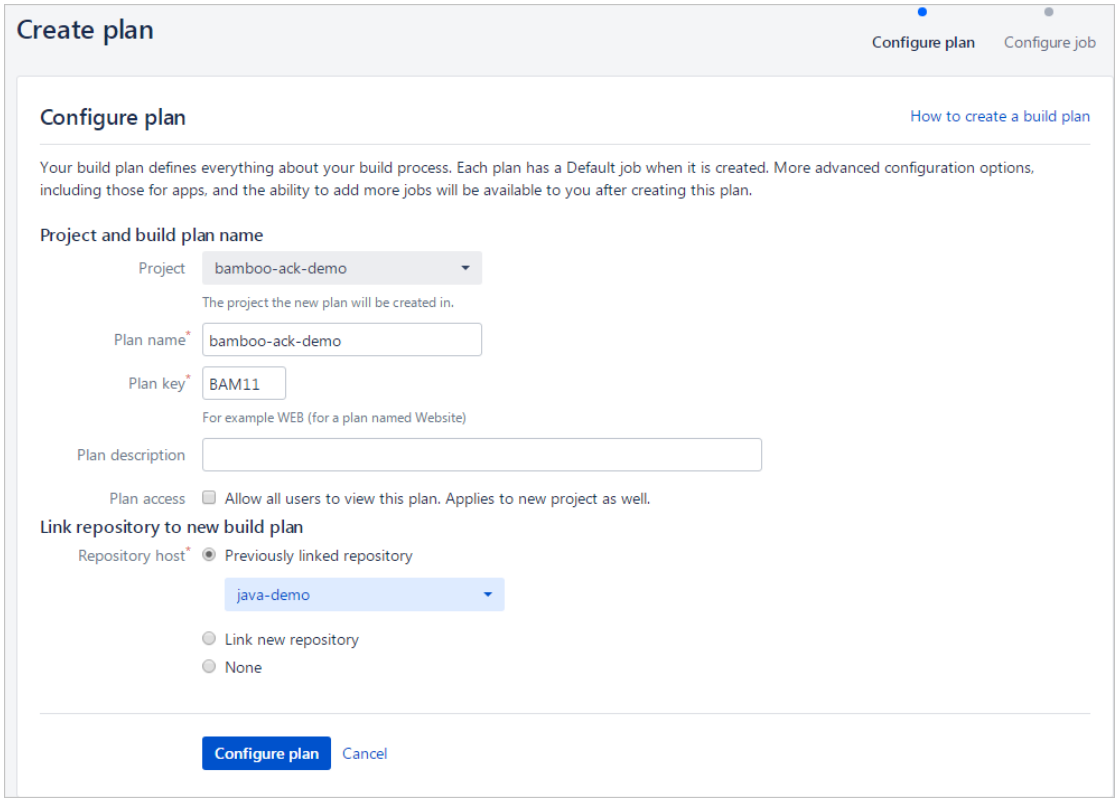
Step 2: Configure a build plan

You can configure a build plan to pull the source code, compile and package the source code, package and push the container image, and deploy the application.

1. Create a build plan.

   i. Log on to the Bamboo console, choose **Create > Create plan** to go to the **Configure plan** page.

ii. Select *bamboo-ack-demo* from the **Project** drop-down list. Set Plan name, Plan key, and Plan description. Select *java-demo* from the Repository host drop-down list. Then, click **Configure plan**.

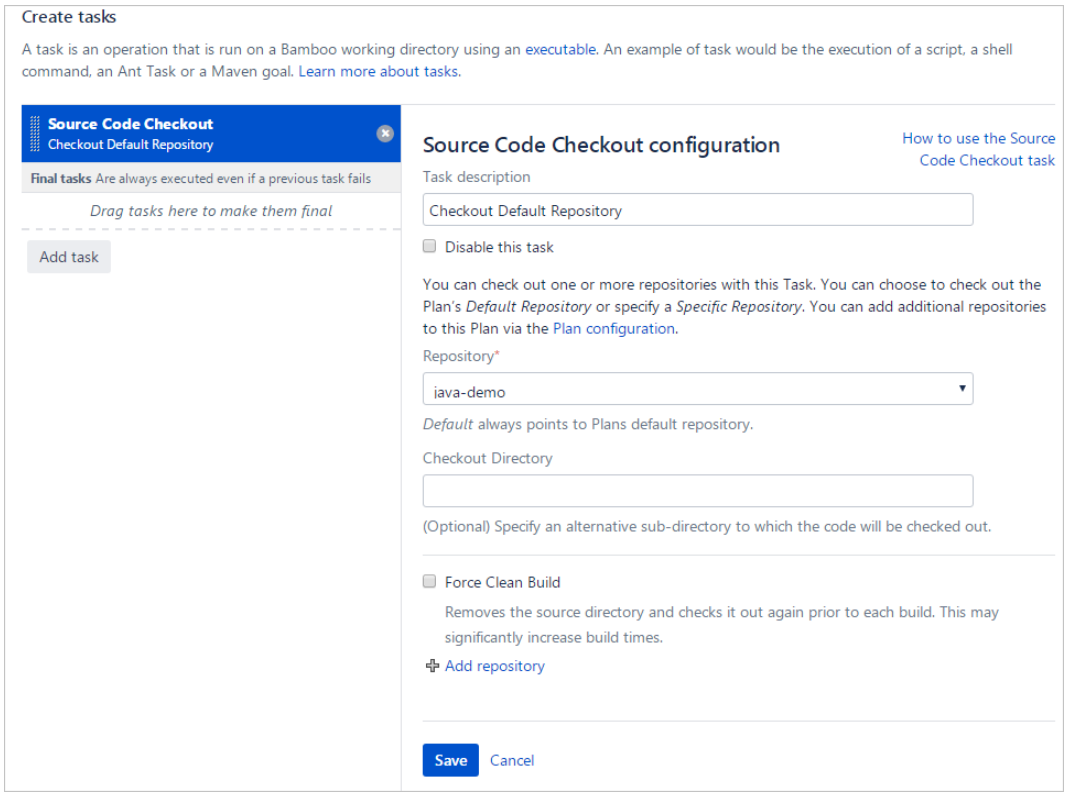


2. The **Configure Job** page appears. On the Configure Job page, configure the stages and add jobs.

i. Pull the source code.

When you create the build plan, you have selected *java-demo* from the Repository host drop-down list. This means that you have configured from where to pull the source code. If you want to change the code repository, perform the following steps.

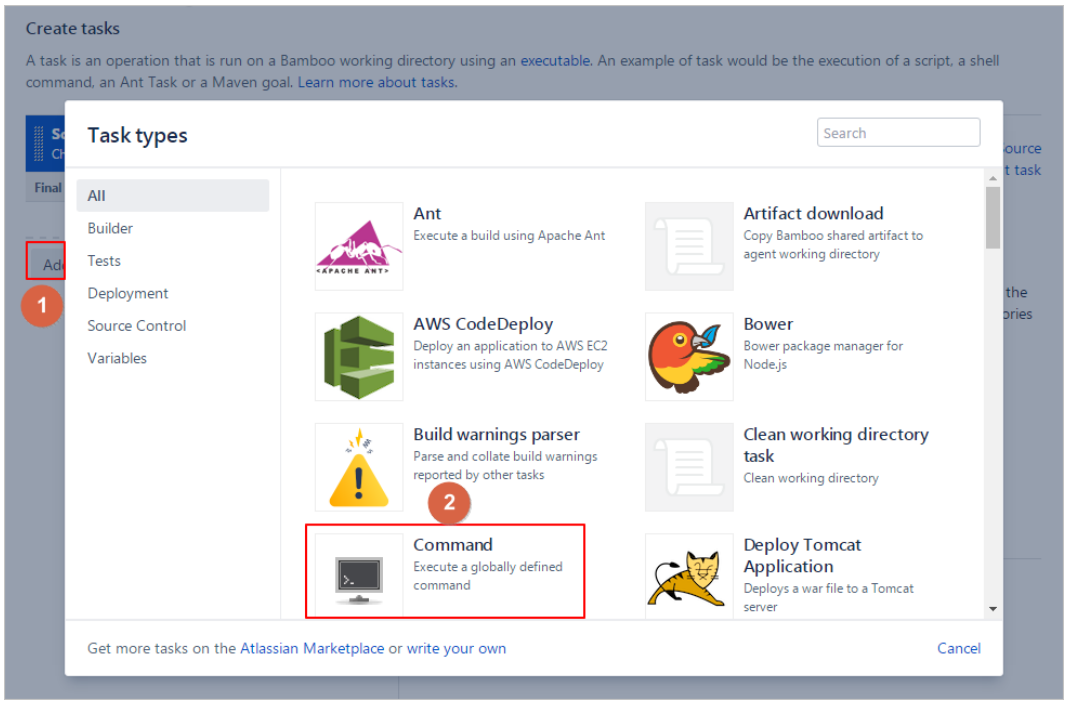
a. In the **Create tasks** section, click **Source Code Checkout**.



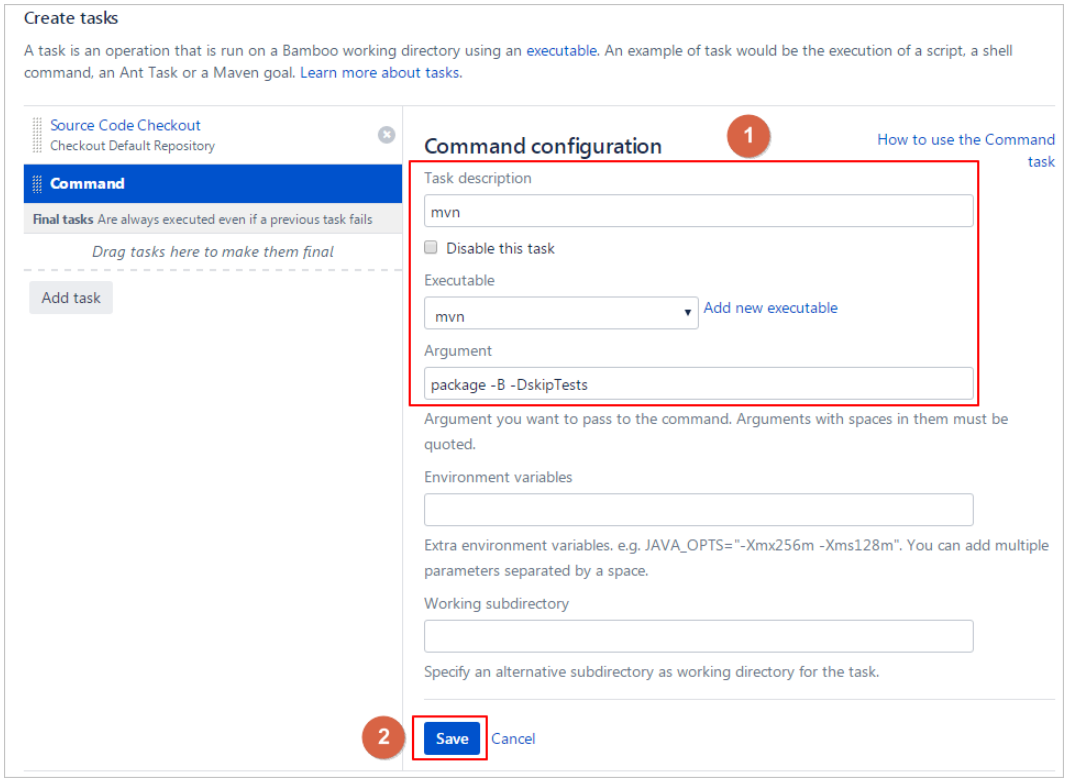
b. In the **Source Code Checkout configuration** panel, modify Repository host based on your requirements and click **Save**.

ii. Use Maven to package the source code.

a. In the **Create tasks** section, click **Add task**. In the **Task types** dialog box, select **Command**.

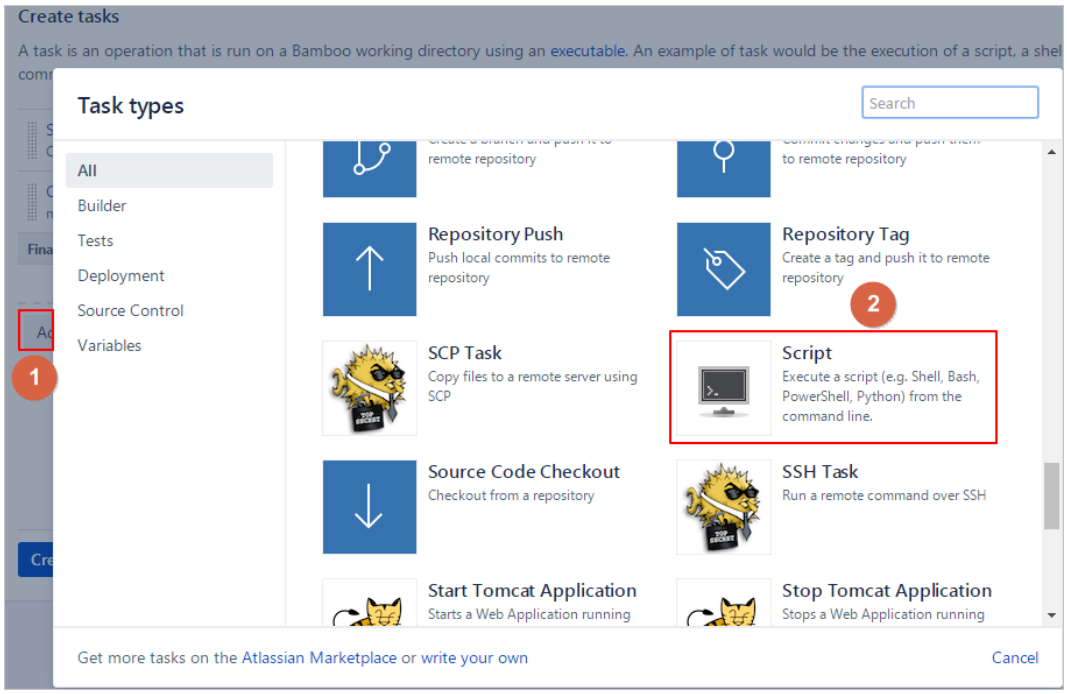


b. In the **Command configuration** panel, set **Task description**, **Executable**, and **Argument**. Click **Save**.

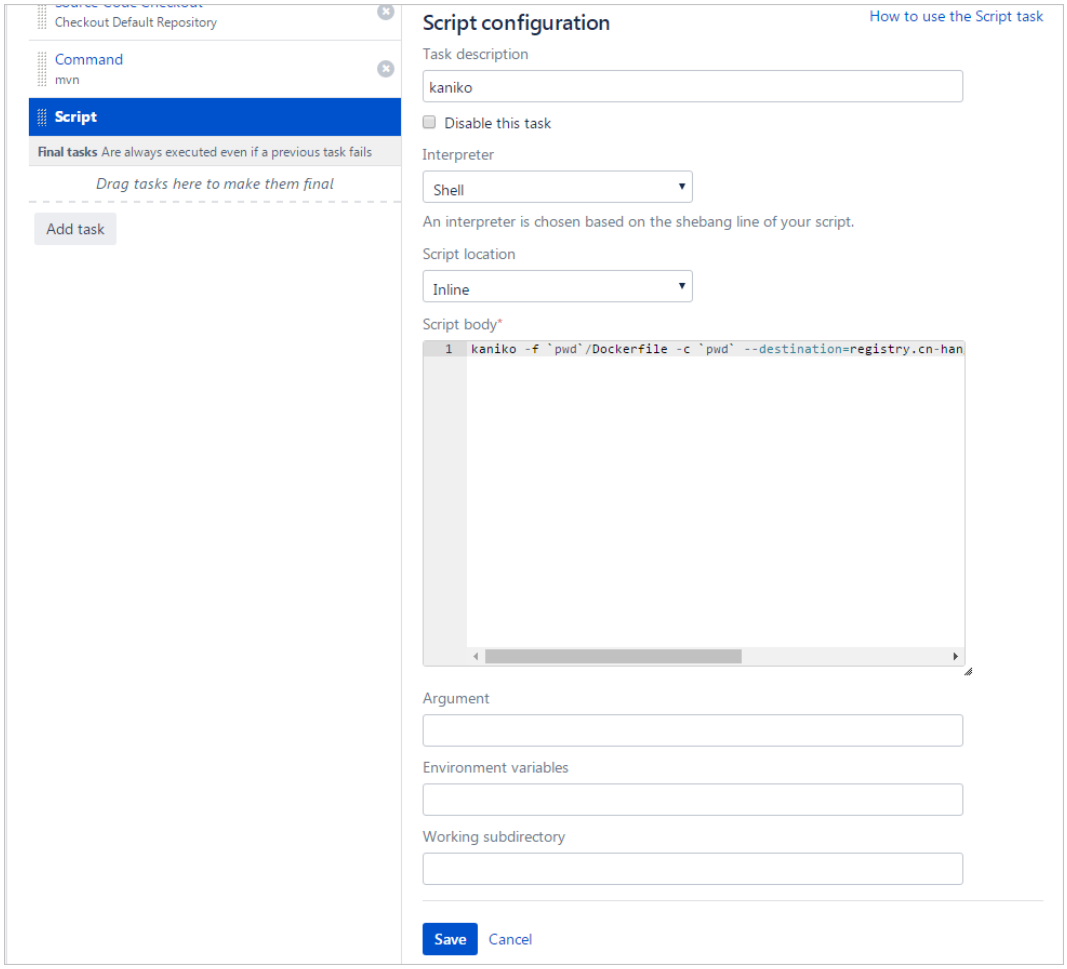


iii. Use kaniko to package and push the container image to the image registry.

a. In the **Create tasks** section, click **Add task**. In the **Task types** dialog box, select **Script**.



b. In the **Script configuration** panel, set **Task description** and **Script location**. Use the default settings for the other parameters. Click **Save**.

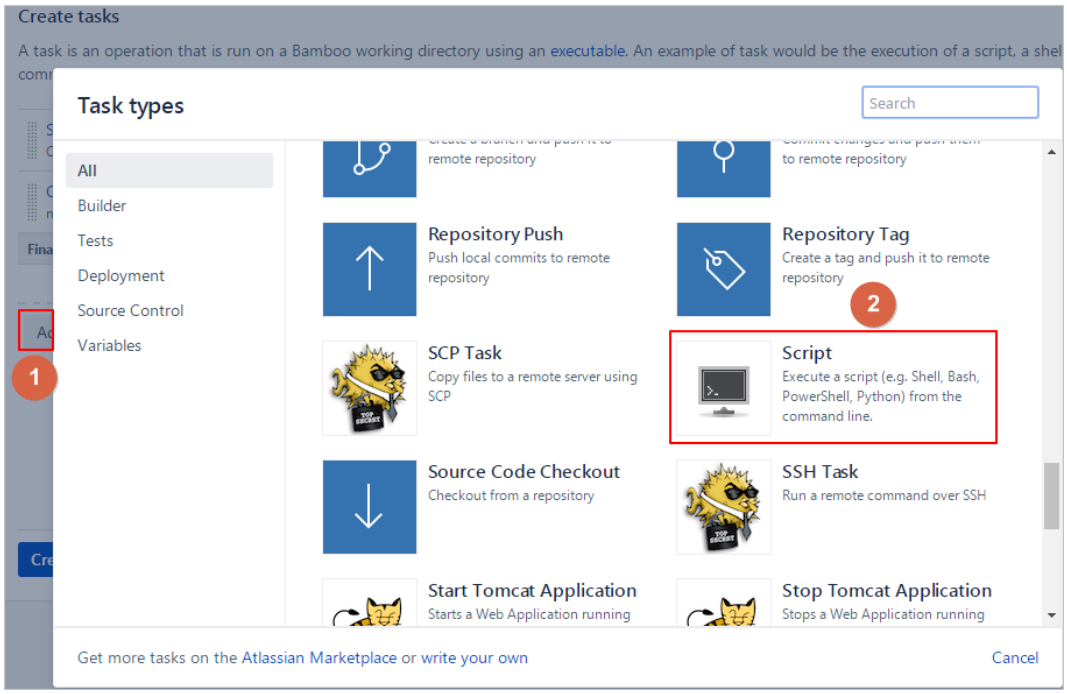


In this example, **Script location** is specified as:

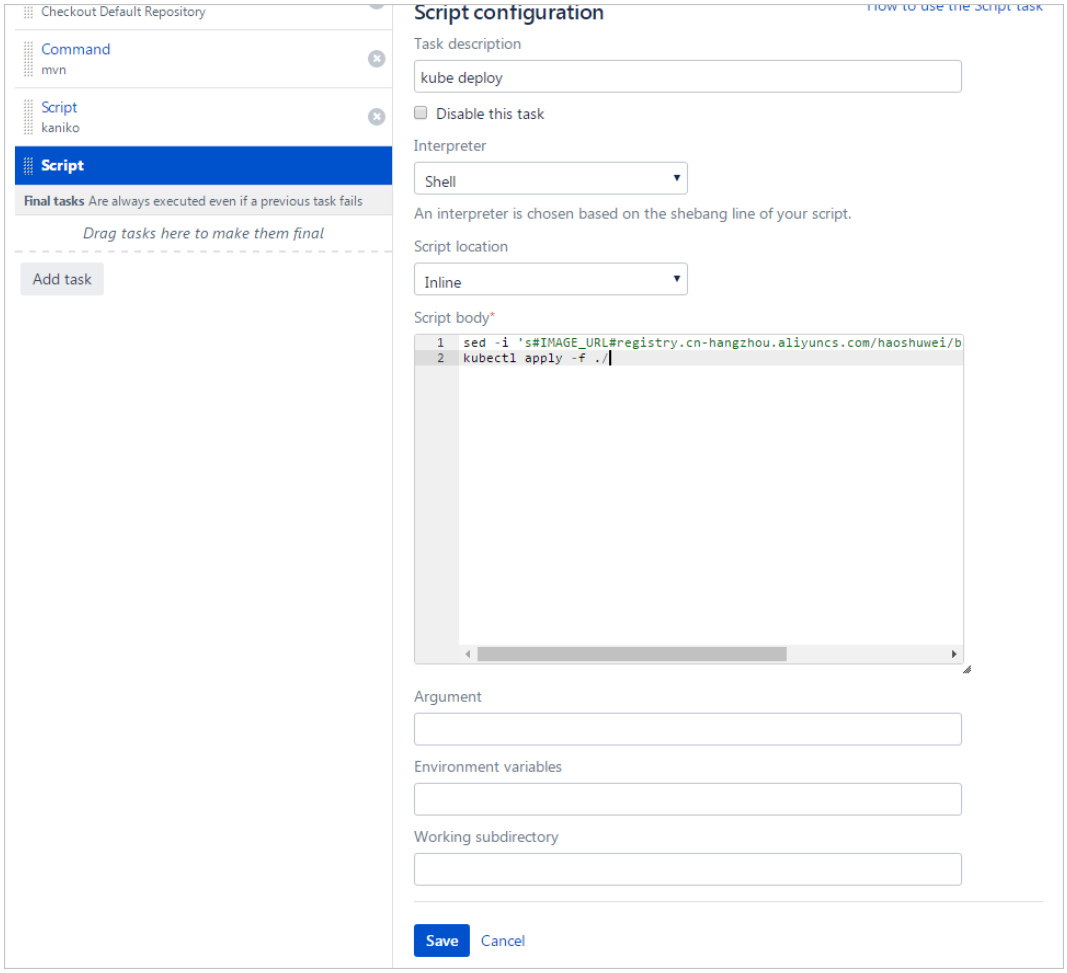
```
kaniko -f `pwd`/Dockerfile -c `pwd` --destination=registry.cn-hangzhou.aliyuncs
.com/haoshuwei/bamboo-java-demo:latest
```

iv. Use kubectl to deploy the application in the ACK cluster.

a. In the **Create tasks** section, click **Add task**. In the **Task types** dialog box, select **Script**.



b. In the **Script configuration** panel, set **Task description** and **Script location**. Use the default settings for the other parameters. Click **Save**.

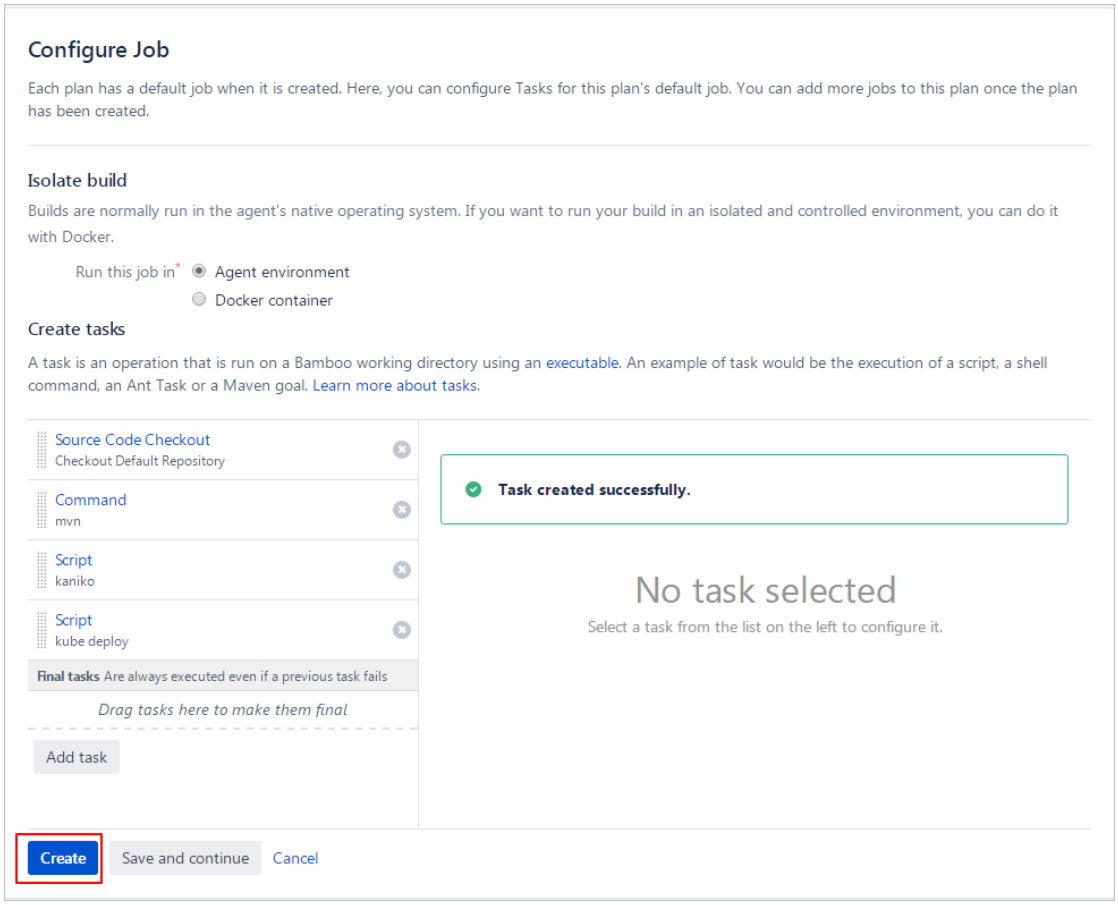


In this example, **Script location** is specified as:

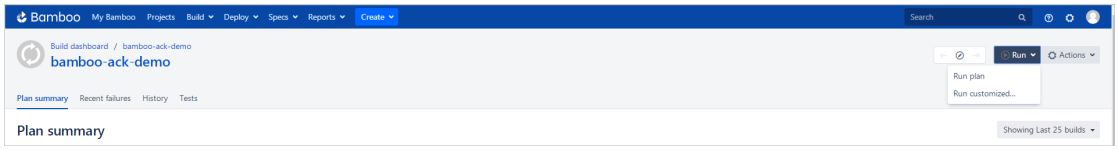
```
sed -i 's#IMAGE_URL#registry.cn-hangzhou.aliyuncs.com/haoshuwei/bamboo-java-dem
o:latest#' ./*.yaml
kubectl apply -f ./
```

3. Run the build plan.

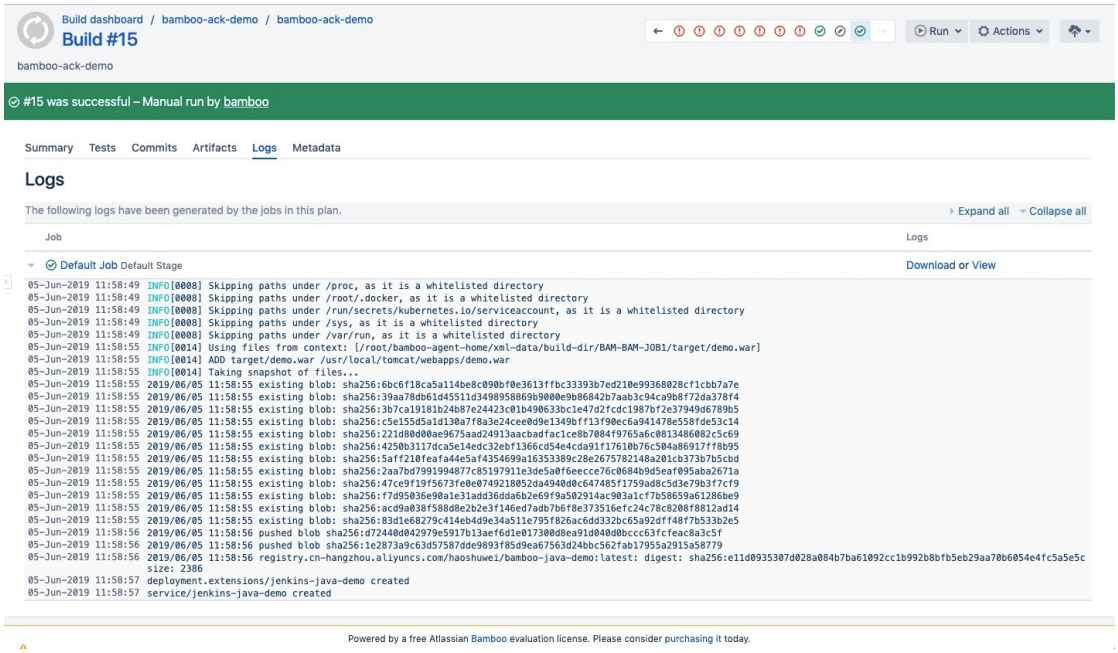
i. After you have completed the preceding steps, click **Create**. The **bamboo-ack-demo** page appears.



ii. In the upper-right corner of the bamboo-ack-demo page, choose **Run > Run plan** to run the newly created build plan.



You can also click the **Logs** tab to view the log of the build plan.



4. Access the application.

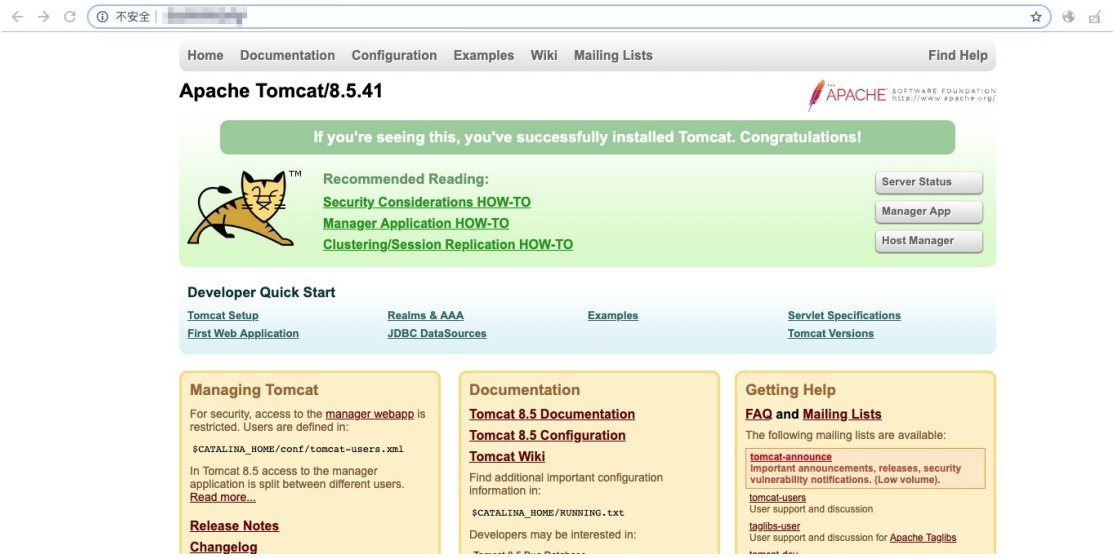
   i. Run the `kubectl -n bamboo get svc` command to query information about the Service for the deployed application.

```
[root@iZbp12i73koztp1cz75skaZ bamboo]# kubectl -n bamboo get svc
```

Expected output:

```
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP       PORT(S)
AGE
jenkins-java-demo   LoadBalancer   xx.xx.xx.xx     xx.xx.xx.xx       80:32668/TCP
39m
```

ii. You can enter *http://EXTERNAL-IP* into the address bar of your browser to access the deployed application.



References

For more information about how to build the registry.cn-hangzhou.aliyuncs.com/haoshuwei/docker-bamboo-agent:v1 image, see docker-bamboo-agent.

For more information about Bamboo, see Bamboo.

10.Migrate a self-built Kubernetes cluster to Container Service for Kubernetes

10.1. Before you start

For purposes of regulatory compliance and security, clusters of Container Service for Kubernetes (ACK) are different from self-managed Kubernetes clusters in some configurations. We recommend that you read this topic before you use ACK.

Cluster planning

ACK supports Elastic Compute Service (ECS) Bare Metal instances. You can also use GPU-accelerated ECS Bare Metal instances based on your business requirements.

- ECS Bare Metal instances

  ECS Bare Metal Instance is a compute service that is designed based on the state-of-the-art virtualization 2.0 technology developed by Alibaba Cloud. ECS Bare Metal instances combine the elasticity of virtual machines and the performance and features of physical machines. For example, ECS Bare Metal instances provide exclusive computing resources, chip-level security, confidential computing, and compatibility with multiple private clouds. ECS Bare Metal instances also support processors with heterogeneous instruction sets.

- GPU-accelerated ECS Bare Metal instances

  GPU-accelerated ECS Bare Metal instances are suitable for GPU-related computing scenarios, such as AI, deep learning, video processing, scientific computing, and data visualization.

Network planning

- The network type for ECS instances in an ACK cluster: virtual private cloud (VPC) or classic network.
  - VPC: We recommend that you use this network type.

    VPCs implement isolation at Layer 2 and provide improved security and flexibility than classic networks.

  - Classic network

    Classic networks implement isolation at Layer 3. All instances in a classic network are deployed in a shared infrastructure network.

- The network plug-in to control communication among pods in an ACK cluster: Terway or Flannel.
  - Terway: We recommend that you use this network plug-in.

    Terway is a network plug-in that is developed by Alibaba Cloud. Terway allows you to assign elastic network interfaces (ENIs) to containers. It also allows you to customize Kubernetes network policies to control intercommunication among containers, and implement bandwidth throttling on individual containers. If you do not need to use Kubernetes network policies, you can use Flannel as the network plug-in. In other cases, we recommend that you use Terway.

- Flannel

  Flannel is a simple and stable Container Network Interface (CNI) plug-in that is developed by the Kubernetes community. You can use Flannel with virtual private clouds (VPCs) of Alibaba Cloud. This enables your clusters and containers to run in a high-speed and stable network. Flannel does not support standard Kubernetes network policies.

Capacity planning

- Basic capacity: When you create an ACK cluster, you can set a basic capacity based on your business requirements and budget.
- Scalable capacity: After an ACK cluster is created, you can configure auto scaling to scale the cluster as required.
  - horizontal pod autoscaling

    ACK supports Horizontal Pod Autoscaling (HPA). After HPA is enabled, an ACK cluster can automatically increase or reduce the number of application pods.
  - cluster-autoscaler

    ACK provides cluster-autoscaler for nodes to automatically scale in and out. Regular instances, GPU-accelerated instances, and preemptible instances can be automatically added to or removed from an ACK cluster based on your requirements. This feature is applicable to instances that are deployed across multiple zones and diverse instance types, and also supports different scaling modes.

Data planning

- Database

  Relational Database Service (RDS) is a stable and reliable online database service that supports auto scaling. Based on Apsara Distributed File System and high-performance SSDs of Alibaba Cloud, RDS supports the MySQL, SQL Server, PostgreSQL, PPAS, and MariaDB TX database engines.
- Storage

  Object Storage Service (OSS) is a cloud storage service that features large capacity, high security, cost-effectiveness, and high reliability.
- Container images

  Container Registry provides a secure service for the secure maintenance and lifecycle management of application images. It allows you to build images from code repositories in China and overseas, perform security scans on images to detect potential risks, and simplify licensing on images. Container Registry provides instances of Container Registry Enterprise Edition and default instances.

Security planning

- Infrastructure security

  You can set security group rules to secure the infrastructure of an ACK cluster.
- Image security

  ACK allows you to use private images and perform security scans on images to detect potential risks.
- Kubernetes application security

  ACK supports Kubernetes network policies to control communications among applications.

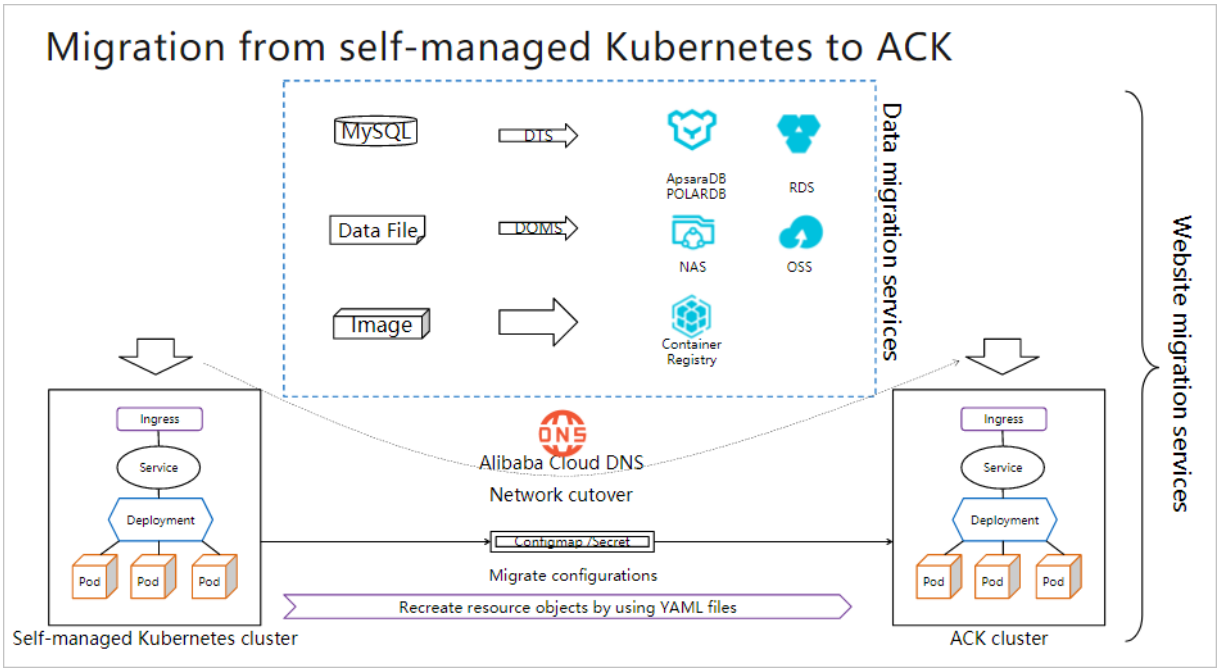
Basic O&M

- Monitoring: You can use Cloud Monitor or Application Real-Time Monitoring Service (ARMS) to monitor ACK clusters and applications. You can also set alert rules based on monitoring metrics.
  - Cloud Monitor is a service that is provided by Alibaba Cloud for you to monitor Alibaba Cloud resources. In the Cloud Monitor console, you can quickly check the status and performance of the monitored resources. Cloud Monitor allows you to monitor sites and Alibaba Cloud services. You can also customize metrics and alert rules. The Cloud Monitor console provides monitoring statistics and charts to show the status of monitored services. In addition, Cloud Monitor sends notifications of cluster exceptions based on the alert rules and monitoring metrics.
  - ARMS is an application performance management (APM) service that is provided by Alibaba Cloud. It consists of frontend monitoring, application monitoring, and Prometheus monitoring. ARMS allows you to monitor and diagnose both the frontend and the backend, including browsers, mini programs, applications, distributed applications, and containers. This improves the efficiency of operations and maintenance (O&M) on applications.
- Logs: You can use Log Service instead of custom logging solutions to collect logs from ACK clusters. For more information, see Log Service.

10.2. Overview

This topic describes how to migrate a self-managed Kubernetes cluster to Container Service for Kubernetes (ACK) without service disruption.

Migration scheme



Procedure

1. Create and configure an ACK cluster.

   O&M engineers create an ACK cluster and configure cluster resources. This step makes it easy for developers to migrate applications to Kubernetes. For more information, see Create an ACK dedicated cluster.

The following parameters are required when you create the ACK cluster:

- Cluster type: Select one of the following cluster types when you create the ACK cluster.
  - Standard dedicated Kubernetes cluster.

    You need to create and manage the master and worker nodes of the ACK cluster. You have full control over the ACK cluster. You are charged for the resources that are used by the master and worker nodes.
  - Standard managed Kubernetes cluster.

    You need to create and manage only the worker nodes of the ACK cluster. ACK creates and manages the master nodes. You are charged for only the resources that are used by the worker nodes.
- **Operating system**: Select an operating system based on your requirements.
  - By default, the CentOS 7.6 or Aliyun Linux 2.1903 operating system is selected. We recommend that you keep the default settings.
  - If the default system kernel cannot meet your requirements, you can use a custom image. For more information, see Use a custom image to create an ACK cluster.
- **VPC**: Select a virtual private cloud (VPC) and vSwitches for the ACK cluster.
- **SNAT**: Configure Source Network Address Translation (SNAT) entries for the VPC.
- **Public access**: Expose the API server of the ACK cluster with an elastic IP address (EIP).
- **Cloud Monitor agent**: Install the Cloud Monitor agent on the Elastic Compute Service (ECS) instances of the ACK cluster.
- **Log Service**: Install and configure the Log Service agent in the ACK cluster.

2. Migrate data.
   - Migrate databases.
     a. Create a Relational Data Service (RDS) instance.
     b. Configure an RDS whitelist to allow only specified IP addresses to access the RDS instance.
     c. Configure a PrivateZone.

        Use the PrivateZone to resolve the domain name of the database to the IP address of the RDS instance. This saves you the need to modify the database configuration on applications.
     d. Migrate the data in the MySQL database of the self-managed Kubernetes cluster.

        Use Data Transmission Service (DTS) to migrate the data from the MySQL database to the ApsaraDB for RDS database in full, incremental, or two-way synchronization mode. For more information, see Migrate data from a self-managed MySQL database to an ApsaraDB RDS for MySQL instance.
   - Migrate data.
     a. Activate Object Storage Service (OSS).
     b. Create an OSS bucket.

c. Migrate the data that is stored in the self-managed Kubernetes cluster.

Use the ossimport tool to migrate the data in batches from an on-premises server or a third-party cloud storage service to OSS. For example, you can choose a third-party cloud storage service such as Amazon S3, Microsoft Azure, or Tencent Cloud Object Storage. For more information, see Architectures and configurations.

- Migrate images.
  a. Create a Container Registry repository.
  b. Set the credential that is used to access the created Container Registry repository.
  c. Migrate the images of the self-managed Kubernetes cluster.

  You can use the image-syncer tool to migrate the images of the self-managed Kubernetes cluster to the Container Registry repository. For more information, see Use image-syncer to migrate container images.

3. Migrate application configurations.

   O&M engineers or developers use the Velero tool to migrate the cluster or application configurations. For more information, see Migrate applications to an ACK cluster.

   i. Set up the migration environment.
      a. Install the Velero client.
      b. Create an OSS bucket.
      c. Create a Resource Access Management (RAM) user and generate an AccessKey pair.
      d. Deploy the Velero server.
   ii. Back up applications in the self-managed Kubernetes cluster.
      - Back up applications without persistent volumes (PVs).
      - Back up applications with PVs.
   iii. Restore applications in the ACK cluster.
      a. Create a StorageClass.
      b. Restore applications.
   iv. Update the application configurations.
      - Update image registries.
      - Optimize the method to expose the services.
      - Modify the configurations to mount disks.
   v. Debug and start the applications.

4. Perform regression tests.

   Test engineers perform regression tests on the ACK cluster without interrupting the online business.

   i. Configure the domain names for regression tests.
   ii. Test applications.
   iii. Check the logged application events.
   iv. Check the monitoring metrics of the applications.

5. Switch all production traffic to ACK.

O&M engineers modify the Domain Name System (DNS) configuration to switch traffic to the ACK cluster.

i. Use the DNS service: Modify the DNS configuration to switch the traffic.

ii. Upgrade the configurations or code of the client to switch the traffic.

6. Bring the self-managed Kubernetes cluster offline.

   O&M engineers check whether the ACK cluster can be accessed as expected. Then, O&M engineers bring the self-managed Kubernetes cluster offline.

   i. Check whether the ACK cluster can receive and send traffic as expected.

   ii. Bring the self-managed Kubernetes cluster offline.

   iii. Clear the backup files stored in the OSS bucket that is created when you migrate applications.

10.3. Use a custom image to create an ACK cluster

If you migrate a user-created Kubernetes cluster to an ACK cluster, we recommend that you use the default system image and default system services to create the ACK cluster. However, you can also use a custom image to create ACK clusters based on your business requirements. This topic describes how to use a custom image to create an ACK cluster.

Context

If you migrate a user-created Kubernetes cluster to an ACK cluster, we recommend that you use the default system image for CentOS 7.6 or Alibaba Cloud Linux 2.1903 and default system services. The default system services include the operating system kernel, domain name system (DNS) service, and YUM repositories. If you want to use a custom image to create an ACK cluster, use the ack-image-builder tool to create a custom image.

Use ack-image-builder to create a custom image

The ack-image-builder tool is developed based on open source tool HashiCorp Packer. The ack-image-builder tool provides a default template and a verification script for you to create custom images.

By using ack-image-builder, you can reduce errors caused by manual operations. The ack-image-builder tool also records image changes to facilitate troubleshooting. To use the ack-image-builder tool to create a custom image for an ACK cluster, perform the following steps:

1. Install Packer.

   Download Packer from its official website. Make sure that the version is compatible with your operating system. Then, install and verify Packer based on its installation documentation.

   Run the packer version command. The following command output indicates that Packer is installed.

   ```
   packer version
   ```

   ```
   Packer v1.4.1
   ```

2. Create a template in Packer.

   Before you use Packer to create a custom image, create a template in JSON format. In the template, you must specify the image builder and provisioner that are used to create a custom image. In this

example, Alicloud Image Builder and the shell provisioner are used.

```
{
  "variables": {
    "region": "cn-hangzhou",
    "image_name": "test_image{{timestamp}}",
    "source_image": "centos_7_06_64_20G_alibase_20190711.vhd",
    "instance_type": "ecs.n1.large",
    "access_key": "{{env `ALICLOUD_ACCESS_KEY`}}",
    "secret_key": "{{env `ALICLOUD_SECRET_KEY`}}"
  },
  "builders": [
    {
      "type": "alicloud-ecs",
      "access_key": "{{user `access_key`}}",
      "secret_key": "{{user `secret_key`}}",
      "region": "{{user `region`}}",
      "image_name": "{{user `image_name`}}",
      "source_image": "{{user `source_image`}}",
      "ssh_username": "root",
      "instance_type": "{{user `instance_type`}}",
      "io_optimized": "true"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "scripts": [
        "config/default.sh",
        "scripts/updateDNS.sh",
        "scripts/reboot.sh",
        "scripts/verify.sh"
      ],
      "expect_disconnect": true
    }
  ]
}
```

Parameter	Description
access_key	The AccessKey ID of your account.
secret_key	The AccessKey secret of your account.
region	The region of the temporary instance used to create the custom image.
image_name	The name of the custom image.
source_image	The name of the source image that is used to create the custom image. You can obtain the name from the public image list of Alibaba Cloud.

Parameter	Description
instance_type	The type of the temporary instance used to create the custom image.
provisioners	The provisioner used to create the custom image.

3. Create a Resource Access Management (RAM) user and generate an AccessKey pair.

   A wide range of permissions is required to create a custom image. We recommend that you create a RAM user and use a RAM policy to grant the permissions required by Packer to the RAM user. Then, create an AccessKey pair for the RAM user. For more information, see Obtain an AccessKey pair.

4. Add the AccessKey pair information to the template and create a custom image.

   i. Run the following commands to add the AccessKey pair information:

   ```
   export ALICLOUD_ACCESS_KEY=XXXXXX
   export ALICLOUD_SECRET_KEY=XXXXXX
   ```

   ii. Run the following commands to create a custom image:

   ```
   packer build alicloud.json
   ```

```
alicloud-ecs output will be in this color.
==> alicloud-ecs: Prevalidating source region and copied regions...
==> alicloud-ecs: Prevalidating image name...
    alicloud-ecs: Found image ID: centos_7_06_64_20G_alibase_20190711.vhd
==> alicloud-ecs: Creating temporary keypair: xxxxxx
==> alicloud-ecs: Creating vpc...
    alicloud-ecs: Created vpc: xxxxxx
==> alicloud-ecs: Creating vswitch...
    alicloud-ecs: Created vswitch: xxxxxx
==> alicloud-ecs: Creating security group...
    alicloud-ecs: Created security group: xxxxxx
==> alicloud-ecs: Creating instance...
    alicloud-ecs: Created instance: xxxxxx
==> alicloud-ecs: Allocating eip...
    alicloud-ecs: Allocated eip: xxxxxx
    alicloud-ecs: Attach keypair xxxxxx to instance: xxxxxx
==> alicloud-ecs: Starting instance: xxxxxx
==> alicloud-ecs: Using ssh communicator to connect: 47.111.127.54
==> alicloud-ecs: Waiting for SSH to become available...
==> alicloud-ecs: Connected to SSH!
==> alicloud-ecs: Provisioning with shell script: scripts/verify.sh
    alicloud-ecs: [20190726 11:04:10]: Check if kernel version >= 3.10.  Verify Pas
sed!
    alicloud-ecs: [20190726 11:04:10]: Check if systemd version >= 219.  Verify Pas
sed!
    alicloud-ecs: [20190726 11:04:10]: Check if sshd is running and listen on port
22.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if cloud-init is installed.  Verify Pa
ssed!
    alicloud-ecs: [20190726 11:04:10]: Check if wget is installed.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if curl is installed.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubeadm is cleaned up.  Verify Pass
ed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubelet is cleaned up.  Verify Pass
ed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubectl is cleaned up.  Verify Pass
ed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubernetes-cni is cleaned up.  Veri
fy Passed!
==> alicloud-ecs: Stopping instance: xxxxxx
==> alicloud-ecs: Waiting instance stopped: xxxxxx
==> alicloud-ecs: Creating image: test_image1564110199
    alicloud-ecs: Detach keypair xxxxxx from instance: xxxxxxx
==> alicloud-ecs: Cleaning up 'EIP'
==> alicloud-ecs: Cleaning up 'instance'
==> alicloud-ecs: Cleaning up 'security group'
==> alicloud-ecs: Cleaning up 'vSwitch'
==> alicloud-ecs: Cleaning up 'VPC'
==> alicloud-ecs: Deleting temporary keypair...
Build 'alicloud-ecs' finished.
==> Builds finished. The artifacts of successful builds are:
--> alicloud-ecs: Alicloud images were created:
cn-hangzhou: m-bp1aifbnupnaktj00q7s
```

The scripts/verify.sh script is used to verify the custom image.

Use a custom operating system kernel

ACK requires a Linux operating system with the kernel of `V3.10` or later. We recommend that you update only the RPM packages to be customized. You must set boot parameters for the kernel.

You can use the following code:

```
cat scripts/updateOSKernel.sh
#! /bin/bash
VERSION_KERNEL="3.10.0-1062.4.3.el7"
yum  localinstall -y  http://xxx.xxx.xxx.xxx/kernel-${VERSION_KERNEL}.x86_64.rpm   http://x
xx.xxx.xxx.xxx/kernel-devel-${VERSION_KERNEL}.x86_64.rpm   http://xxx.xxx.xxx.xxx/kernel-he
aders-${VERSION_KERNEL}.x86_64.rpm
grub_num=$(cat /etc/grub2.cfg |awk -F\' '$1=="menuentry " {print i   " : " $2}' |grep $VERS
ION_KERNEL |awk -F ':' '{print $1}')
grub2-set-default $grub_num
```

Note We recommend that you do not run commands that update all RPM packages, such as the `yum update -y` command.

Customize the operating system kernel

When you customize kernel parameters, do not overwrite the following parameters:

```
["vm.max_map_count"]="262144"
["kernel.softlockup_panic"]="1"
["kernel.softlockup_all_cpu_backtrace"]="1"
["net.core.somaxconn"]="32768"
["net.core.rmem_max"]="16777216"
["net.core.wmem_max"]="16777216"
["net.ipv4.tcp_wmem"]="4096 12582912 16777216"
["net.ipv4.tcp_rmem"]="4096 12582912 16777216"
["net.ipv4.tcp_max_syn_backlog"]="8096"
["net.ipv4.tcp_slow_start_after_idle"]="0"
["net.core.netdev_max_backlog"]="16384"
["fs.file-max"]="2097152"
["fs.inotify.max_user_instances"]="8192"
["fs.inotify.max_user_watches"]="524288"
["fs.inotify.max_queued_events"]="16384"
["net.ipv4.ip_forward"]="1"
["net.bridge.bridge-nf-call-iptables"]="1"
["fs.may_detach_mounts"]="1"
["net.ipv4.conf.default.rp_filter"]="0"
["net.ipv4.tcp_tw_reuse"]="0"
["net.ipv4.tcp_tw_recycle"]="0"
```

> **Note** If you need to modify some of the preceding parameters, submit a ticket to request Alibaba Cloud engineers to analyze the effects. After you are authorized to modify the preceding parameters, you can go to the page for creating or scaling out a cluster, click **Show Advanced Options**, and then enter your script in the **User Data** field.

Use a custom DNS service

If you use a custom DNS service, pay attention to the following notes:

- Add Alibaba Cloud name servers to the upstream name servers of the custom DNS service.

```
cat /etc/resolv.conf
options timeout:2 attempts:3 rotate single-request-reopen
; generated by /usr/sbin/dhclient-script
nameserver 100.XX.XX.136
nameserver 100.XX.XX.138
```

- Lock the */etc/resolve.conf* file after you modify it. Otherwise, cloud-init restores the file to default settings after ECS instances restart. You can use the following code:

```
cat scripts/updateDNS.sh
#! /bin/bash
# unlock DNS file in case it was locked
chattr -i /etc/resolv.conf
# Using your custom nameserver to replace xxx.xxx.xxx.xxx
echo -e "nameserver xxx.xxx.xxx.xxx\nnameserver xxx.xxx.xxx.xxx" > /etc/resolv.conf
# Keep resolv locked to prevent overwriting by cloudinit/NetworkManager
chattr  i /etc/resolv.conf
```

- Ensure adequate performance of the custom DNS service.

  Make sure that the performance of the custom DNS service can meet the requirements if your cluster contains a large number of nodes.

Use a custom YUM repository

If you use a custom YUM repository, pay attention to the following notes:

- Do not update all RPM packages.

  Update only the RPM packages to be installed. Do not run the `yum update -y` command to update all RPM packages.

- Ensure adequate performance of the YUM repository.

  If you want to add a large number of worker nodes to the cluster at a time and update RPM packages based on the YUM repository, make sure that the performance of the YUM repository can meet your business requirements. You can use the following code:

```
cat scripts/add-yum-repo.sh
#! /bin/bash
cat << EOF > /etc/yum.repos.d/my.repo
[base]
name=CentOS-\$releasever
enabled=1
failovermethod=priority
baseurl=http://mirrors.cloud.aliyuncs.com/centos/\$releasever/os/\$basearch/
gpgcheck=1
gpgkey=http://mirrors.cloud.aliyuncs.com/centos/RPM-GPG-KEY-CentOS-7
EOF
```

Preload the container images of DaemonSet components

If you want to add more than 1,000 worker nodes to the cluster at a time, we recommend that you preload the container images of DaemonSet components before you create the custom image. This reduces the workload of pulling these container images when nodes start and improves the efficiency of cluster scale-outs.

1. Compress the required container images to TAR packages and store them in the custom image.

   Assume that the ACK cluster uses the Terway network plug-in and the Container Storage Interface (CSI) storage plug-in, and resides in the China (Hangzhou) region. You can use the following script to preload the container images of the preceding plug-ins:

   ```
   cat scripts/prepare-images.sh
   #! /bin/bash
   set -x -e
   EXPORT_PATH=/preheated
   # Install Docker.
   yum install -y docker
   systemctl start docker
   # Pull and save images.
   images=(
   registry-vpc.cn-hangzhou.aliyuncs.com/acs/csi-plugin:v1.14.5.60-5318afe-aliyun
   registry-vpc.cn-hangzhou.aliyuncs.com/acs/terway:v1.0.10.78-g97729ee-aliyun
   )
   mkdir -p ${EXPORT_PATH}
   for image in "${images[@]}"; do
       echo "preheating ${image}"
       docker pull ${image}
       docker save -o ${EXPORT_PATH}/$(echo ${image}| md5sum | cut -f1 -d" ").tar ${image}
   done
   # Uninstall Docker.
   yum erase -y docker
   rm -rf /var/lib/docker
   ```

2. Log on to the ACK console. Go to the cluster creation page, click **Show Advanced Options**, and then enter the following script in the **User Data** field:

   ```
   ls /preheated/ | xargs -n 1 -i docker load -i /preheated/{}
   rm -rf /preheated
   ```

Edit the configuration file of the custom image

Add the following configurations about `provisioners` to the `alicloud.json` file for creating the custom image:

```
"provisioners": [
  {
    "type": "shell",
    "scripts": [
      "config/default.sh",
      "scripts/updateOSKernel.sh",
      "scripts/updateDNS.sh",
      "scripts/add-yum-repo.sh",
      "scripts/prepare-images.sh",
      "scripts/reboot.sh",
      "scripts/verify.sh"
    ],
    "expect_disconnect": true
  }
]
```

Note The `config/default.sh`, `scripts/reboot.sh`, and `scripts/verify.sh` scripts are default scripts that you must run. Others are custom scripts.

The `config/default.sh` script sets the time zone and disables swap partitions.

The `scripts/verify.sh` script checks whether the custom image meets the requirements of the desired ACK cluster.

After you edit the configuration file of the custom image, you can create the custom image and use it to create or scale out an ACK cluster.

Create an ACK cluster

We recommend that you first create a dedicated Kubernetes cluster that contains no worker nodes or a managed Kubernetes cluster that contains two worker nodes, add worker nodes that use a custom image to the cluster, and verify the result. This saves time and decreases the probability of errors.

1. Use the default system image to create a dedicated Kubernetes cluster that contains three or five master nodes and no worker nodes. For more information, see Create an ACK dedicated cluster.

   Note If you create a managed Kubernetes cluster, select at least two worker nodes. For more information, see Create an ACK managed cluster.

2. Add worker nodes that use a custom image to the cluster. For more information, see Increase the number of nodes in an ACK cluster.

   If you want to run an initialization script after you add the worker nodes to the cluster, you can configure the user data for Elastic Compute Service (ECS) instances.

   Note To use **custom images** and configure the **user data**, .

10.4. Migrate source servers to Container Registry

Server Migration Center (SMC) allows you to migrate servers to Container Registry. You can use SMC to migrate containerized applications to Container Registry at low costs. Containerized applications achieve automatic management and distribution of compute resources and ensure the fast and secure deployment of applications. This improves resource usage and reduces compute costs. This topic describes how to migrate a server to Container Registry.

Prerequisites

- The server is not based on a Windows operating system.
- Container Registry is activated and a container image repository is created. For more information, see Create a repository and build images.
- A Resource Access Management (RAM) role is created for the intermediate instance that is generated by SMC for migration. The following parameters are used to configure the RAM role. For more information, see Create a RAM role for a trusted Alibaba Cloud service.
  - Trusted Entity Type: Select **Alibaba Cloud Service**.
  - Role Type: Select **Normal Service Role**.
  - Trusted Service: Select **Elastic Compute Service**.
- A custom policy is created for the RAM role of the intermediate instance. The policy grants the minimum permissions that are required to migrate a server to Container Registry. The following example shows a sample policy. This policy is attached to the RAM role. For more information, see Create a custom policy and Grant permissions to a RAM role.

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cr:GetAuthorizationToken",
                "cr:PushRepository",
                "cr:PullRepository",
                "cr:CreateRepository"
            ],
            "Resource": [
                "*"
            ]
        }
    ],
    "Version": "1"
}
```

- The information about the source server is imported to the SMC console. For more information, see Step 1: Import the information of a migration source.

> **Notice**
> - The SMC client V2.3.0 and later support server migration to Container Registry. You can use these SMC clients to import the information about the source server. Download the latest version of the SMC client.
> - Ensure that the client-side program is running during migration. If data transfer is interrupted, you can restart the client and the migration task to resume migration.

Context

- For information about Docker container images, see Terms.
- During migration, SMC creates an intermediate instance. The intermediate instance incurs a small fee. For more information, see Pay-as-you-go.
- SMC releases the intermediate instance after the migration task enters the **Finished** or **Expired** state, or when the task is deleted.

(Optional) Step 1: Exclude dynamic data directories from migration

To ensure stable migration, we recommend that you exclude dynamic data directories, such as data directories of large databases. Then, you can stop the services on the source server and start the migration task. Skip this step if you do not need to exclude dynamic data directories.

To exclude dynamic data directories, perform the following steps. You do not need to stop services that are running on the source server. To enable the throttling environment, perform the following steps:

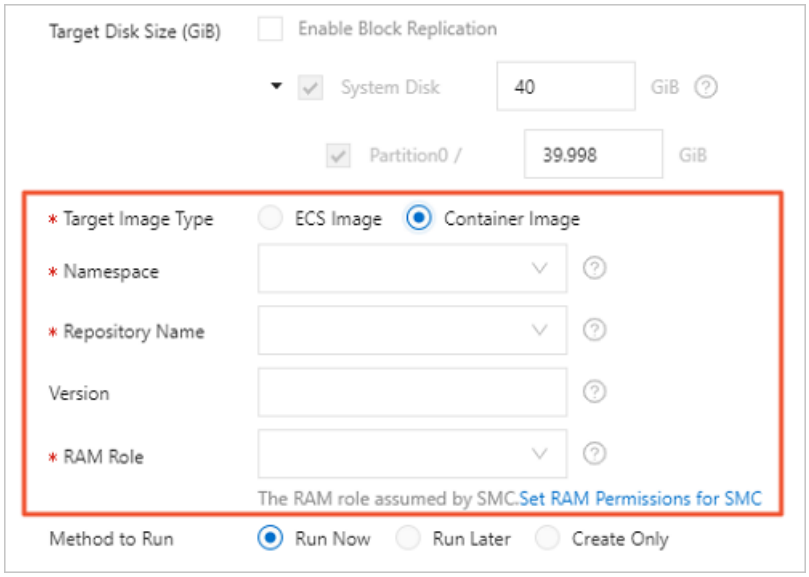
1. Log on to the source server.
2. Configure the SMC client and exclude dynamic data directories.

   For more information, see Exclude files or directories from migration.

Step 2: Create and start a migration task

You can perform the following steps to exclude dynamic data directories. You do not need to stop your services that are running on the source server. To enable the throttling environment, perform the following steps:

1. Log on to the SMC console.
2. In the left-side navigation pane, click **Migration Sources**.
3. Find the source server from which you want to migrate data.
4. Click **Create Migration Task** in the **Actions** column.
5. On the **Create Migration Task** page, set the container image parameters.



Set the following parameters to configure the container image: For more information about other parameters, see Step 2: Create and start a migration task.

- **Target Image Type**: the type of the destination image. Select **Container Image**.
- **Namespace**: the namespace of the destination image.
- **Repository Name**: the name of the repository where the destination image is stored.
- **Version**: the version of the destination image.
- **RAM Role**: the RAM role that is attached to the intermediate instance.

The migration task immediately runs after it is created. Possible results are:

- If the migration task enters the Finished state, the task is completed and a container image is generated.
- If the migration task enters the **InError** state, the task fails. You can check the logs to troubleshoot the failure. Then, restart the migration task. For information about common errors and solutions, see SMC FAQ.

Step 3: Verify the container image

After you migrate the source server to Container Registry, a container image is created. Perform the following steps to verify the container image. A container image on which NGINX is deployed is used as an example.

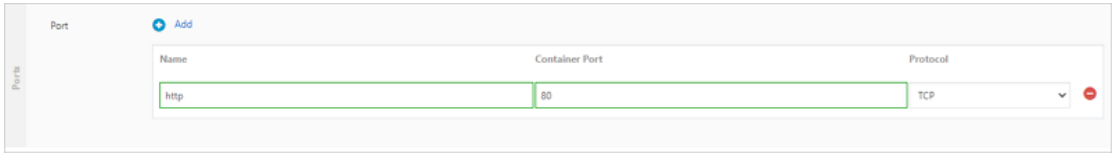
1. Create a Kubernetes cluster in the Container Service for Kubernetes (ACK) console. For more information, see Create an ACK managed cluster.
2. Log on to the ACK console.
3. In the left-side navigation pane, click **Clusters** to view the created Kubernetes cluster.
4. In the **Actions** column, click **Applications**.
5. On the **Deployments** tab, click **Create from Image** to create a stateless application.

   Set the following parameters. For information about other parameters and operations, see Create a stateless application by using a Deployment.

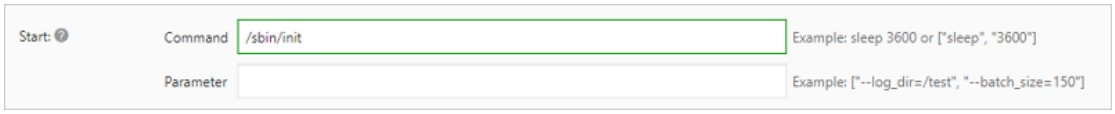
   - On the **Basic Information** tab, set the following parameters:
     - **Name**: the name of the application. Set the value to nginx.

- **Replicas**: the number of application replicas. Set the value to 1.
- **Type**: the type of the application. Set the value to **Deployments**.

- In the **Container** step, set the following parameters:
  - **Image Name**: Click **Select Image** to select the container image that is generated after the migration is complete. If the container image repository and the Kubernetes cluster are deployed in the same region, you can use the Virtual Private Cloud (VPC) endpoint of the container image to pull the image.
  - **Image Version**: Click **Select Image Version** to select the version of the container image that is generated after the migration is complete.
  - **Set Image Pull Secret**: This parameter is required if the container image is a private image. You can also use a plug-in to pull the image. This method does not require a Secret. For more information, see Use the aliyun-acr-credential-helper component to pull images without a password.
  - **Port**: Add port 80.

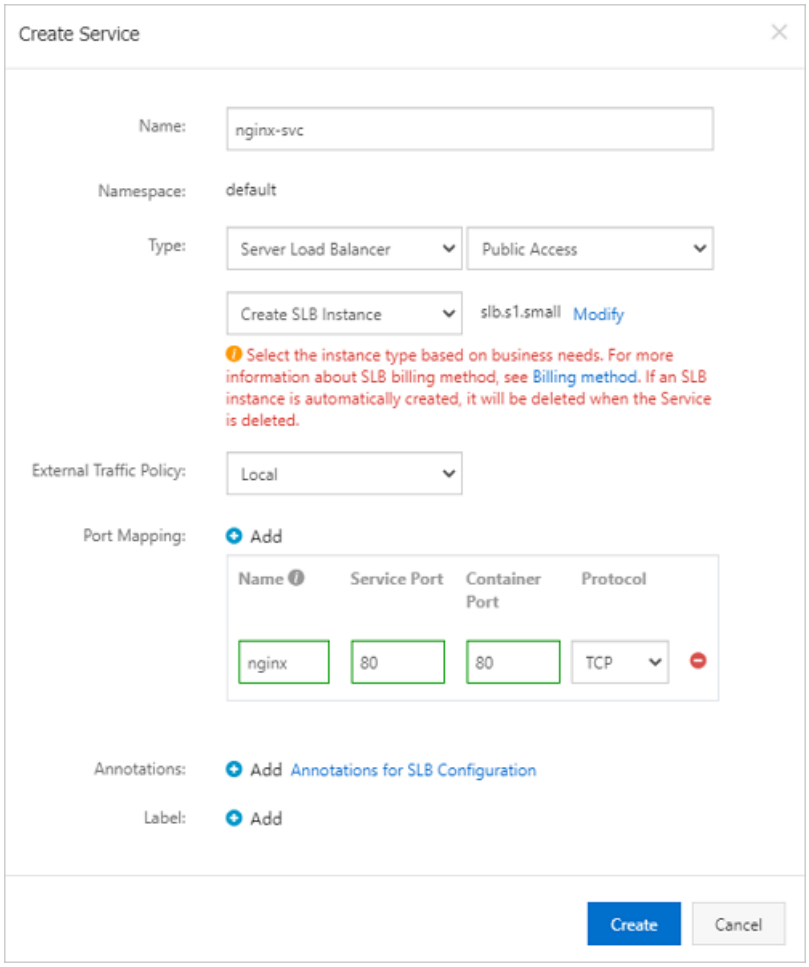


  - **Start**: Enter the **/sbin/init** command.



- In the **Advanced** step, create a Service to access the application.

Use the example values in the following figure when you set the required parameters.



6. After the Service is created, click **Details** to view the application status.
7. In the left-side navigation pane, click **Services and Ingresses > Services**. On the Services page, view the **External Endpoint** of the Service.
8. You can use a browser to access the external endpoint.

10.5. Migrate container images

10.5.1. Use image-syncer to migrate container images

Image migration and synchronization between image repositories are required to migrate applications from a self-managed Kubernetes cluster to a Container Service for Kubernetes (ACK) cluster. You can use image-syncer to migrate and synchronize multiple images from self-managed image repositories to Alibaba Cloud Container Registry (ACR) on the fly at a time. This topic describes how to use image-syncer to migrate container images.

Context

Compared with the Kubernetes clusters created and maintained by other cloud providers, ACK is superior in terms of service costs, maintenance expenses, ease-of-use, and long-term stability. A growing number of cloud providers want to migrate their Kubernetes workloads to ACK. If the number of images is small, you can run the **docker pull** and **docker push** commands to migrate the images. If you run the commands to migrate more than a hundred images or an image repository that stores TB-level data, the migration process takes a long time and may cause data loss. In this case, the image synchronization capability is required for migrating images between image repositories. The open source tool image-syncer developed by Alibaba Cloud provides this capability and has helped many cloud service providers migrate images. The maximum image repository capacity is larger than 3 TB. The server that runs image-syncer can make full use of the server bandwidth, and no requirement exists for the disk capacity of the server.

image-syncer overview

Image migration and synchronization between image repositories are required to migrate applications from a self-managed Kubernetes cluster to an ACK cluster. The traditional image synchronization method uses a script that contains the **docker pull** or **docker push** command and has the following limits:

- You must remove existing images from the disks of the server where the destination image repository resides and store the migrated images on these disks. Therefore, this method does not apply to large-scale image migration.
- The Docker daemon is required. The Docker daemon limits the number of images that can be pulled or pushed concurrently. As a result, you cannot perform high-concurrency image synchronization.
- HTTP API operations are required to implement some features. You cannot synchronize images by using only the Docker CLI. As a result, you must write a complex synchronization script.

image-syncer is an easy-to-use tool for migrating or synchronizing a large number of images at a time. By using image-syncer, you can synchronize Docker images from or to almost all major image repository services based on Docker Registry V2, such as ACR, Docker Hub, Quay.io, and Harbor. This tool has been used to migrate TB-level images in production environments. For more information, see image-syncer.

Features

image-syncer has the following features:

- Synchronizes images from multiple source image repositories to multiple destination image repositories.
- Supports Docker image repository services based on Docker Registry V2.

  For example, image-syncer supports Docker Hub, Quay.io, Alibaba Cloud Container Registry, and Harbor.
- Synchronizes images by using only the memory and network resources. Images are not stored on the disks of the server where the destination image repository resides. This improves the synchronization efficiency.
- Supports incremental synchronization.

image-syncer uses a file to record the blob information about synchronized images. Therefore, images that have been synchronized are not synchronized again.

- Supports concurrent synchronization.

  You can modify the number of images that can be concurrently pulled or pushed in the configuration file.

- Automatically retries failed synchronization tasks to resolve most image synchronization issues caused by network jitters.
- Programs such as the Docker daemon are not required.

By using the image-syncer tool, you can migrate, copy, and perform incremental synchronization of images from an image repository. Make sure that image-syncer can communicate with the source and destination repositories. image-syncer has no requirements on hardware resources. However, the number of images that are concurrently synchronized must be equal to the number of network connections on image-syncer. The memory consumed by image-syncer is less than or equal to the product of the number of images that are concurrently synchronized and the size of the largest image layer. Therefore, the memory of the server that runs image-syncer may be exhausted only if the size of the largest image layer and the number of images that are concurrently synchronized are too large. In addition, image-syncer provides a retransmission mechanism to avoid occasional failures during synchronization. image-syncer counts the number of images that fail to be synchronized when synchronization ends and provides detailed logs to help you locate issues.

Preparations

To use image-syncer, prepare a configuration file. The following code block shows an example of the configuration file:

```
{
    "auth": {                   // The authentication information field. Each object contai
ns the username and password that are required to access a registry.
                                // In most cases, image-syncer must have permissions to pul
l images from and access tags in the source registry.
                                // image-syncer must have permissions to push images to and
create repositories in the destination registry. If no authentication information is provid
ed for a registry, image-syncer accesses the registry in anonymous mode.
        "quay.io": {            // The URL of the registry, which must be the same as that
of the registry in image URLs.
            "username": "xxx",              // Optional. The username.
            "password": "xxxxxxxxx",        // Optional. The password.
            "insecure": true                // Optional. Specifies whether the repository
is accessed through HTTP. Default value: false. Only image-syncer of V1.0.1 and later suppo
rt this parameter.
        },
        "registry.cn-beijing.aliyuncs.com": {
            "username": "xxx",
            "password": "xxxxxxxxx"
        },
        "registry.hub.docker.com": {
            "username": "xxx",
            "password": "xxxxxxxxxx"
        }
    },
    "images": {
        // The field that describes image synchronization rules. Each rule is a key-value p
```

```
        // The field that describes image synchronization rules. Each rule is a key value p
air. The key specifies the URL of the source repository and the value specifies the URL of
the destination repository.
        // You cannot synchronize an entire namespace or registry based on one rule. You ca
n synchronize only one repository based on one rule.
        // The URLs of the source and destination repositories are in the format of registr
y/namespace/repository:tag, which is similar to the image URL format used in the docker pul
l or docker push command.
        // The URL of the source repository must contain registry/namespace/repository. If
the URL of the destination repository is not an empty string, it must also contain registry
/namespace/repository.
        // The URL of the source repository cannot be an empty string. To synchronize image
s from a source repository to multiple destination repositories, you must configure multipl
e rules.
        // The name and tags of the destination repository can be different from those of t
he source repository. In this case, the image synchronization rule works in the same way as
the combination of the docker pull, docker tag, and docker push commands.
        "quay.io/coreos/kube-rbac-proxy": "quay.io/ruohe/kube-rbac-proxy",
        "xxxx":"xxxxx",
        "xxx/xxx/xx:tag1,tag2,tag3":"xxx/xxx/xx"
        // If the URL of the source repository does not contain tags, all images in the sou
rce repository are synchronized to the destination repository with the original tags. In th
is case, the URL of the destination repository cannot contain tags.
        // If the URL of source repository contains only one tag, only images that has this
tag in the source repository are synchronized to the destination repository. If the URL of
the destination repository does not contain a tag, synchronized images keep the original ta
g.
        // If the URL of the source repository contains multiple tags that are separated wi
th commas (,), such as "a/b/c:1,2,3", the URL of the destination repository cannot contain
tags. Synchronized images keep the original tags.
        // If the URL of the destination repository is an empty string, images are synchron
ized to a repository that has the same name and tags in the default namespace of the defaul
t registry. The default registry and namespace can be set through command parameters or env
ironment variables.
    }
}
```

Examples

- Synchronize images from a self-managed Harbor instance to Container Registry Default Instance Edition
- Synchronize images from a self-managed Harbor project to Container Registry Enterprise Edition

10.5.2. Synchronize images from a self-managed Harbor instance to Container Registry Default Instance Edition

Container Registry is a service that is provided by Alibaba Cloud for managing container images. You can use Container Registry to manage the lifecycle of container images in 20 regions around the world. Container Registry is integrated with other Alibaba Cloud services such as Container Service for Kubernetes (ACK) to provide an all-in-one solution for managing cloud-native applications. This topic describes how to use image-syncer to synchronize images from a self-managed Harbor instance to an instance of Container Registry Default Instance Edition.

Prerequisites

The Container Registry service is activated.

You can log on to the to activate Container Registry.

Create a namespace

A namespace allows you to manage a collection of repositories. You can manage permissions on repositories and repository attributes. You can enable **Automatically Create Repository** for a namespace. When you run the **docker push** command to push images to a repository that does not exist in the namespace, the repository is automatically created.

> **Note** The repository that is created by using the **docker push** command can be public or private based on the default repository type of the namespace.

1. Log on to the .
2. In the left-side navigation pane, choose **Default Instance > Namespaces**.
3. On the **Namespaces** page, click **Create Namespace** in the upper-right corner.
4. In the **Create Namespace** dialog box, set parameters for the namespace and click **Confirm**.

After the namespace is created, you can find it on the **Namespaces** page. You can also manage namespaces on the Namespaces page. For more information, see Manage namespaces.

Grant permissions to a RAM user

Before you perform operations as a Resource Access Management (RAM) user, create a RAM user and grant permissions to the RAM user. Skip this step if you use an Alibaba Cloud account to perform subsequent operations.

1. Create a RAM user. For more information, see Create a RAM user.
2. Grant permissions to the RAM user. For more information, see Create a custom RAM policy.

   In this example, the RAM user requires only permissions to create and update image repositories in the image-syncer namespace. The following policy grants the RAM user the minimum required permissions:

```
{
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cr:CreateRepository",
                "cr:UpdateRepository",
                "cr:PushRepository",
                "cr:PullRepository"
            ],
            "Resource": [
                "acs:cr:*:*:repository/image-syncer/*"
            ]
        }
    ],
    "Version": "1"
}
```

Create a credential

Before you pull private images or upload images, you must run the `docker login` command to log on to the instance with a credential. Perform the following steps to create a credential:

1. In the left-side navigation pane, choose **Default Instance > Access Credential**.
2. On the **Access Credential** page, click **Set Password**.
3. In the **Set Password** dialog box, set **Password** and **Confirm Password** and click **OK**.

You can call an API operation to obtain a temporary token that you can use to access a Container Registry instance. For more information, see GetAuthorizationToken.

Configure image-syncer

In this example, images in the *library/nginx* repository of a self-managed Harbor instance are synchronized to the image-syncer namespace on an instance of Container Registry Default Instance Edition in the China (Beijing) region. The name of the source repository, which is nginx, is used as the name of the destination repository. The following example shows how to configure image-syncer:

```
{
    "auth": {
        "harbor.myk8s.paas.com:32080": {
            "username": "admin",
            "password": "xxxxxxxxx",
            "insecure": true
        },
        "registry.cn-beijing.aliyuncs.com": {
            "username": "acr_pusher@1938562138124787",
            "password": "xxxxxxxx"
        }
    },
    "images": {
        "harbor.myk8s.paas.com:32080/library/nginx": ""
    }
}
```

- `harbor.myk8s.paas.com:32080`: the endpoint of the self-managed Harbor instance. You must replace the value with an actual endpoint.
  - `username`: the username of the self-managed Harbor instance. The value is admin in this example.
  - `password`: the password of the self-managed Harbor instance.
  - `insecure`: Set this parameter to true.
- `registry.cn-beijing.aliyuncs.com`: the endpoint of the destination repository. In this example, the image is deployed in the China (Beijing) region.
  - `username`: the username in the credential.
  - `password`: the password in the credential.
- `"harbor.myk8s.paas.com:32080/library/nginx": ""`: access the *library/nginx* repository through the endpoint harbor.myk8s.paas.com:32080.

Use image-syncer to synchronize images

1. Download the latest installation package of image-syncer.

   > **Note** Only the Linux AMD64 version is supported.

2. Install and configure image-syncer.

   For more information, see Install and configure image-syncer.

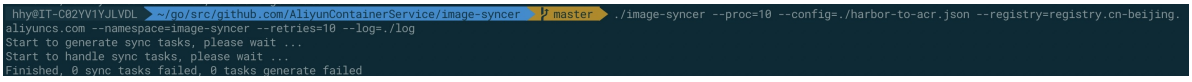
3. Run the following command to synchronize images:

```
# Set the default destination repository to registry.cn-beijing.aliyuncs.com and the default destination namespace to image-syncer.
# Set both the number of images that can be synchronized at a time and the maximum number of retries to 10.
# Record logs in the ./log file. If the file does not exist, it is automatically created. By default, image-syncer logs are stored in Stderr if the log file is not specified.
# Specify harbor-to-acr.json as the configuration file. Its content is described in the previous section.
./image-syncer --proc=10 --config=./harbor-to-acr.json --registry=registry.cn-beijing.aliyuncs.com --namespace=image-syncer --retries=10 --log=./log
```

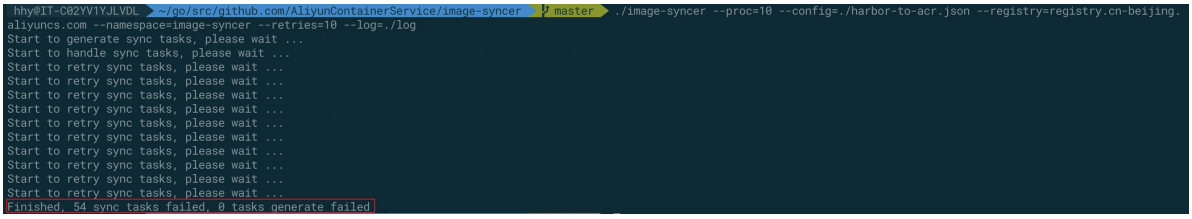
Synchronization result

Each time you synchronize an image, image-syncer generates a synchronization task, runs the task, and retries if the task fails. Each task synchronizes an image that is represented by a tag. If no tag is specified for a rule in the configuration file, image-syncer lists all the tags in the source repository and **generates synchronization tasks** for all the images. If image-syncer fails to generate synchronization tasks, image-syncer retries after it runs generated tasks.

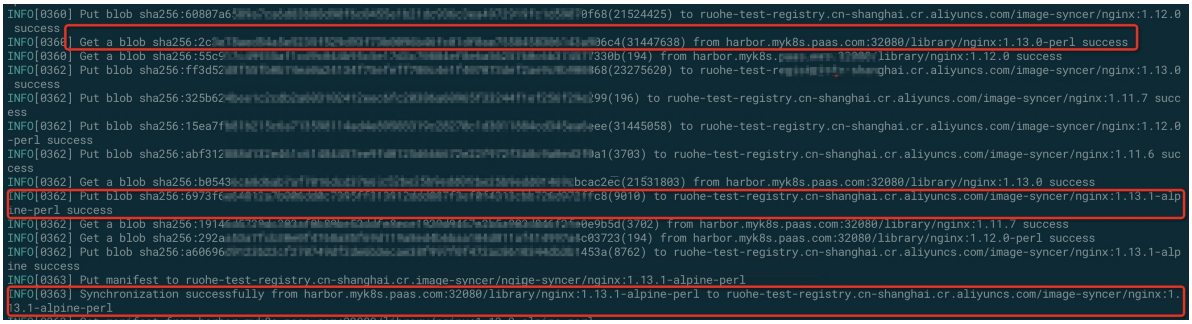
- The following figure shows the output of a successful synchronization task.



- The following figure shows the output of a failed synchronization task. Possible reasons include invalid usernames or passwords.



- The following figure shows the logs of image-syncer.



10.5.3. Synchronize images from a self-managed Harbor project to Container Registry Enterprise Edition

You can migrate images from a self-managed Harbor project to a Container Registry Enterprise Edition instance. You can also customize a domain name for the Container Registry Enterprise Edition instance. This topic describes how to synchronize images from a self-managed Harbor project to a Container Registry Enterprise Edition instance.

Context

Container Registry Enterprise Edition provides an enterprise-class secure service for managing container images and Helm charts. It provides enterprise-class security and allows you to distribute images to thousands of nodes concurrently and synchronize images across regions on a global scale. This service also allows you to create cloud-native application delivery chains to automatically deliver images globally upon source code changes in multiple scenarios. This service is suitable for enterprise customers that have high security requirements, deploy services in multiple regions, and use clusters that consist of a large number of nodes.

Step 1: Migrate backend data of the Harbor instance

- If your Harbor instance uses Apsara File Storage NAS as the backend storage, you must migrate data from NAS to an Object Storage Service (OSS) bucket. For more information, see Migrate data from NAS to OSS.
- Skip this step if the backend data of the Harbor instance is stored in OSS.

Step 2: Select an OSS bucket

When you create a Container Registry Enterprise Edition instance, you can select an existing OSS bucket as the backend storage of the instance.

1. Attach a RAM role to the account and grant the RAM role the permission to manage the OSS bucket. For more information, see Use RAM to grant permissions to access custom OSS buckets.

2. Create a Container Registry Enterprise Edition instance.

   When you create a Container Registry Enterprise Edition instance, set **Instance Storage** to **Custom** and select a bucket. For more information, see Create a Container Registry Enterprise Edition instance.

Step 3: Import images

1.
2.
3.
4.
5.
6. On the management page of the Container Registry Enterprise Edition instance, choose **Instances > Image Import** in the left-side navigation pane,
7. On the Image Import page, click **Trigger Task**.
8. In the **Tips** dialog box, select **Confirm to import** and click **Confirm**.

> **Note** On the Image Import page, find the created task and click **Details** in the **Actions** column to view the progress.

Step 4: Bind a custom domain name to the instance

You can bind a custom domain name that has a Secure Sockets Layer (SSL) certificate to a Container Registry Enterprise Edition instance. After you perform this operation, you can use the custom domain name to access the instance over HTTPS.

We recommend that you set the custom domain name of the Container Registry Enterprise Edition instance to the domain name of the self-managed Harbor instance. For more information, see Use a custom domain name to access a Container Registry Enterprise Edition instance.

10.6. Migrate applications to an ACK cluster

This topic describes how to use Velero and restic to migrate cloud-native applications and data on persistent volumes (PVs) from a self-managed Kubernetes cluster to a Container Service for Kubernetes (ACK) cluster. You can also use Velero and restic to migrate applications and data on PVs from a Kubernetes cluster that is managed by a third-party cloud provider to an ACK cluster.

Prerequisites

In most cases, a self-managed Kubernetes cluster is deployed in a data center and container images are stored in a self-managed image repository. Before you migrate applications, you must migrate the container images to Alibaba Cloud Container Registry (ACR). For more information, see Use image-syncer to migrate container images.

In this example, a WordPress application that uses the following container images is used as an example:

```
registry.api.paas.com:5000/admin/wordpress:latest
registry.api.paas.com:5000/admin/mysql:8
```

The following code block shows the addresses of the images after migration:

```
registry.cn-hangzhou.aliyuncs.com/ack-migration/wordpress:latest
registry.cn-hangzhou.aliyuncs.com/ack-migration/mysql:8
```

Context

In this example, the WordPress application is migrated from a self-managed Kubernetes cluster to an ACK cluster. The WordPress application consists of two components: WordPress and MySQL. Each component uses a Network File System (NFS) PV to store application data. The WordPress application provides services of the NodePort type.

Procedure

1. Set up the environment
2. Back up the application in the self-managed Kubernetes cluster
3. Restore the application in the ACK cluster
4. Update application configurations
5. Debug and start the application.

Set up the environment

Perform the following steps to deploy Velero in the ACK cluster and the self-managed Kubernetes cluster:

> **Note** You must deploy both the Velero client and the Velero server.

1. Install the Velero client.

   Download Velero. Run the following command to install Velero and verify the result:

   ```
   curl -o /usr/bin/velero https://public-bucket-1.oss-cn-hangzhou.aliyuncs.com/velero &&
   chmod +x /usr/bin/velero
   ```

2. Create an Object Storage Service (OSS) bucket. For more information, see Create buckets.

   You must first create an OSS bucket to store data of the application and PVs. We recommend that you create an OSS bucket for each self-managed Kubernetes cluster.

   i. Log on to the OSS console.

   ii. On the **Overview** page, click **Create Bucket** on the right side. You can also click the Create icon (+) next to Storage Capacity on the left side.

   iii. In the **Create Bucket** panel, set parameters for the bucket.

   In this example, the OSS bucket is named `ls-velero` and deployed in the China (Hangzhou) region.

3. Create a Resource Access Management (RAM) user and generate an AccessKey pair. For more information, see Create a RAM user.

If you use the AccessKey pair of an Alibaba Cloud account, skip this step.

```
{
    "Version": "1",
    "Statement": [
        {
            "Action": [
                "ecs:DescribeSnapshots",
                "ecs:CreateSnapshot",
                "ecs:DeleteSnapshot",
                "ecs:DescribeDisks",
                "ecs:CreateDisk",
                "ecs:Addtags",
                "oss:PutObject",
                "oss:GetObject",
                "oss:DeleteObject",
                "oss:GetBucket",
                "oss:ListObjects"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

4. Deploy the Velero server.
   i. Add the AccessKey pair generated in Step 3 to the *credentials-velero* deployment file of Velero.

   ```
   ALIBABA_CLOUD_ACCESS_KEY_ID=<access_key_id>
   ALIBABA_CLOUD_ACCESS_KEY_SECRET=<access_key_secret>
   ```

   ii. Run the following command to deploy Velero:

   ```
   velero install --provider alibabacloud --image registry.cn-hongkong.aliyuncs.com/acs/velero:1.5.2-e99d74d4-aliyun --bucket ls-velero --secret-file ./credentials-velero --use-volume-snapshots=false --backup-location-config region=cn-hangzhou --use-restic --plugins registry.cn-hangzhou.aliyuncs.com/acs/velero-plugin-alibabacloud:v1.5.1-a063222 --wait
   ```

   Run the following command to query the pod status:

   ```
   kubectl -n velero get po
   ```

   Expected output:

```
NAME                      READY   STATUS    RESTARTS   AGE
restic-fqwsc              1/1     Running   0          41s
restic-kfzqt              1/1     Running   0          41s
restic-klxhc              1/1     Running   0          41s
restic-ql2kr              1/1     Running   0          41s
restic-qrsrn              1/1     Running   0          41s
restic-srjmm              1/1     Running   0          41s
velero-67b975f5cb-68nj4   1/1     Running   0          41s
```

Back up the application in the self-managed Kubernetes cluster

- If you want to back up only the WordPress application, run the following command:

  ```
  velero backup create wordpress-backup-without-pv --include-namespaces wordpress
  ```

  Expected output:

  ```
  Backup request "wordpress-backup-without-pv" submitted successfully.
  Run `velero backup describe wordpress-backup-without-pv` or `velero backup logs wordpress
  -backup-without-pv` for more details.
  ```

  Run the following command to back up the application:

  ```
  velero backup get
  ```

  Expected output:

  ```
  NAME                          STATUS      CREATED                         EXPIRES   STORA
  GE LOCATION   SELECTOR
  wordpress-backup-without-pv   Completed   2019-12-12 14:08:24 +0800 CST   29d       defau
  lt            <none>
  ```

- If you want to back up the data of the WordPress application and PVs, run the following commands:

  ```
  # Annotate the pods to which PVs are attached. In this example, the WordPress applica
  tion runs on the wordpress-7cf5849f47-mbvx4 and mysql-74dddbdcc8-h2tls pods.
  kubectl -n wordpress annotate pod/wordpress-7cf5849f47-mbvx4 backup.velero.io/backup-
  volumes=wordpress-persistent-storage
  pod/wordpress-7cf5849f47-mbvx4 annotated
  ```

  ```
  # The PV attached to the wordpress-7cf5849f47-mbvx4 pod is mysql-persistent-storage.
  The PV attached to the mysql-74dddbdcc8-h2tls pod is wordpress-persistent-storage. Ru
  n the following commands to annotate the pods:
  kubectl -n wordpress annotate pod/mysql-74dddbdcc8-h2tls backup.velero.io/backup-volu
  mes=mysql-persistent-storage
  pod/mysql-74dddbdcc8-h2tls annotated
  ```

  ```
  # Back up the WordPress application.
  velero backup create wordpress-backup-with-pv --include-namespaces wordpress
  ```

  Expected output:

```
Backup request "wordpress-backup-with-pv" submitted successfully.
Run `velero backup describe wordpress-backup-with-pv` or `velero backup logs wordpress-backup-with-pv` for more details.
```

```
velero backup get
```

Expected output:

```
NAME                          STATUS      CREATED                         EXPIRES   S
TORAGE LOCATION   SELECTOR
wordpress-backup-with-pv      Completed   2019-12-12 14:23:40 +0800 CST   29d       d
efault            <none>
wordpress-backup-without-pv   Completed   2019-12-12 14:08:24 +0800 CST   29d       d
efault            <none>
```

In the OSS console, you can view the files that have been backed up in the OSS bucket.

Restore the application in the ACK cluster

In the self-managed Kubernetes cluster, the WordPress application uses NFS PVs. In the ACK cluster, you can configure Apsara File Storage NAS volumes to store data of the WordPress application. In this example, a StorageClass named nfs is created for the WordPress application, and Alibaba Cloud SSDs are used as backend storage.

The CSI plug-in is used in this example. For more information, see Use a dynamically provisioned disk volume.

1. Create a StorageClass.

   If you back up only the WordPress application, skip this step.

   ```
   cat nfs.yaml
   ```

   The following code block shows a sample template:

   ```
   # Output
   apiVersion: storage.k8s.io/v1
   kind: StorageClass
   metadata:
      name: nfs
   provisioner: diskplugin.csi.alibabacloud.com
   parameters:
       type: cloud_ssd
   reclaimPolicy: Retain
   ```

   Run the following command to deploy the application:

   ```
   kubectl apply -f nfs.yaml
   ```

   Expected output:

   ```
   storageclass.storage.k8s.io/nfs created
   ```

2. Restore the WordPress application.

   Use Velero to restore the WordPress application in the ACK cluster. This completes the migration process.

```
velero restore create --from-backup wordpress-backup-with-pv
velero restore get
```

Expected output:

```
NAME                                        BACKUP                      STATUS       WARNI
NGS   ERRORS   CREATED                         SELECTOR
wordpress-backup-with-pv-20191212152745   wordpress-backup-with-pv   InProgress   0
0        2019-12-12 15:27:45 +0800 CST   <none>
```

Run the following command to check the status of the WordPress application. An error is returned to indicate that the container images cannot be pulled.

```
$ kubectl -n wordpress get po
```

Expected output:

```
NAME                         READY   STATUS         RESTARTS   AGE
mysql-669b4666cd-trsnz       0/1     ErrImagePull   0          19m
mysql-74dddbdcc8-h2tls       0/1     Init:0/1       0          19m
wordpress-7cf5849f47-mbvx4   0/1     Init:0/1       0          19m
wordpress-bb5d74d95-xcjxw    0/1     ErrImagePull   0          19m
```

Update application configurations

Application configurations include the image address, service exposure method, and the mount point of the storage disks used by the application. In this example, only the image address is updated.

1.
2.
3. In the left-side navigation pane, Choose **Applications > Deployments**. Select the cluster and namespace to which the application is migrated.
4. Find the WordPress application, and choose **More > View in Yaml** in the **Actions** column.
5. In the **Edit YAML** dialog box, change the `image` field to the address of the application image after migration and click **Update**.

> **Note** You can obtain the address of the application image after migration from the Prerequisites section in this topic.

Run the following command to check the status of the WordPress application:

```
kubectl -n wordpress get po
```

Expected output:

```
NAME                         READY   STATUS    RESTARTS   AGE
mysql-678b5d8499-vckfd       1/1     Running   0          100s
wordpress-8566f5f7d8-7shk6   1/1     Running   0          3m18s
```

Debug and start the application.

The WordPress application consists of the WordPress and MySQL components. Each component uses an NFS PV to store application data. The WordPress application provides services of the NodePort type. You can use the following YAML file to debug and start the application:

```
# 1. Create a StorageClass named nfs.
cat nfs-sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: nfs
provisioner: helm.default/nfs
reclaimPolicy: Delete
kubectl apply -f  nfs-sc.yaml
# 2. Create a Secret to store the password of the MySQL component. For example, if the pass
word is mysql, run the echo -n "mysql" |base64 command to query the Base64-encoded string o
f the password.
cat secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql
type: Opaque
data:
  password: bXlzcWw=
kubectl apply -f secret.yaml
# 3. Create a Service, a persistent volume claim (PVC), and a Deployment for the MySQL comp
onent.
cat mysql.yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  type: ClusterIP
  ports:
    - port: 3306
  selector:
    app: mysql
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-volumeclaim
  annotations:
    volume.beta.kubernetes.io/storage-class: "nfs"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
```

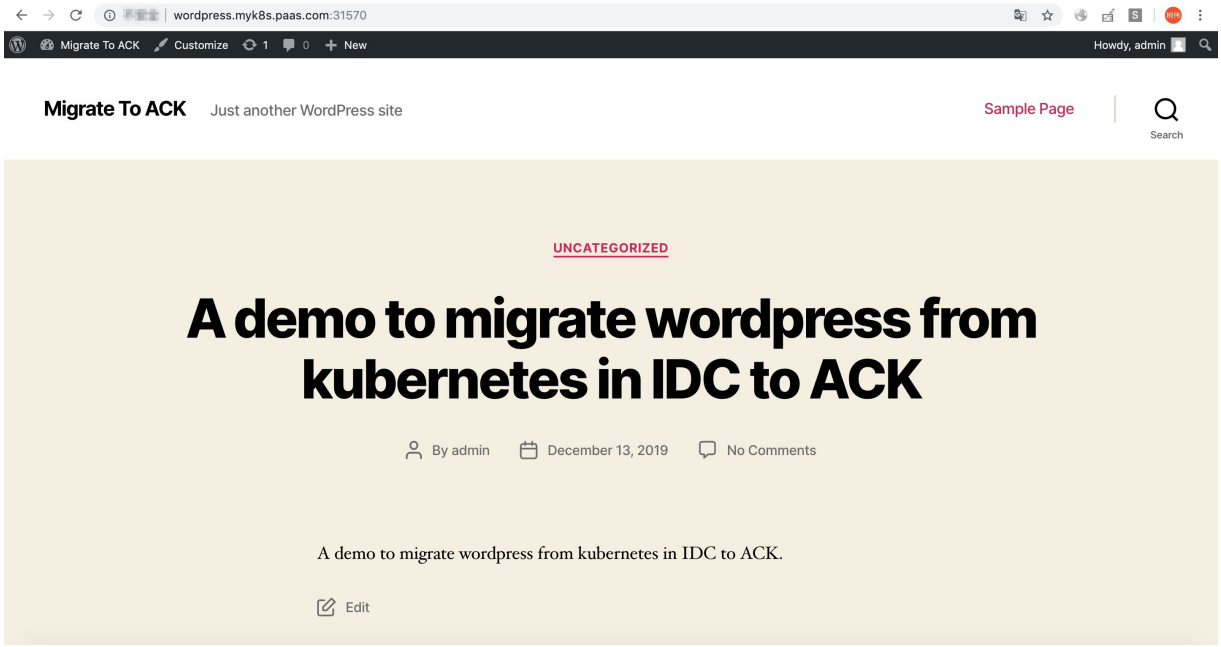
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      securityContext:
        runAsUser: 999
        runAsGroup: 999
        fsGroup: 999
      containers:
        - image: registry.api.paas.com:5000/admin/mysql:8
          name: mysql
          args:
            - "--default-authentication-plugin=mysql_native_password"
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql
                  key: password
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-volumeclaim
kubectl apply -f mysql.yaml
 # 4. Create a PVC, a Deployment, and a Service for the WordPress component.
cat wordpress.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: wordpress
  name: wordpress
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
```

```
      protocol: TCP
      nodePort: 31570
  selector:
    app: wordpress
  type: NodePort
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wordpress-volumeclaim
  annotations:
    volume.beta.kubernetes.io/storage-class: "nfs"
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  replicas: 1
  selector:
    matchLabels:
      app: wordpress
  template:
    metadata:
      labels:
        app: wordpress
    spec:
      containers:
        - image: registry.api.paas.com:5000/admin/wordpress
          name: wordpress
          env:
          - name: WORDPRESS_DB_HOST
            value: mysql:3306
          - name: WORDPRESS_DB_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql
                key: password
          ports:
            - containerPort: 80
              name: wordpress
          volumeMounts:
            - name: wordpress-persistent-storage
              mountPath: /var/www/html
      volumes:
        - name: wordpress-persistent-storage
          persistentVolumeClaim:
```

```
            claimName: wordpress-volumeclaim
kubectl apply -f wordpress.yaml
```

Result

In the test environment, bind the domain name of the WordPress application to the IP address of the node where the application is deployed in the hosts file of your operating system. Then, access the WordPress application through *http://wordpress.myk8s.paas.com:31570/*.



11.Migrate applications from a Swarm cluster to a Kubernetes cluster

11.1. Swarm cluster to Kubernetes Cluster features comparison

11.1.1. Overview

This topic describes the prerequisites and limits that are used to compare clusters of Container Service for Swarm and clusters of Container Service for Kubernetes (ACK).

Prerequisites

An ACK cluster is created. For more information, see Create an ACK dedicated cluster.

> Note
>
> - ACK provides the following types of clusters: the dedicated Kubernetes cluster, the managed Kubernetes cluster, the multi-zone Kubernetes cluster, and the serverless Kubernetes (ASK) cluster (in public preview).
> - In this topic, an application is created in a Swarm and an ACK cluster to compare the features of Swarm and ACK clusters.

Limits

In this topic, the features of Swarm and Kubernetes clusters are compared based on the following limits:

- Both applications are stateless.
- Application data is stored in databases or volumes.

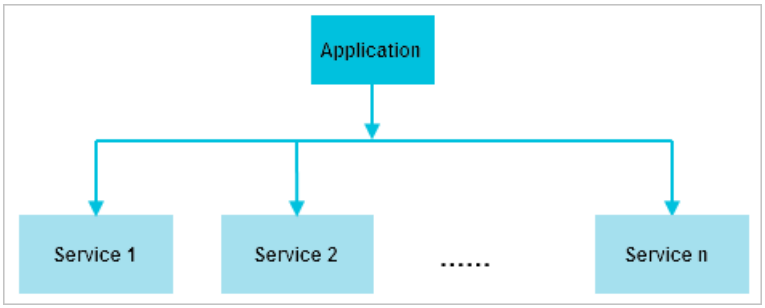
11.1.2. Concepts

This topic describes the differences in the concepts used in Swarm and Kubernetes clusters.

Application

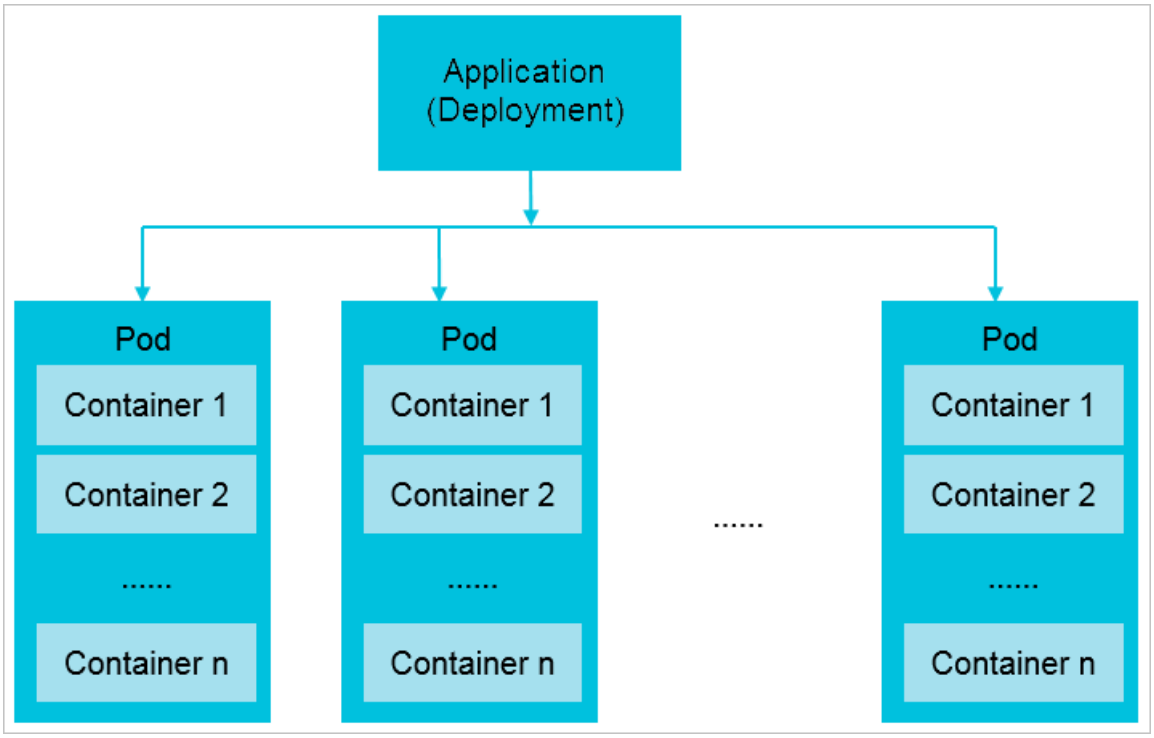
Swarm cluster

In Swarm clusters, applications are similar to projects. One application consists of multiple services. A service is an instance that provides specific functions of an application. Services support horizontal scaling.



Kubernetes cluster

In Kubernetes clusters, applications are deployments that expose functions externally. A deployment manages one or more pods and containers. Pods are the smallest deployable computing units in Kubernetes. One pod may consist of multiple containers. A pod is meant to run a single instance of an application. Pods support horizontal scaling and can be scheduled to different nodes.



Note In the preceding figure, one pod consists of multiple containers. We recommend that you use one pod to run a single container in practical use.

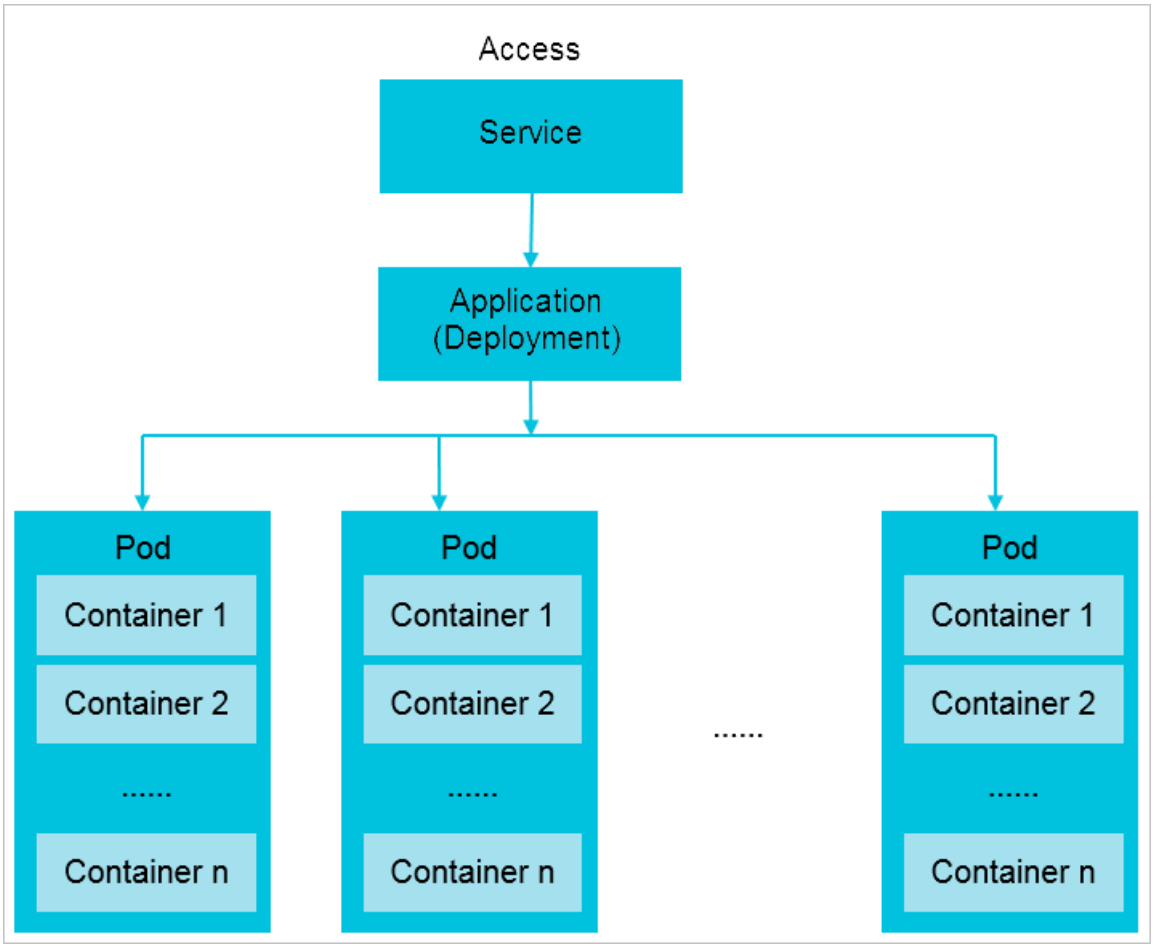
Service

Swarm cluster

In Swarm clusters, a service is an instance of an application that provides specific functions. When you create an application in a Swarm cluster, access to the application services is directly exposed.

Kubernetes cluster

In Kubernetes clusters, service is an abstract concept. You can use services to expose deployments to external users.



Access an application

Swarm cluster

When you create an application in a Swarm cluster, you can choose one of the following methods to expose the application. All these methods enable external access to the application.

- <HostIP>:<port>
- Simple routing
- SLB routing

Kubernetes cluster

After you create an application in a Kubernetes cluster, you need to create services to expose the application. To access an application from within a Kubernetes cluster, you can use service names, which only support internal access. To access an application from outside a Kubernetes cluster, you can create NodePort or LoadBalancer services to expose the application.

- ClusterIP (Supports internal access)
- NodePort (Similar to <HostIP>:<port> in Swarm clusters)
- LoadBalancer (Similar to SLB routing in Swarm clusters)
- Domain names based on ingresses. (Similar to simple routing in Swarm clusters)

11.1.3. Comparison of basic settings for creating an application from an image

This topic describes the differences in the basic settings when you create an application from an image in a cluster of Container Service for Swarm and in a cluster of Container Service for Kubernetes (ACK).

Create an application from an image

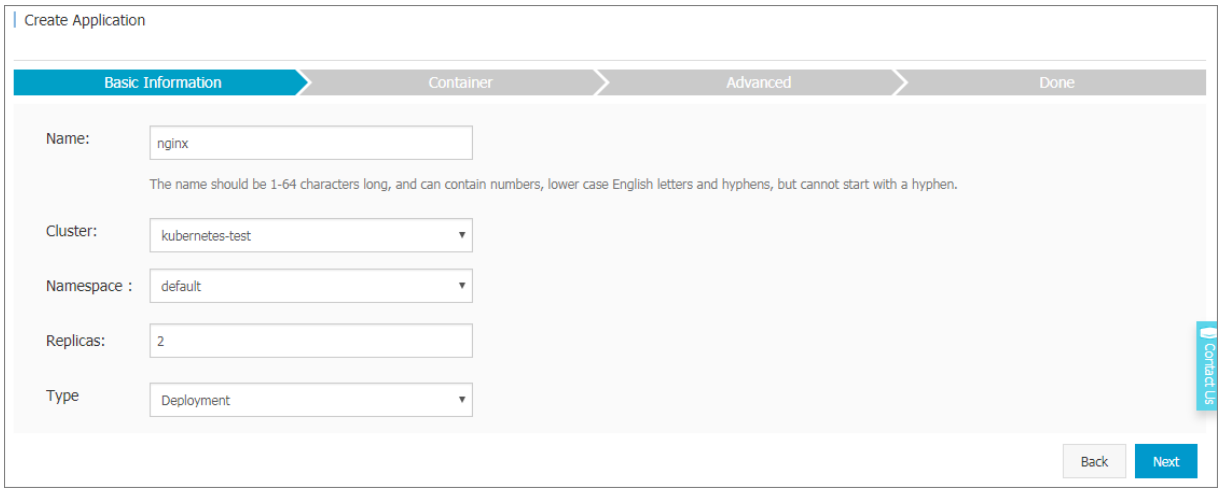
The interfaces for creating an application from an image in a Swarm cluster and in an ACK cluster are quite different.

- For more information about how to create an application from an image in a Swarm cluster, see Create an application.
- For more information about how to create an application from an image in an ACK cluster, see Create a stateless application by using a Deployment.

Basic settings

Swarm cluster

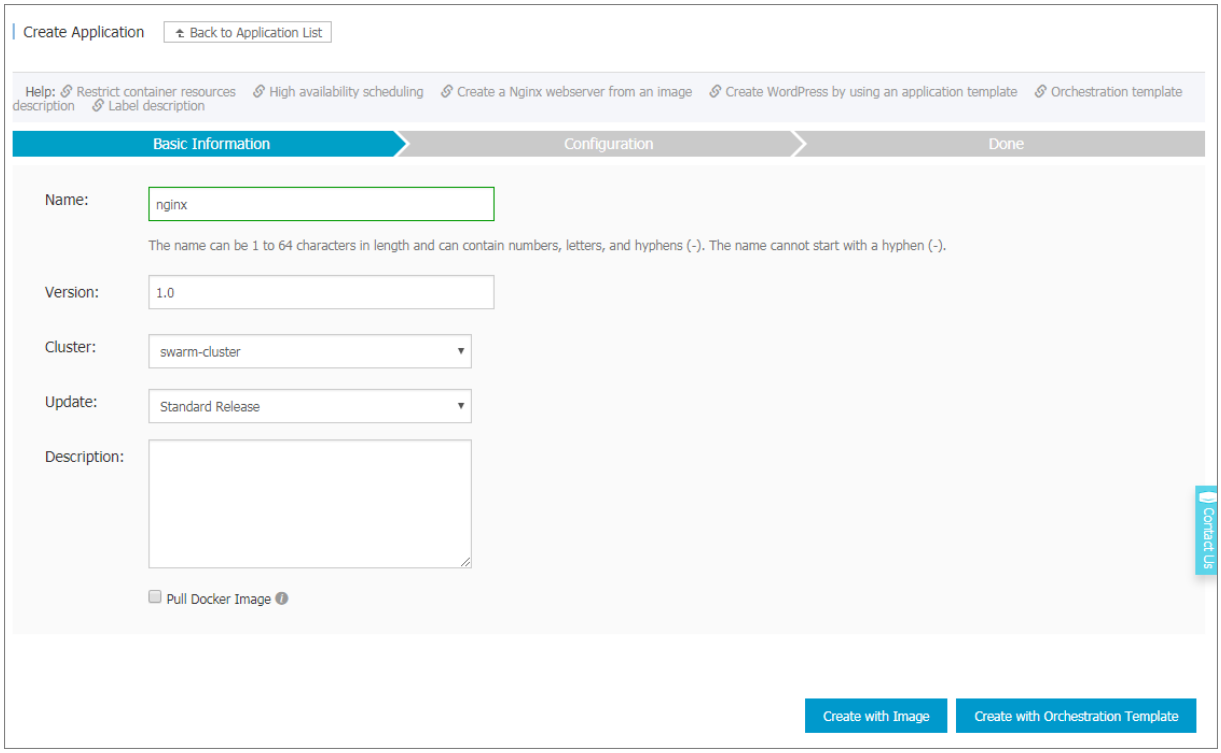
On the Basic Information wizard page, you must specify the following information: application name, application version, cluster for deployment, deployment model, and description.



ACK cluster

On the Basic Information wizard page, you must specify the following information that is required only by an ACK cluster: namespace for deployment, number of replicas, and application type.

Namespace is a Kubernetes-specific concept. Namespaces are used to isolate resources such as CPU resources within a Kubernetes cluster. You can also use namespaces to distinguish different environments, such as staging environments and development environments. If you want to isolate resources among production environments, we recommend that you create a cluster for each production environment. For more information about Kubernetes namespaces, see Basic concepts.

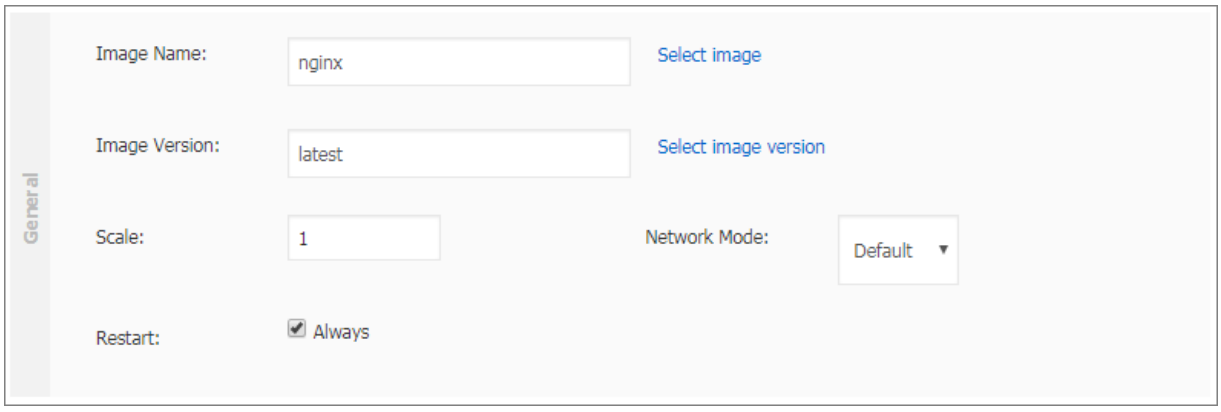


General

In the General section, you must specify the image name and image version.

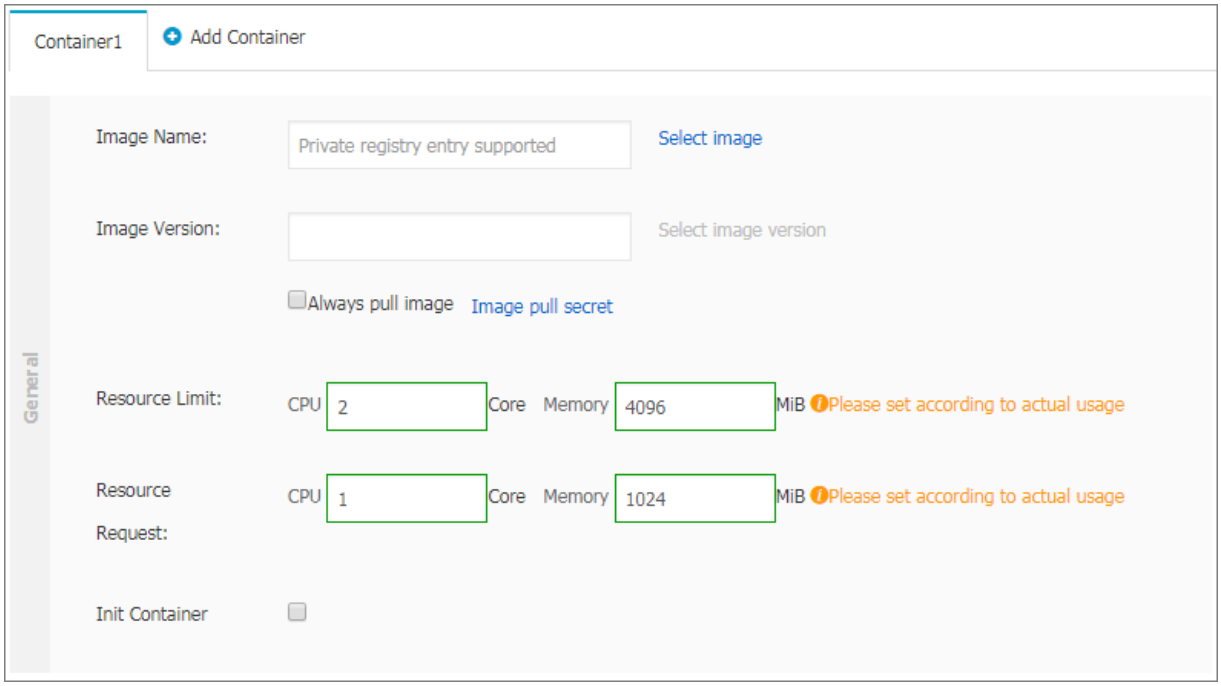
Swarm cluster

You can set **Network Mode** to **Default** or **host**.



ACK cluster

- The network mode is automatically selected when you create an ACK cluster. You can select **Flannel** or **Terway** as the network plug-in. For more information, see Work with Terway.
- The Required Resources field specifies the amount of CPU and memory resources that are required by a container. The Resource Limit field specifies the maximum resource amount that can be allocated to a container. These fields are similar to the **CPU Limit** and **Memory Limit** fields in the **Container** section when you create an application from an image in a Swarm cluster.



11.1.4. Network configurations for deploying an application from an image

This topic describes the differences in network configurations when you deploy an application from an image in a Container Service for Swarm cluster and in a Container Service for Kubernetes (ACK) cluster.

Deploy an application from an image

The user interfaces for deploying an application from an image in a Container Service for Swarm cluster and in an ACK cluster are quite different.

- For more information about how to deploy an application from an image in a Container Service for Swarm cluster, see Create an application.
- For more information about how to deploy an application from an image in an ACK cluster, see Create a stateless application by using a Deployment.

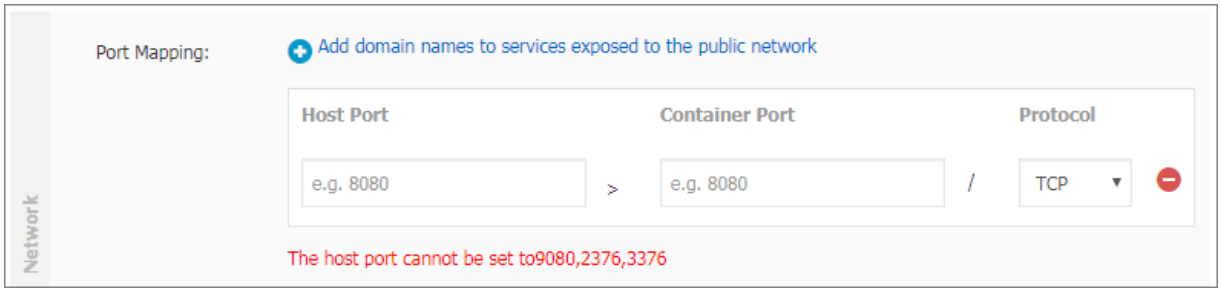
Network configurations

In Container Service for Swarm clusters, **network configurations** are used to expose applications to external access.

Port mapping

Container Service for Swarm cluster

You can configure **port mapping** to map the application port to a port on the host. After you specify a host port number, the application port is mapped to this port on each host. You can access the application by sending requests to `<HostIP>:<Port>`.



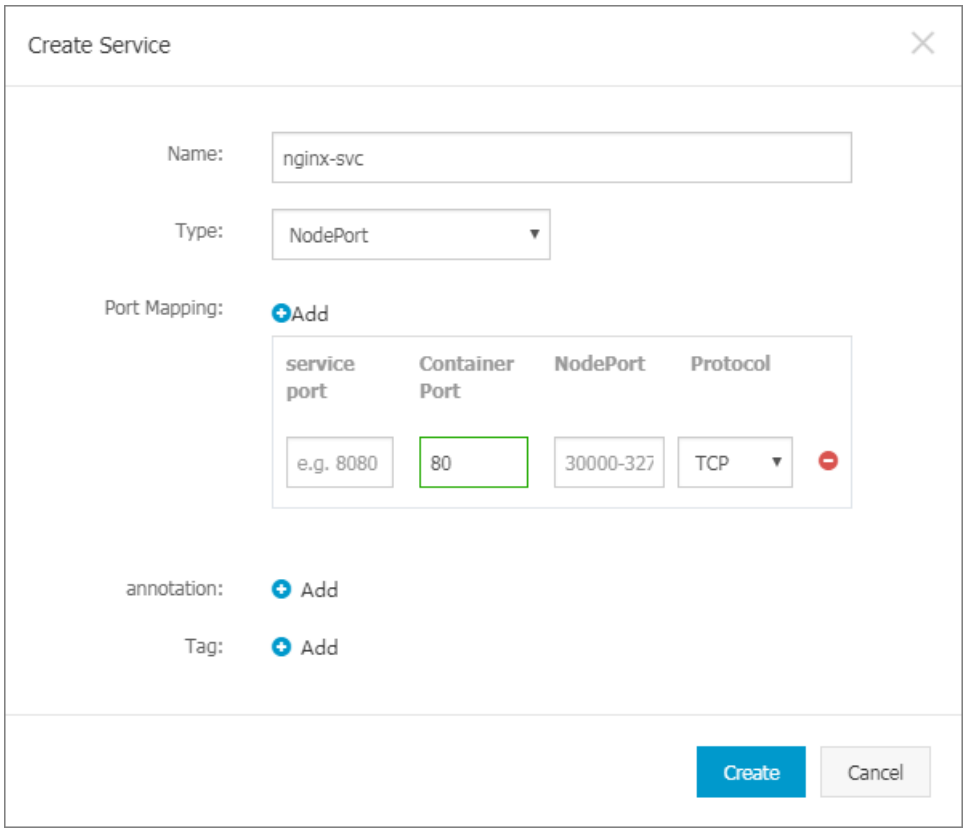
ACK cluster

You can create a **NodePort** Service to expose your application to external access. You can use one of the following methods to create a Service:

Method 1: Create a NodePort Service when you deploy an application

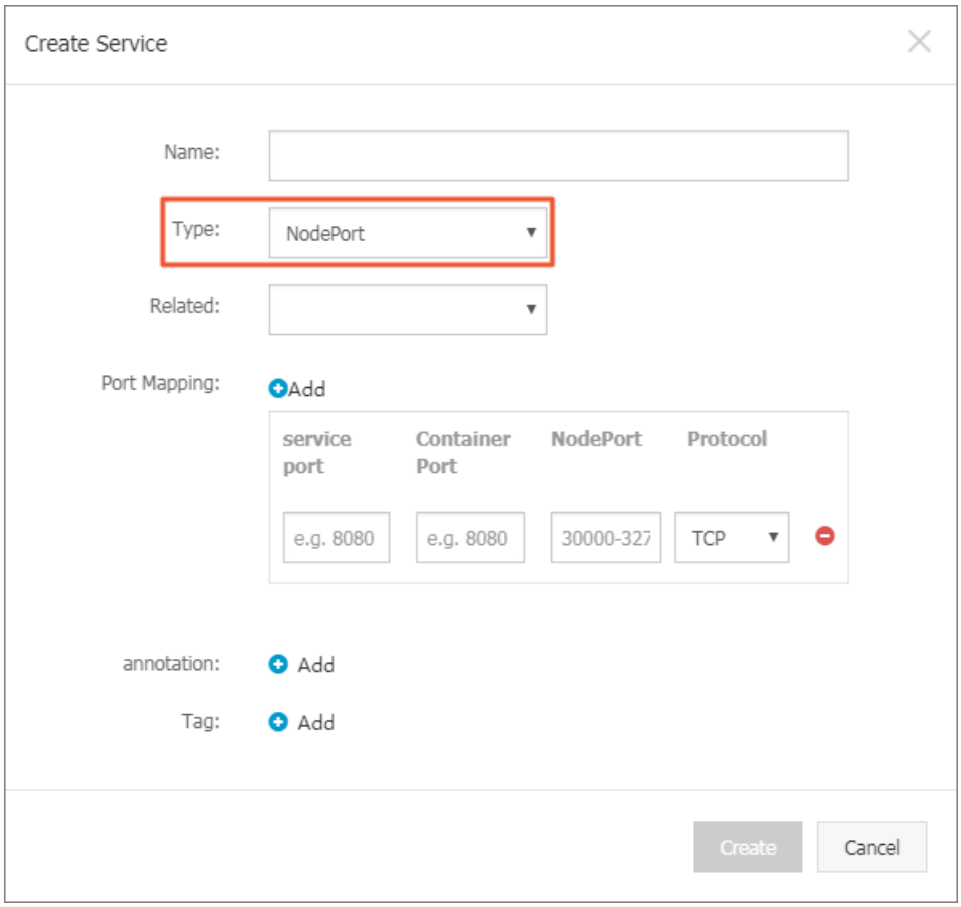
1. After you complete the settings on the **Container** wizard page, proceed to the **Advanced** wizard page. In the **Access Control** section, click **Create** on the right side of **Services**.

2. Select **Node Port** from the **Type** drop-down list. For more information, see Create a stateless application by using a Deployment.



Method 2: Directly create a NodePort Service

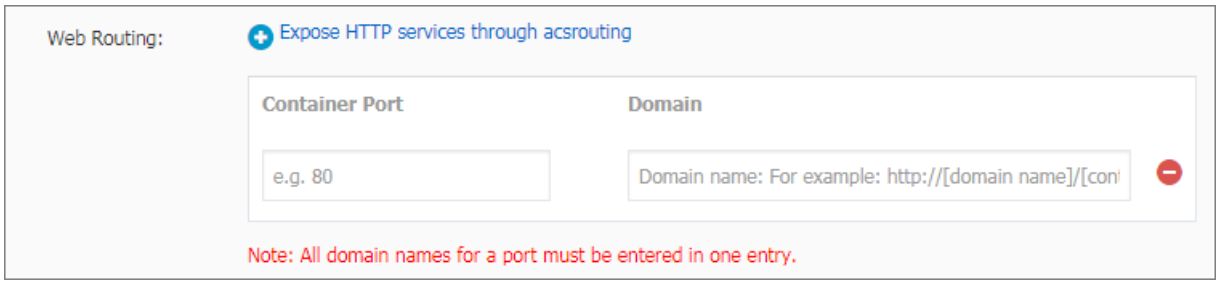
1.
2.
3.
4.
5. Select the namespace to which the Service belongs. In the upper-right corner of the Services page, click **Create**. In the **Create Service** dialog box, select **Node Port** from the **Type** drop-down list. For more information, see Manage Services.



Simple routing

Container Service for Swarm cluster

You can configure **simple routing** to expose your application through a domain name. You can specify a custom domain name or use the default domain name that is provided by Container Service for Swarm.



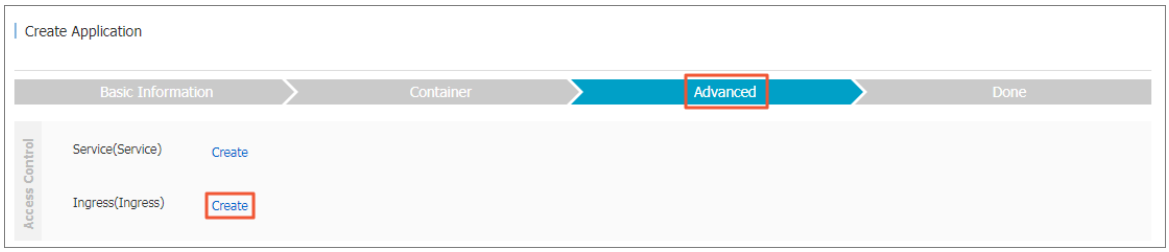
ACK cluster

You can create an Ingress to implement simple routing and other related features. You can also use Ingresses to implement blue-green releases and canary releases for applications in ACK clusters. For more information, see Use Ingresses to implement canary releases and blue-green releases.

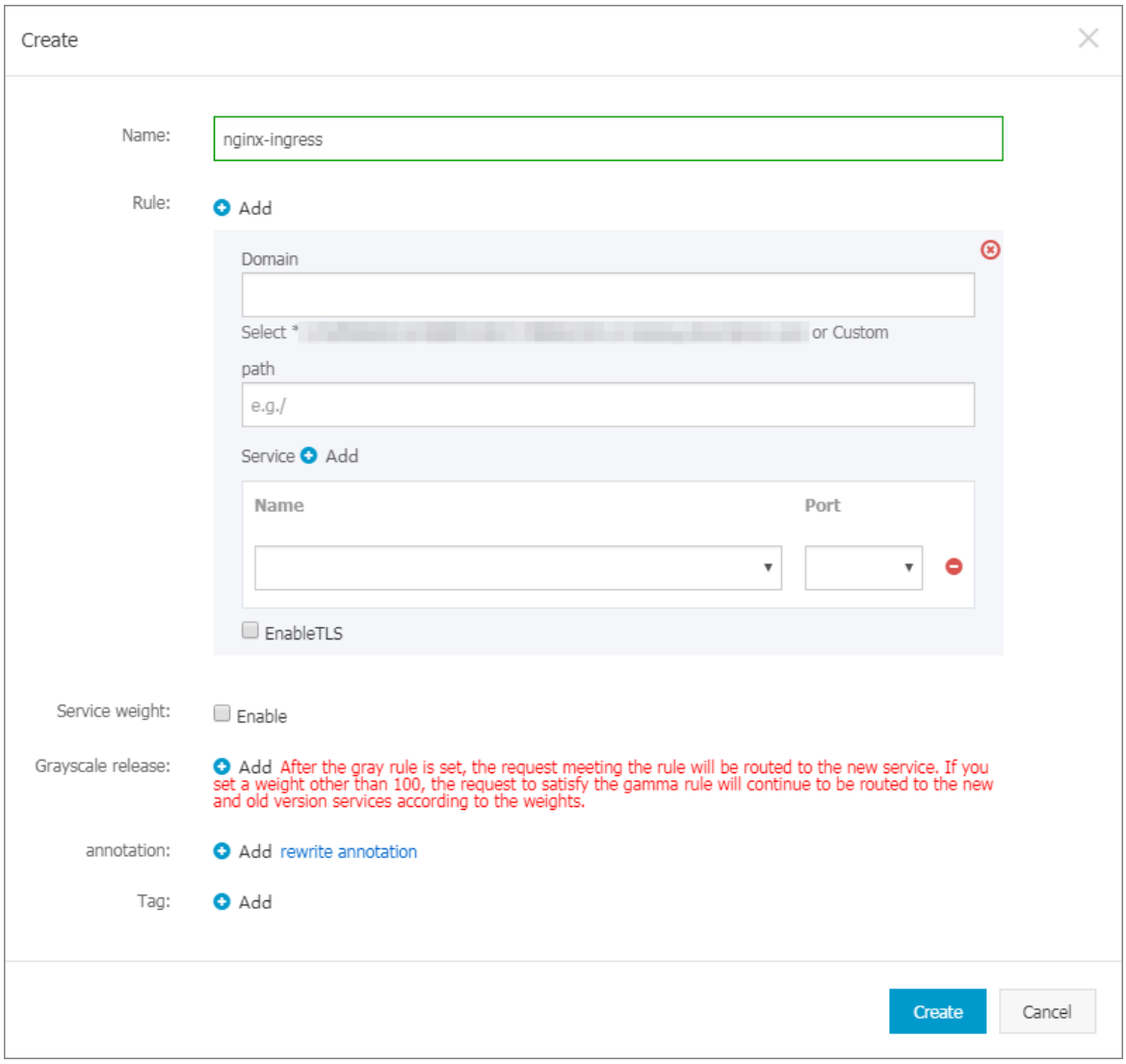
You can use one of the following methods to create an Ingress:

Method 1: Create an Ingress when you deploy an application

1. After you complete the settings on the **Container** wizard page, proceed to the **Advanced** wizard page. In the **Access Control** section, click **Create** on the right side of **Ingresses**.

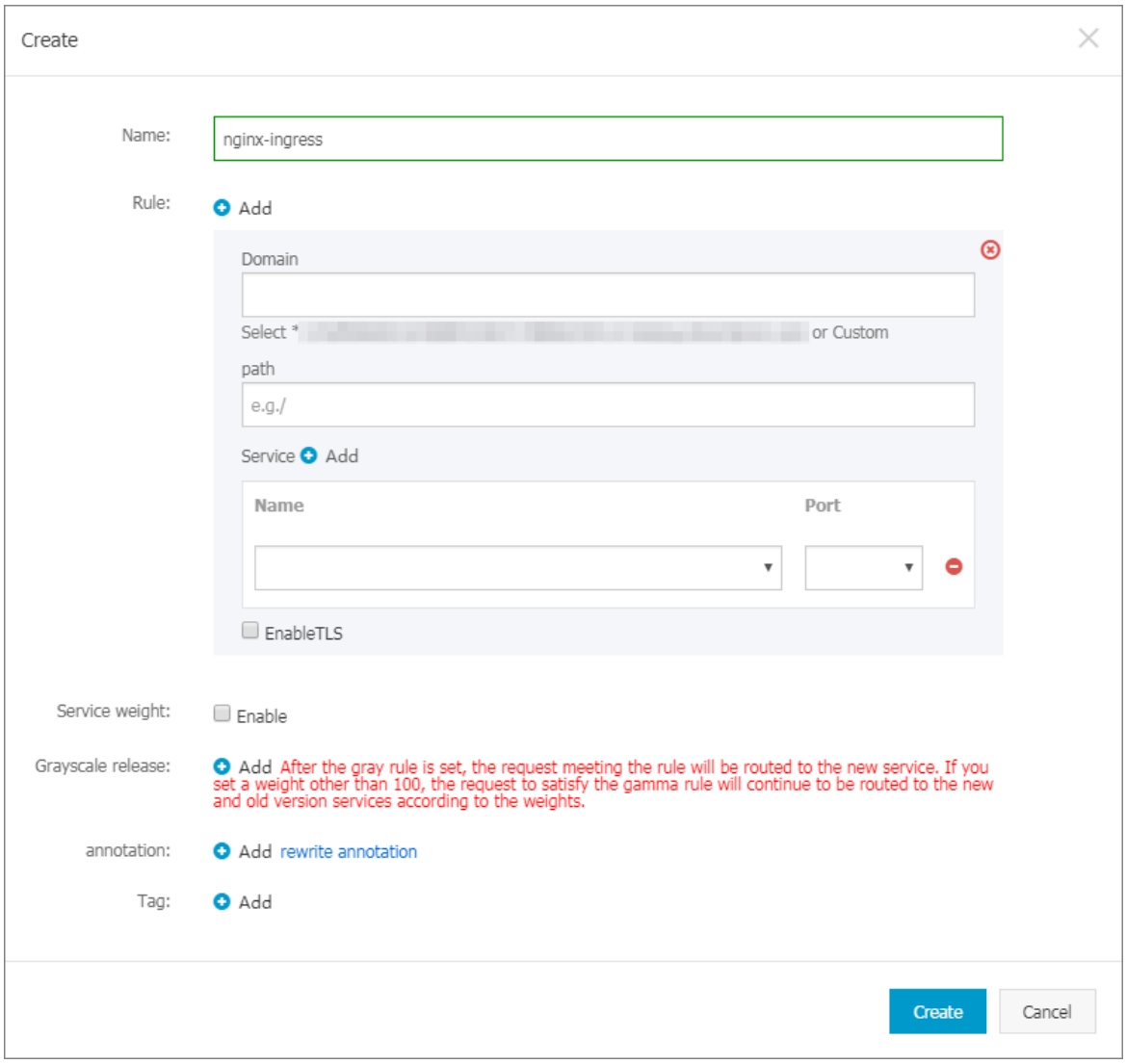


2. Deploy a stateless application from an image. For more information, see Create a stateless application by using a Deployment.



Method 2: Directly create an Ingress

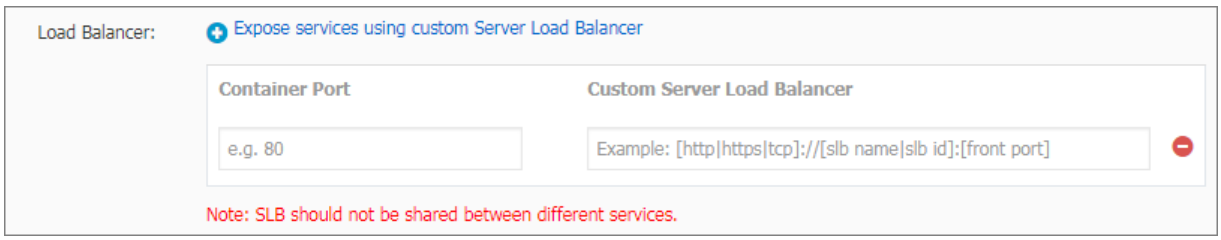
1.
2.
3.
4.
5. Select the namespace to which the Ingress belongs and click **Create** in the upper-right corner of the Ingresses page. For more information, see Manage Ingresses in the ACK console.



Load balancing

Container Service for Swarm cluster

You can configure **load balancing** to expose your application by using Server Load Balancer (SLB). You must create an SLB instance and associate the instance IP and port with your application. Then, you can access the application by sending requests to <SLB_IP>:<Port>.



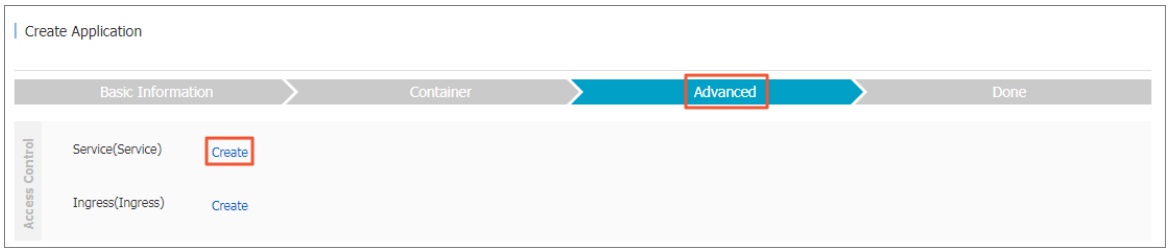
ACK cluster

You can also use an SLB instance to expose your application that is deployed in an ACK cluster. You do not need to manually create and configure an SLB instance. A LoadBalancer Service automatically creates an SLB instance for you. You can specify whether the SLB instance is used to enable access over the Internet or a private network. If you create a LoadBalancer Service by using a YAML template, you can specify to use an existing SLB instance and enable the session persistence feature. For more information, see Manage Services.

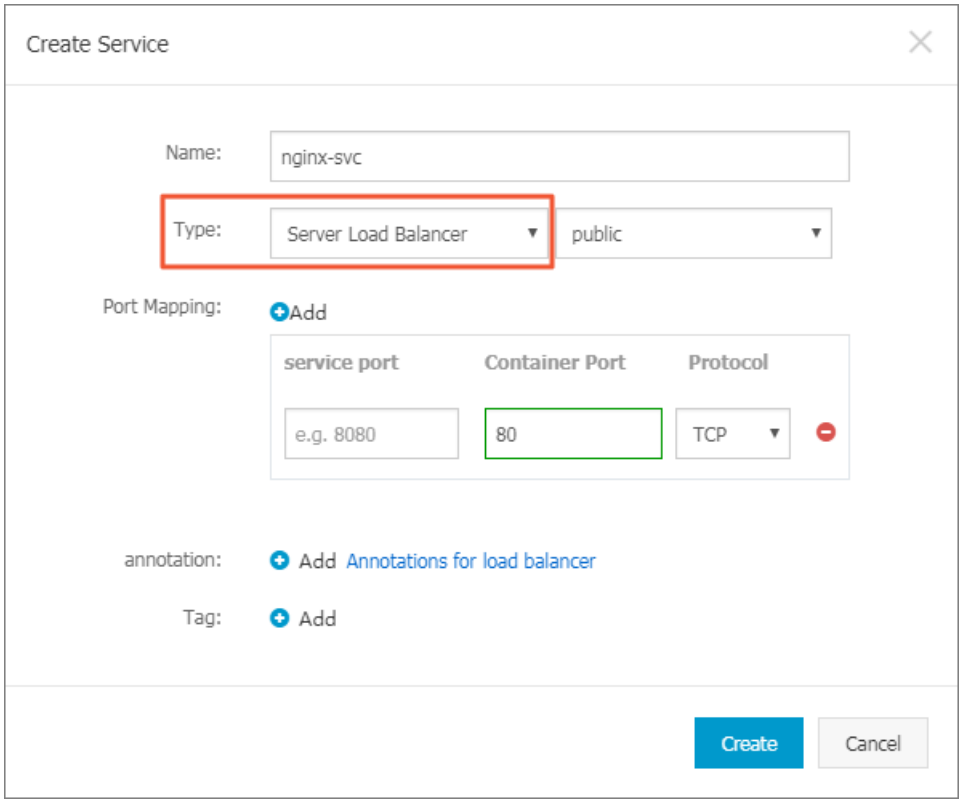
You can use one of the following methods to create a LoadBalancer Service:

Method 1: Create a LoadBalancer Service when you deploy an application

1. After you complete the settings on the **Container** wizard page, proceed to the **Advanced** wizard page. In the **Access Control** section, click **Create** on the right side of **Services**.

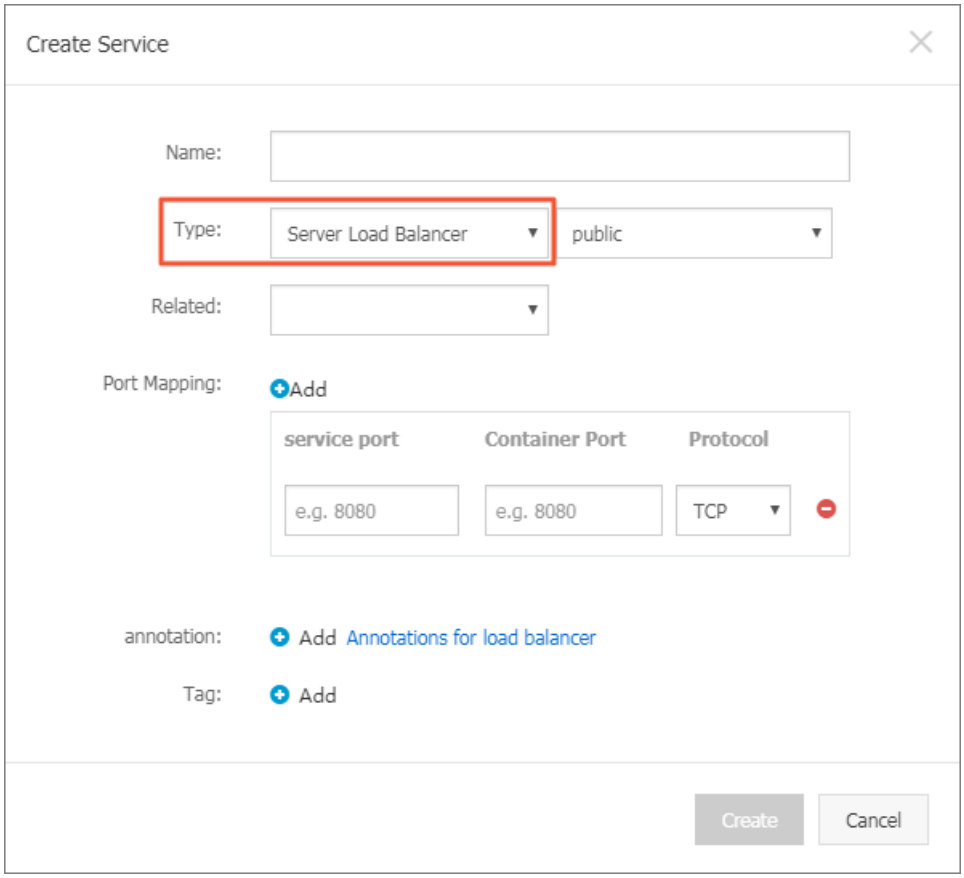


2. Select **Server Load Balancer** from the **Type** drop-down list. For more information, see Create a stateless application by using a Deployment.



Method 2: Directly create a LoadBalancer Service

1.
2.
3.
4.
5. Select the namespace to which the Service belongs. In the upper-right corner of the Services page, click **Create**. In the **Create Service** dialog box, select **Server Load Balancer** from the **Type** drop-down-list. For more information, see Manage Services.



11.1.5. Volume and environment variables

This topic describes the differences in volume and environment variable configurations when you create an application from an image in a Container Service for Swarm cluster and in a Container Service for Kubernetes (ACK) cluster.

Create an application from an image

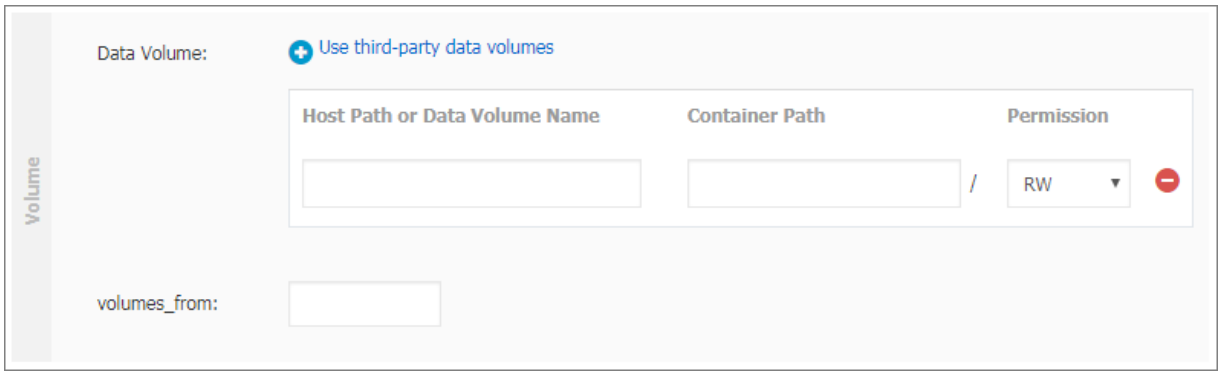
The interfaces for creating an application from an image in a Swarm cluster and in an ACK cluster are quite different.

- For more information about how to create an application from an image in a Swarm cluster, see Create an application.
- For more information about how to create an application from an image in an ACK cluster, see Create a stateless application by using a Deployment.

Volume

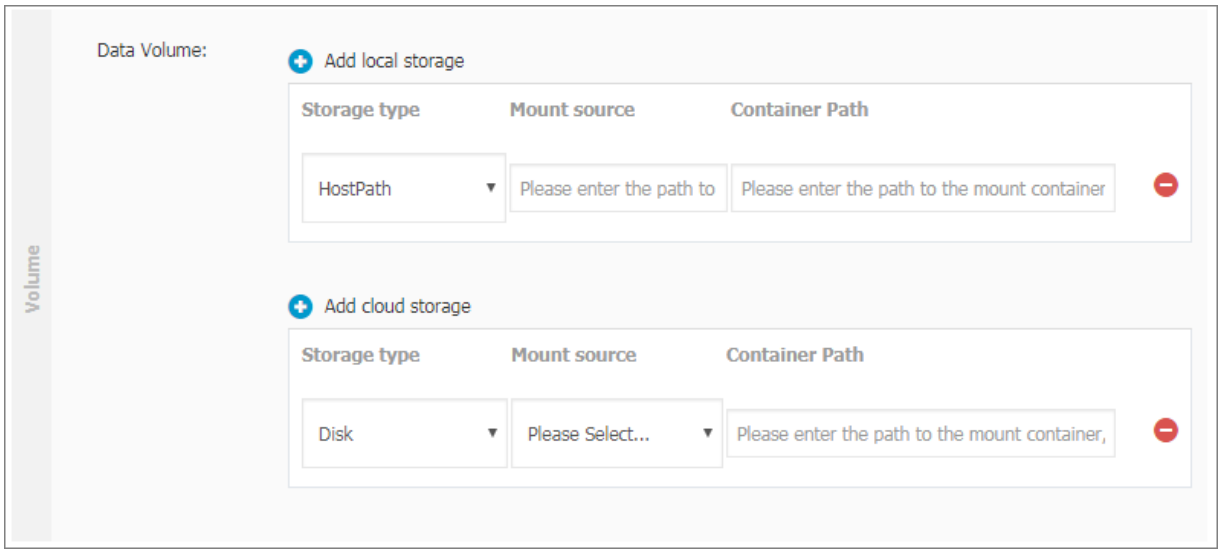
Swarm cluster

You need to specify your cloud or on-premises storage path.



ACK cluster

You can configure storage media in ACK clusters in the same way as in Swarm clusters, except that ACK clusters use a different method to mount storage media.



You can use on-premises storage or cloud storage.

- On-premises storage includes host directories, ConfigMaps, Secrets, and temporary directories.
- Cloud storage includes cloud disks, Apsara File Storage NAS (NAS) file systems, and Object Storage Service (OSS) buckets.

Environment variable

You can configure **environment variables** in ACK clusters in the same way as in Swarm clusters. You need only to enter a key and a value to define an environment variable.

11.1.6. Container configurations and labels for creating an application from an image

This topic describes the differences in container configurations and labels when you create an application from an image in a Container Service for Swarm cluster and in a Container Service for Kubernetes (ACK) cluster.

Create an application from an image

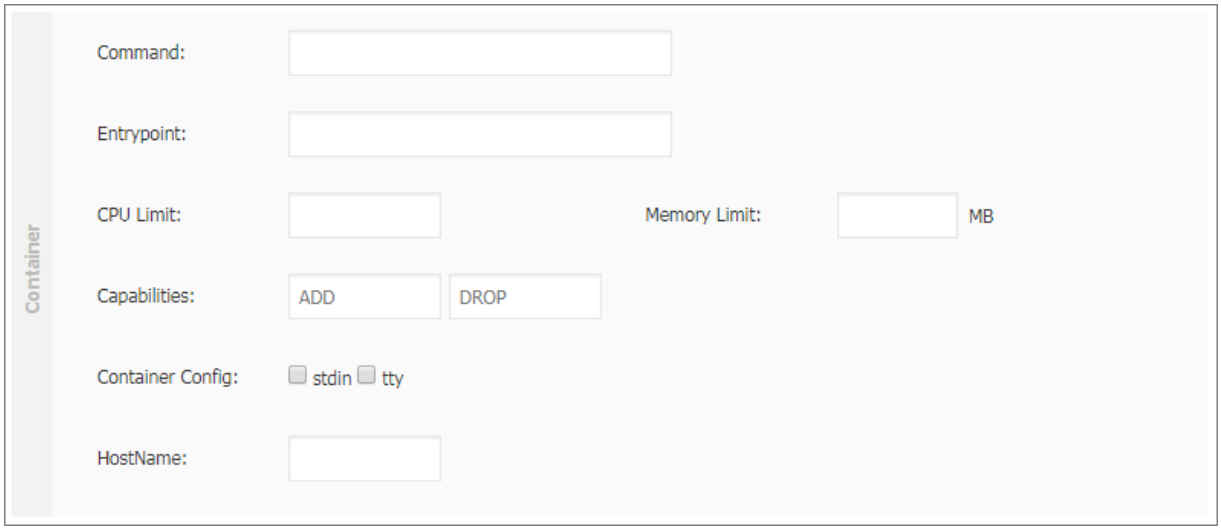
The interfaces for creating an application from an image in a Swarm cluster and in an ACK cluster are quite different.

- For more information about how to create an application from an image in a Swarm cluster, see Create an application.
- For more information about how to create an application from an image in an ACK cluster, see Create a stateless application by using a Deployment.

Container configurations

Swarm cluster

To configure a container, you must set start commands (**Command** and **Entrypoint**), resource limits (**CPU** and **Memory**), and start parameters.



ACK cluster

The **container** configurations in Swarm clusters are similar to the basic configurations and lifecycle configurations of containers in an ACK cluster.

- For more information about the **lifecycle configurations** of a container in an ACK cluster, see Create a stateless application by using a Deployment.
  - **Start**
  - **Post Start**
  - **Pre Stop**

- For more information about how to configure the **basic settings** of a container in an ACK cluster, see Create a stateless application by using a Deployment. For the recommended settings, see Recommended configurations for high reliability.
  - **Resource Limit**
  - **Required Resources**

Labels

In Swarm clusters, labels can be used to implement health checks, domain name settings, and logging.

In ACK clusters, labels are only used to identify applications. When you create an application in an ACK cluster, a label that is named after the application name is automatically generated. This label is not displayed when you create the application from an image. You can configure YAML files to use the label.

11.1.7. Health check and auto scaling configurations for creating an application from an image

This topic describes the differences in health check and auto scaling configurations when you create an application from an image in a Container Service for Swarm cluster and in a Container Service for Kubernetes (ACK) cluster.

Create an application from an image

The interfaces for creating an application from an image in a Swarm cluster and in an ACK cluster are quite different.

- For more information about how to create an application from an image in a Swarm cluster, see Create an application.
- For more information about how to create an application from an image in an ACK cluster, see Create a stateless application by using a Deployment.

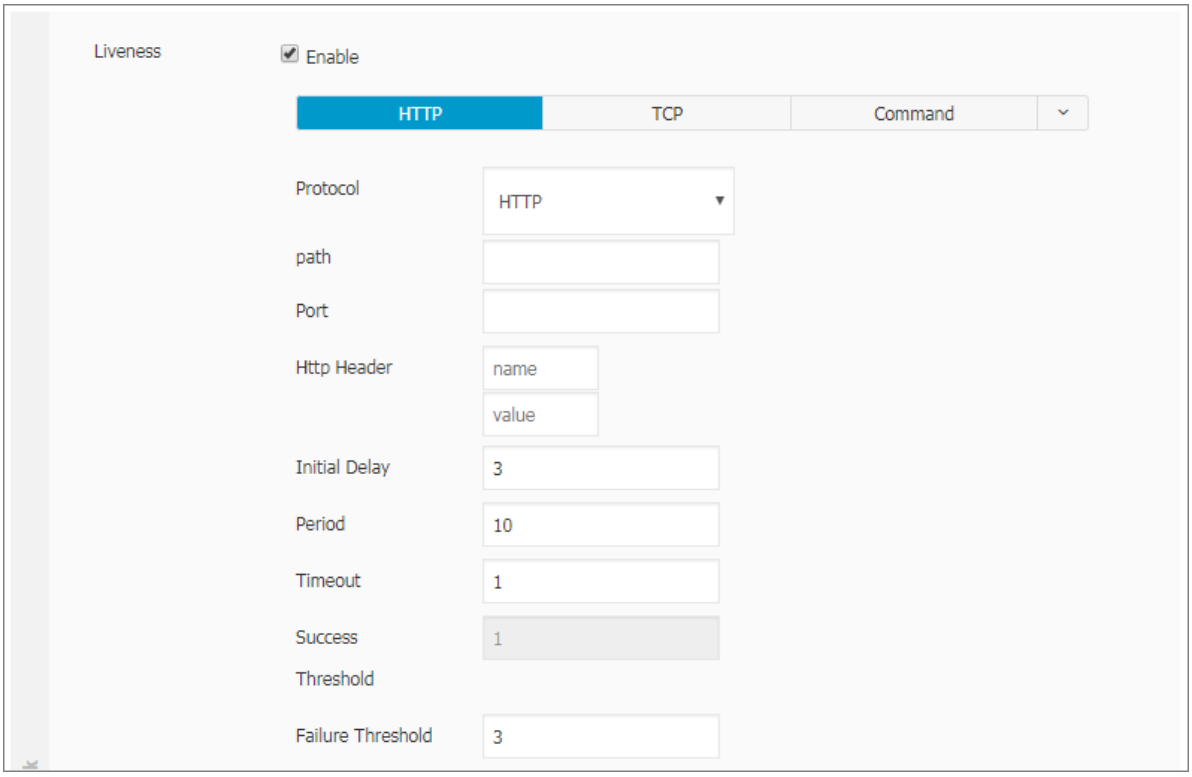
Health check

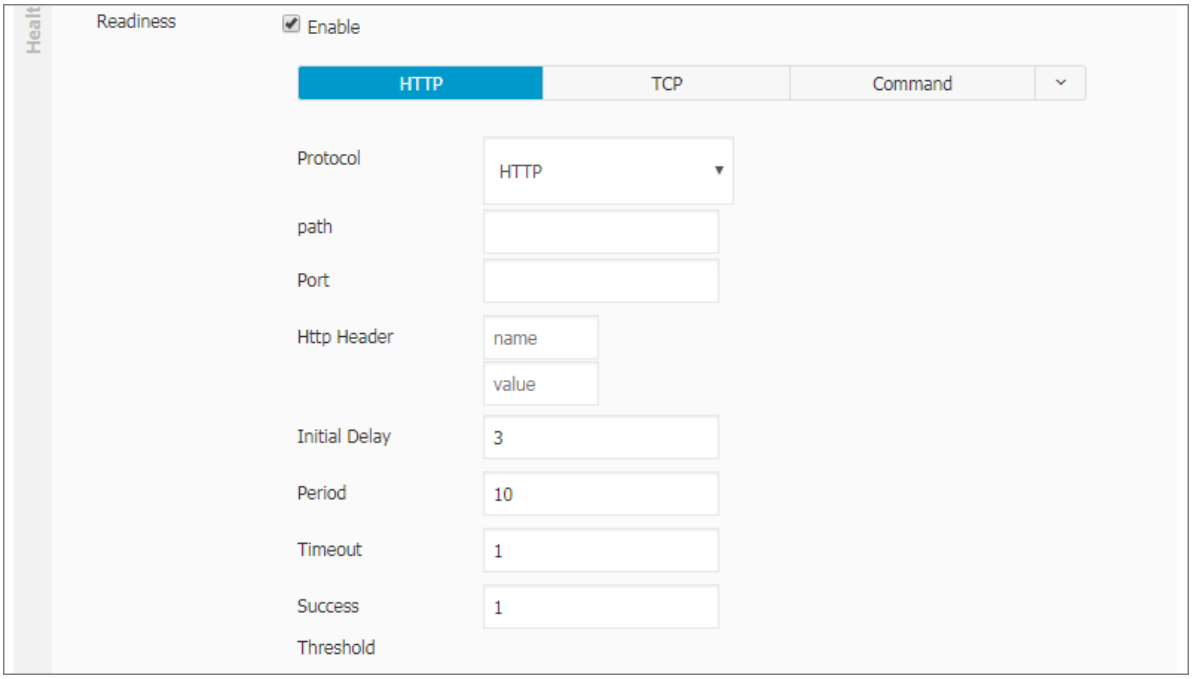
- Swarm cluster

  Health checks are implemented by using labels.

- ACK cluster

You can configure health check settings on the **Container** wizard page. You can configure **liveness** and **readiness** checks.





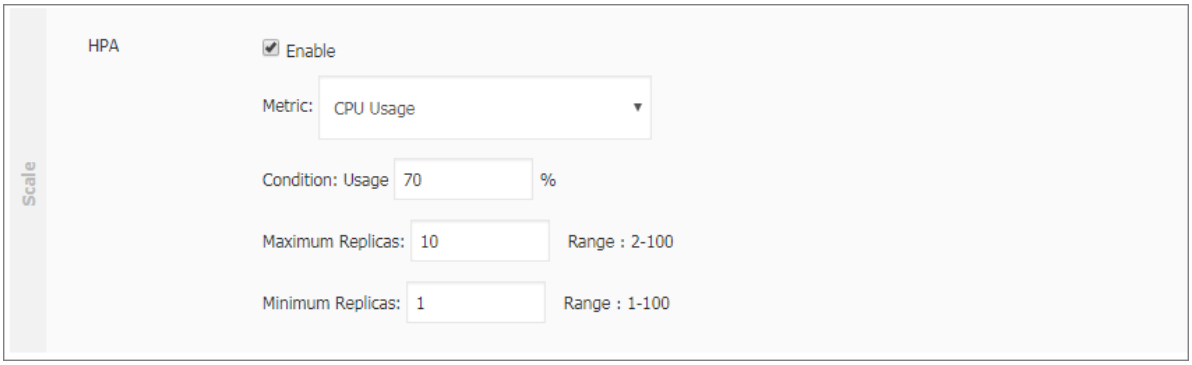
Auto scaling

- Swarm cluster

  Supports auto scaling based on the CPU and memory usage.

- ACK cluster

On the **Advanced** wizard page for creating an application from an image, you can enable horizontal scaling for pods based on the CPU and memory usage.



11.1.8. YAML file configurations

This article describes the comparison of the YAML files that are used to create applications in a Container Service for Swarm cluster and in a Container Service for Kubernetes (ACK) cluster.

Background

The formats of the YAML files that are used to create applications in a Swarm cluster and in an ACK cluster are different.

- You can use Kompose to convert a YAML file for a Swarm cluster into a YAML file for an ACK cluster. After the YAML file is converted, you must check the content of the file.

  You can obtain Kompose from the GitHub website.

  You can download Kompose at one of the following URLs:

  - The Kompose download URL for the macOS operating system: Mac
  - The Kompose download URL for the Linux operating system: Linux
  - The Kompose download URL for the Windows operating system: Window

  > **Note** Kompose does not support specific custom labels in ACK. These custom labels are listed in Labels not supported by Kompose. Alibaba Cloud ACK team will continue its development efforts to add support for all custom labels.

  Labels not supported by Kompose

Label	Related topic
external	External
dns_options	dns_options
oom_kill_disable	oom_kill_disable
affinity:service	Service deployment constraints (affinity:service)

- You can also manually modify a YAML file.

This topic describes the comparison of the YAML files that are used in Swarm and in ACK clusters. The sample YAML files that are used in this topic can be modified based on your business requirements.

YAML file comparison

Swarm cluster

The following template is an example of the *wordpress-swarm.yaml* YAML file that is used in a Swarm cluster. The YAML file is compared with the YAML files that are used in an ACK cluster. You can refer to the numeric marks in the comments for the comparison details.

```
web:        #---1
  image: registry.aliyuncs.com/acs-sample/wordpress:4.5        #---2
  ports:        #---3
    - '80'
  environment:        #---4
    WORDPRESS_AUTH_KEY: changeme            #---5
    WORDPRESS_SECURE_AUTH_KEY: changeme        #---5
    WORDPRESS_LOGGED_IN_KEY: changeme        #---5
    WORDPRESS_NONCE_KEY: changeme            #---5
    WORDPRESS_AUTH_SALT: changeme            #---5
    WORDPRESS_SECURE_AUTH_SALT: changeme        #---5
    WORDPRESS_LOGGED_IN_SALT: changeme        #---5
    WORDPRESS_NONCE_SALT: changeme            #---5
    WORDPRESS_NONCE_AA: changeme            #---5
  restart: always        #---6
  links:            #---7
    - 'db:mysql'
  labels:            #---8
    aliyun.logs: /var/log        #---9
    aliyun.probe.url: http://container/license.txt        #---10
    aliyun.probe.initial_delay_seconds: '10'            #---10
    aliyun.routing.port_80: http://wordpress            #---11
    aliyun.scale: '3'                            #---12
db:             #---1
  image: registry.aliyuncs.com/acs-sample/mysql:5.7        #---2
  environment:        #---4
    MYSQL_ROOT_PASSWORD: password        #---5
  restart: always        #---6
  labels:        #---8
    aliyun.logs: /var/log/mysql        #---9
```

ACK cluster

If you deploy the *wordpress-swarm.yaml* file in a Swarm cluster, a WordPress application is created. In an ACK cluster, a web application and a database application are required to provide the same services as the WordPress application in the Swarm cluster.

To deploy the web and database applications in an ACK cluster, you must create two Deployments and two Services in the ACK cluster. The Services are created to enable access to the Deployments.

You can use the following YAML files to create the Deployment and Service for the web application, which provides the same services as the web service in the Swarm cluster.

> **Note** The following YAML files are only used as examples in comparison with the *wordpress-swarm.yaml* file in Swarm. You must modify the YAML files based on your business requirements before you deploy them.

- The following template is an example of the *wordpress-kubernetes-web-deployment.yaml* file:

```
apiVersion: apps/v1     # The API version.
kind: Deployment        # The type of the resource that you want to create.
metadata:
  name: wordpress      #---1
  labels:            #---8 Labels are only used to identify resources in Kubernetes.
    app: wordpress
spec:    # The resource details.
  replicas: 2        #---12 The number of replicas.
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
strategy:
type: Recreate
  template:   # The pod details.
    metadata:
      labels:  # The same as the previous labels field.
        app: wordpress
        tier: frontend
    spec:    # The container details.
      containers:    #
      - image: wordpress:4   #---2 The image and its version.
        name: wordpress
        env:    #---4 The environment variable. You can reference ConfigMaps and Secrets
through env.
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql  #---7 The name of the MySQL service that is accessed by
the WordPress application.
        - name: WORDPRESS_DB_PASSWORD    #---5 You can use a Secret to store the password
.
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-wordpress
        ports:   #---3 The application port.
        - containerPort: 80
          name: wordpress
livenessProbe:        #add health check    ---10 Enables the health check feature.
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 30
          timeoutSeconds: 5
          periodSeconds: 5
        readinessProbe:       #add health check       ---10 Enables the health check feat
ure.
          httpGet:
```

```
            path: /
            port: 8080
          initialDelaySeconds: 5
          timeoutSeconds: 1
          periodSeconds: 5
        volumeMounts:  # Mounts the volume to the container.
        - name: wordpress-pvc
          mountPath: /var/www/html
      volumes:   # Creates a persistent volume (PV) and a persistent volume claim (PVC) t
o use the volume.
      - name: wordpress-pvc
        persistentVolumeClaim:
          claimName: wordpress-pv-claim
```

- The following template is an example of the *wordpress-kubernetes-web-service.yaml* file:

```
apiVersion: v1   # The API version.
kind: Service    # The type of the resource that you want to create.
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80   # The Service port.
  selector:  # Uses a label to associate the Service with the application.
    app: wordpress
    tier: frontend
  type: LoadBalancer  #---11 The method that is used to access the application. In this e
xample, a LoadBalancer Service is used to automatically create a Server Load Balancer (SL
B) instance.
```

You can use the following YAML files to create the Deployment and Service for the database application, which provides the same services as the database service in the Swarm cluster.

> **Note** The following YAML files are only used as examples in comparison with the *wordpress-swarm.yaml* file in Swarm. You must modify the YAML files before you deploy the YAML files.

- The following template is an example of the *wordpress-kubernetes-db-deployment.yaml* file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-mysql
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: wordpress-mysql-pvc
          mountPath: /var/lib/mysql
      volumes:
      - name: wordpress-mysql-pvc
        persistentVolumeClaim:
          claimName: wordpress-mysql-pv-claim
```

- The following template is an example of the *wordpress-kubernetes-db-service.yaml* file:

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
      clusterIP: None
```

11.1.9. Networks

This topic describes the differences in the networks supported by clusters of Container Service for Swarm and clusters of Container Service for Kubernetes (ACK).

Swarm cluster

Swarm clusters support the following network types:

- Virtual private cloud (VPC)
- Classic network

ACK cluster

ACK clusters support only VPCs. For more information, see Plan CIDR blocks for an ACK cluster.

- If a Swarm cluster is deployed in a VPC, you must select the same VPC when you create an ACK cluster for migration. This ensures that the Swarm cluster and ACK cluster can communicate with each other.
- If a Swarm cluster is deployed in the classic network, you must first connect the classic network to the VPC where you want to deploy the ACK cluster. This is because ACK clusters support only VPCs. For more information, see Overview.

  After the classic network is connected to the VPC, IP addresses for access from VPCs are generated for Alibaba Cloud services that are used by the Swarm cluster, such as Object Storage Service (OSS), Network Attached Storage (NAS), and Relational Database Service (RDS). The created ACK cluster can connect to these IP addresses to access these Alibaba Cloud services through the VPC where the ACK cluster is deployed.

11.1.10. Compare logging and monitoring in Container Service for Swarm and Container Service for Kubernetes

This topic compares the logging and monitoring features of a Container Service for Swarm cluster and those of a Container Service for Kubernetes (ACK) cluster.

Logging

Container Service for Swarm cluster

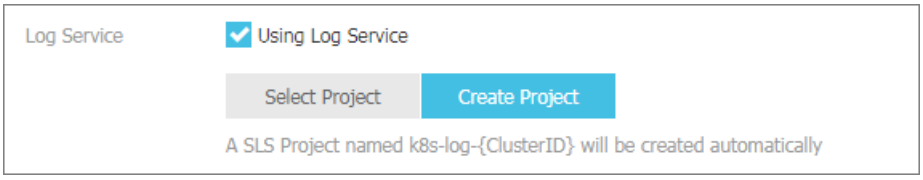
A Container Service for Swarm cluster implements the logging feature based on **labels**.

ACK cluster

For an ACK cluster, the logging feature is configured and implemented in the following way:

- Create an ACK cluster:

  On the **Create Kubernetes Cluster** page, select **Enable Log Service**. Then, the Log Service plug-in is automatically configured for the cluster. You can use an existing project or create a project.

  You can also manually install the Log Service plug-in after an ACK cluster is created. For more information, see Collect log data from containers by using Log Service.

- Configure Log Service when you create an application in the cluster. For more information, see Collect log data from containers by using Log Service.
- Use Log Service after you create an application in the cluster. For more information, see Use the Log Service console to collect container text logs in DaemonSet mode and Use the Log Service console to collect container stdout and stderr in DaemonSet mode.

Monitoring

To enable monitoring for a Container Service for Swarm cluster or an ACK cluster, select **Install CloudMonitor Agent on ECS Instance** on the **Create Kubernetes Cluster** page. Then, you can view the monitoring data of the created Elastic Compute Service (ECS) instances in the Cloud Monitor console.

For more information about how ACK clusters integrate the Cloud Monitor service, see Monitor basic resources.

11.1.11. Access applications by using different methods

This topic describes the methods that are used to access applications in a Container Service for Swarm cluster or Container Service for Kubernetes (ACK) cluster. The methods allow an application to access other applications within the same cluster or in other clusters.

Access applications within the same cluster

Container Service for Swarm cluster

To access a service within the same Container Service for Swarm cluster, you can use the `links` label to specify the name of the service as an environment variable.

For example, as described in YAML files used for creating applications, web services provided by a WordPress application are associated with a *MySQL* service. You can access the MySQL service by specifying its name *mysql* after the container is started.

```
links:            #---7
    - 'db:mysql'
```

ACK cluster

To access a service within the same ACK cluster, you can use the cluster IP or name of the service. We recommend that you use service names to access applications within the cluster.

When you create an application, you can specify the name of the service that you want to access as an environment variable.

For example, as described in YAML files used for creating applications, a WordPress application uses an environment variable to call a *MySQL* service.

```
spec:
      containers:
      - image: wordpress:4
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql    #---7 Specify the name of the MySQL service that you wa
nt to access. The name is mysql in this example.
        - name: WORDPRESS_DB_PASSWORD
```

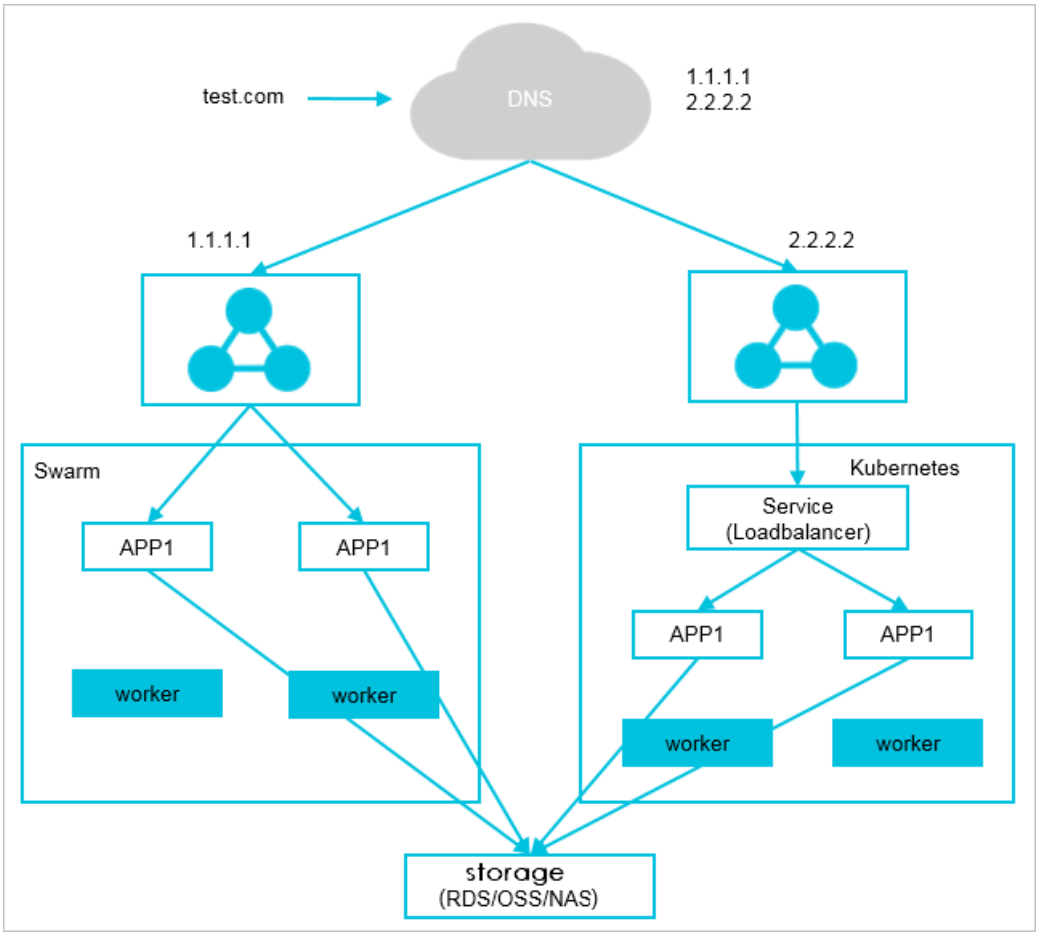
Access applications in other clusters

Access an application by using a domain name

> **Note**
>
> - Ensure the network connectivity to the classic network or virtual private cloud (VPC).
> - The DNS server supports load balancing and can distribute traffic to different backend IP addresses.
> - Applications that are accessed through domain names can be migrated from Swarm to Kubernetes without service disruption.

Simple routing (use a domain name bound to the default SLB instance of a Container Service for Swarm cluster)

To migrate an application from a Container Service for Swarm cluster to an ACK cluster, you must first create an application in the ACK cluster. You must verify the functions of the application before you migrate the Swarm application to the ACK cluster.



Migration scheme

- Perform the following operations to create an application in the ACK cluster:
  - In the ACK cluster, create an application of the same type as the application that you want to migrate from a Container Service for Swarm cluster.
  - Create a Loadbalancer service for the application in the ACK cluster.
  - The Loadbalancer service automatically creates a Server Load Balancer (SLB) instance. The IP address of the SLB instance is 2.2.2.2 in this example.
  - Modify DNS records to add the IP address 2.2.2.2 as a backend IP address for the domain name *test.com*.
- Verify that the application in the ACK cluster is available

  The application is available when it is accessed through the IP address 2.2.2.2.
- Switch traffic to the Kubernetes application

  Modify DNS records to remove the backend IP address 1.1.1.1 for the domain name *test.com*.

All traffic destined for the application in the Container Service for Swarm cluster is forwarded to the Kubernetes application through DNS.

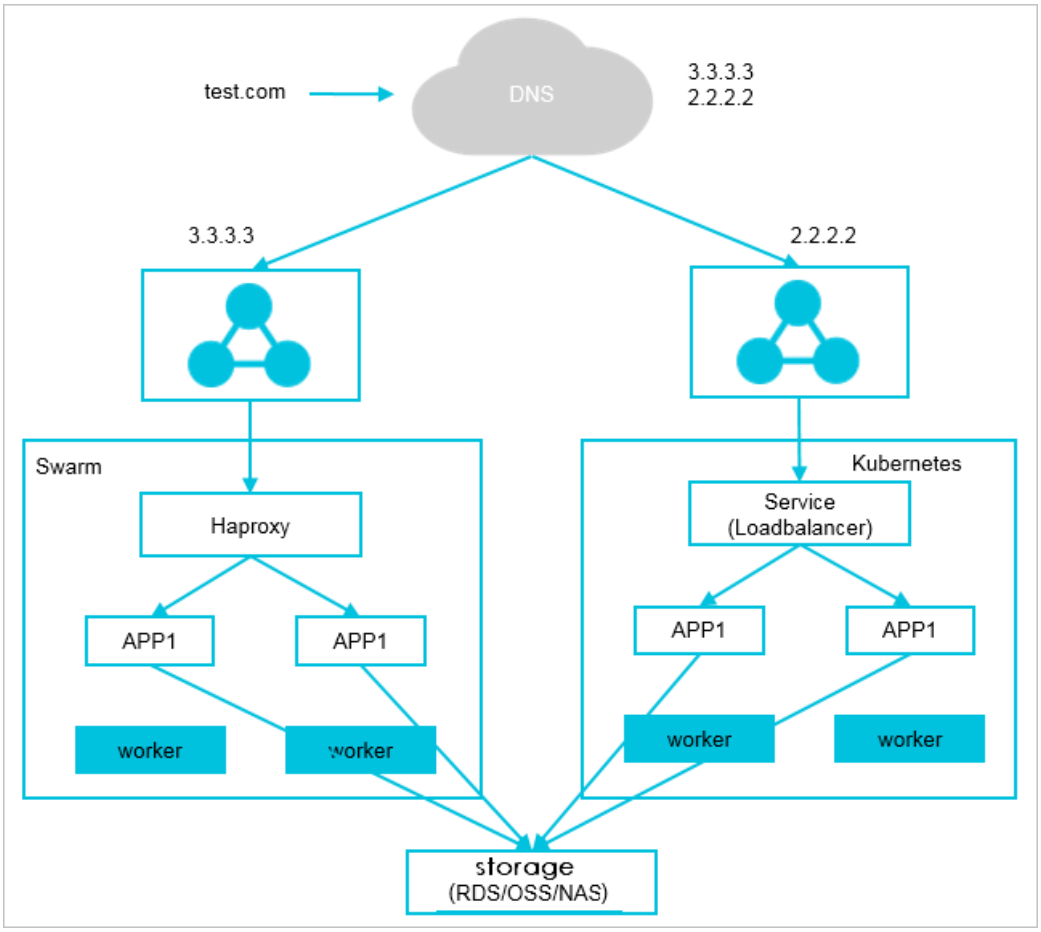
Simple routing (use a domain name bound to a self-managed SLB instance of a Container Service for Swarm cluster)

You can bind a domain name to the default SLB instance or a self-managed SLB instance attached to a Container Service for Swarm cluster. The two methods differ in the following ways:

- The SLB instances are different.
- By default, Alibaba Cloud DNS is used. To use a custom domain name, you must manually resolve the domain name.

Migration scheme

The procedure is the same as the procedure to migrate an application from Swarm to Kubernetes by using the default SLB instance. You must create an application in the ACK cluster and verify the functions of the application before you switch all traffic to the Kubernetes application.



Access through <HostIP>:<port>

If an application is accessed through <HostIP>:<port>, you cannot migrate the application from a Container Service for Swarm cluster to an ACK cluster without service disruption. We recommend that you migrate the application during off-peak hours.

Migration scheme

1. Create an application in the ACK cluster and use a NodePort service to expose the application. For more information, see Network configurations for deploying an application from an image.
2. Record the NodePort and replace the <port> of the Container Service for Swarm cluster with the <NodePort> of the ACK cluster.

   > **Note** To perform this step, you must stop and modify applications one after another.

3. Attach worker nodes of the ACK cluster to the SLB instance that is attached to the Container Service for Swarm cluster.

4. The SLB instance forwards a percentage of traffic to the Kubernetes application. After you verify the functions of this application, you can remove the nodes of the Container Service for Swarm cluster from the SLB instance. This switches all traffic to the Kubernetes application.

Access through an SLB instance

If an application is accessed through an SLB instance, you cannot migrate the application from a Container Service for Swarm cluster to an ACK cluster without service disruption. We recommend that you migrate the application during off-peak hours.

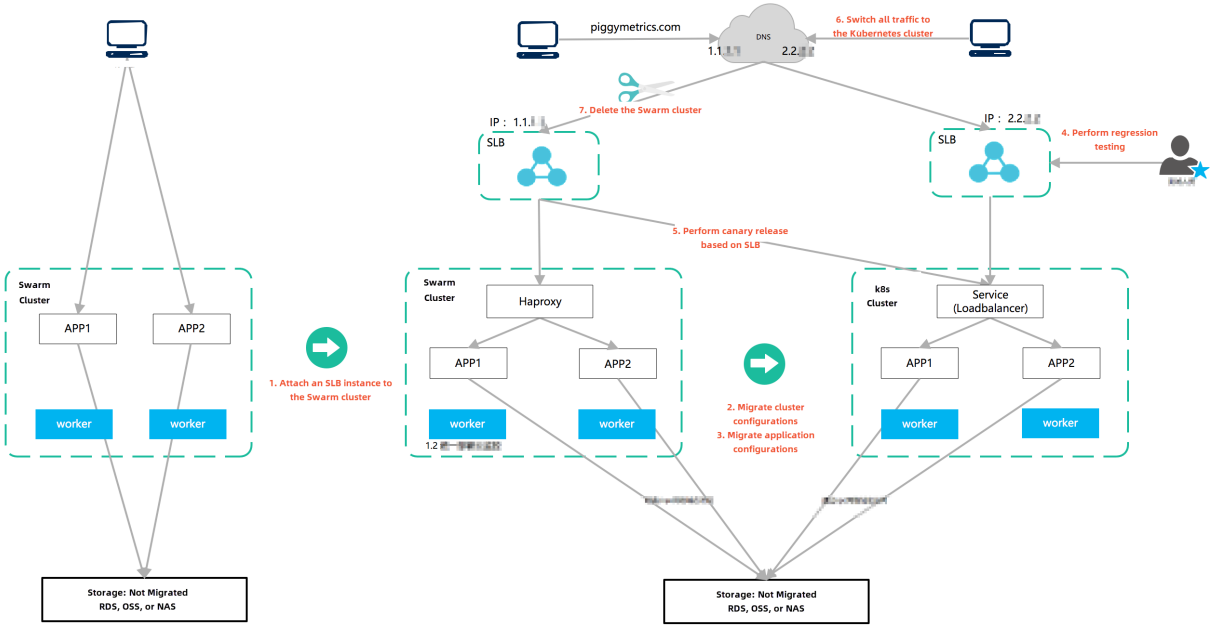
Migration scheme

You can use LoadBalancer services in the ACK cluster in the same way as you use SLB instances in the Container Service for Swarm cluster. For more information, see Network configurations for deploying an application from an image.

11.2. Overview

This topic describes how to migrate services from a Container Service for Swarm cluster to a Container Service for Kubernetes (ACK) cluster and avoid service disruption with best efforts. Seven steps are required to complete the migration.

Migration scheme



Procedure

1. Standardize the Container Service for Swarm cluster.

   O&M engineers perform O&M operations in the Container Service for Swarm cluster to reduce migration costs and risks.

i. Clients access applications in the Container Service for Swarm cluster through a Server Load Balancer (SLB) instance. This enables you to **route a percentage of production traffic to the ACK cluster** in real time.

- If you access applications through an SLB instance, you can route a percentage of production traffic to the ACK cluster to verify that the migration is successful. This also allows you to roll back to Swarm when an error occurs. Changes in the SLB console take effect immediately.
- If you use other methods to access applications, it may take 0 to 48 hours to update the client configurations or roll back DNS records. This poses negative impacts on your services.

ii. Deploy Cloud Monitor in the Container Service for Swarm cluster to monitor the Elastic Compute Service (ECS) instances running in the cluster and make sure that traffic is routed to the ACK cluster.

2. Configure clusters for migration.

   O&M engineers create an ACK cluster and configure cluster resources. This reduces the workloads of developers during the migration process.

   i. Migrate servers and network resources.
   ii. Migrate node tags.
   iii. Verify the connectivity to the virtual private cloud (VPC) where the cluster is deployed.
   iv. Migrate volumes.
   v. Migrate configurations.

3. Migrate application configurations.

   Developers use Kompose to migrate application configurations to the ACK cluster.

   i. Set up an environment to complete the migration task.
   ii. Preprocess Swarm Compose files.
   iii. Convert the Swarm Compose files to Kubernetes resource files.
   iv. Deploy the Kubernetes resource files.
   v. Manually migrate application configurations.
   vi. Debug application startup.
   vii. Migrate logging configurations of applications.

4. Perform regression testing for the migrated applications.

   Test engineers perform regression testing to validate the functions of the migrated applications without affecting the application performance.

   i. Configure test domain names for the applications.
   ii. Test application functions.
   iii. Verify that application log data can be collected.
   iv. Verify that the applications are monitored.

5. Route traffic destined for the Container Service for Swarm cluster to the ACK cluster.

   O&M engineers route traffic to the ACK cluster through canary releases. They can roll back to the Container Service for Swarm cluster when an error occurs.

   i. Route traffic to the NodePort service.

ii. Roll back to the Container Service for Swarm cluster if necessary.

6. Switch all production traffic to the ACK cluster.

   O&M engineers switch all production traffic to the ACK cluster by changing DNS settings or upgrading clients.

   i. Update DNS records to switch traffic.

   ii. Upgrade client code or configurations to switch traffic.

7. Delete the Container Service for Swarm cluster.

   After O&M engineers verify that traffic is routed to the ACK cluster, delete the Container Service for Swarm cluster.

   i. Verify that no traffic is routed to the Container Service for Swarm cluster.

   ii. Delete the Container Service for Swarm cluster and release its resources.

11.3. Attach an SLB instance to a Swarm cluster

Context

If applications in a cluster of Container Service for Swarm are accessed through *<HostIP>:<port>* or node IP addresses resolved from domain names certified by ICP filing, you cannot migrate the applications to Kubernetes without service disruption. In addition, you cannot roll back the migration in real time when an error occurs. To fix the issue, you must attach a Server Load Balancer (SLB) instance to the Swarm cluster so that you can manage access to applications in the Swarm cluster. Then, you can install the Cloud Monitor agent to monitor the Swarm cluster. This way, you can migrate applications without service disruption and roll back the migration when an error occurs.

Access a Swarm cluster through an SLB instance

1. Create an SLB instance.
   - Automatically create an SLB instance

     If you have selected **Automatically Create SLB Instances** when you create the Swarm cluster, a pay-as-you-go SLB instance is automatically created.

     You can log on to the SLB console to view the created SLB instance. Make sure that the SLB instance has a listener that listens on TCP port 9080 for the attached backend servers.
   - Manually create an SLB instance

     If you do not select **Automatically Create SLB Instances** when you create the cluster, you must manually create an SLB instance and attach it to the Swarm cluster. Perform the following steps:

     a. Create an SLB instance. For more information, see Create a CLB instance.

     b. Attach the SLB instance to the Swarm cluster. This ensures that the Swarm cluster automatically updates the SLB configurations. For more information, see Bind and unbind a Server Load Balancer instance.
2. Configure simple routing.

If applications are accessed through *<HostIP>:<port>*, the host IP addresses are configured in the client. If cluster nodes are replaced or new nodes are added to the cluster, you must upgrade the client to access applications deployed on new nodes. Therefore, we recommend that you configure domain names for applications and allow access to applications through domain names. We recommend that you access applications through domain names.

> **Note** If you do not have custom domain names, Container Service for Kubernetes (ACK) allows you to configure a separate domain name for each application or service for free. The domain name provided by ACK is in the format of *XXX.$cluster_id.$region_id.alicontainer.com*. You can customize *XXX* based on your applications or services.

This allows users to access applications through custom domain names or testing domain names that are provided by ACK. The *<HostIP>:<port>* of Elastic Compute Service (ECS) instances or SLB instances are no longer required to access applications. For more information, see Simple routing - supports HTTP and HTTPS.

3. Test access to an application through the SLB instance.

   Use the domain name configured in the previous step to access your application, test whether the application runs as expected, and check whether the SLB instance has received traffic.

   i. Access your application through the SLB instance.

   ii. Check whether the SLB instance has received traffic.

4. Change the address that receives inbound traffic.

   - Change DNS records: If your application was accessed through a domain name, change the IP address to which the domain name points from the node IP address to the IP address of the SLB instance at your DNS service provider.

     > **Note** DNS servers at all levels cache DNS records. After you change DNS records, you must monitor the traffic that flows through the SLB instance to check whether the change takes effect.

   - Change client configurations: If your application was accessed through a node IP address, upgrade the client to allow access to the application only through domain names.

Install the Cloud Monitor agent on ECS instances of the Swarm cluster (optional)

If you do not select **Install CloudMonitor Agent on ECS Nodes** when you create the Swarm cluster, the Cloud Monitor agent is not installed on the ECS instances of the Swarm cluster. As a result, you cannot view the monitoring information about the ECS instances in the Cloud Monitor console during cluster migration. For example, when you bring the Swarm cluster offline, you cannot use Cloud Monitor to check whether the ECS instances are still receiving traffic. This section describes how to view the monitoring information about the ECS instances and manually install the Cloud Monitor agent if it is not installed.

1. View the monitoring information of ECS instances in the Swarm cluster.

   i. Log on to the Container Service for Swarm console. In the left-side navigation pane, click **Nodes**. On the Nodes page, find the node that you want to check and click **Monitor** in the Actions column.

ii. In the left-side navigation pane of the page that appears, click **Monitoring**. In the **Monitoring Information** section, you can view the detailed monitoring information about the node in different time ranges.

If the Cloud Monitor agent is not installed, you cannot view the detailed monitoring information about the ECS instances, such as memory usage, disk usage, and TCP connections. If the Cloud Monitor agent is not installed, you can manually install the Cloud Monitor agent as described in the next step.

2. Manually install the Cloud Monitor agent on ECS instances in the Swarm cluster.
   i. Log on to the Cloud Monitor console. In the left-side navigation pane, choose **Dashboard > Host Monitoring**. On the page that appears, click the **Instances** tab, select the ECS instances, and click **Batch Install** to install the Cloud Monitor agent on the selected ECS instances.
   ii. After the Cloud Monitor agent is installed, choose **Dashboard > Host Monitoring**. Click the name of an ECS instance. In the **Network Metric** section, you can view the curve about the inbound and outbound traffic for the ECS instance.

11.4. Migrate cluster configurations

This topic uses a cluster of Container Service for Swarm named swarm-piggymetrics-cluster as an example to describe how to migrate cluster configurations from Swarm to Kubernetes. The following steps are involved: create a Kubernetes cluster, migrate node labels, check whether the Kubernetes cluster can access other Alibaba Cloud services, migrate volumes, and migrate configuration items.

Prerequisites

The Swarm cluster to be migrated is attached with a Server Load Balancer (SLB) instance. For more information, see Attach an SLB instance to a Swarm cluster.

Context

This topic uses cluster swarm-piggymetrics-cluster as an example to describe how to migrate cluster configurations from Swarm to Kubernetes. The configurations migration process includes the following steps: create a Kubernetes cluster, migrate node labels, check whether the Kubernetes cluster can access other Alibaba Cloud services over a virtual private cloud (VPC), migrate volumes, and migrate configuration items.

Create a Kubernetes cluster

Create a Kubernetes cluster based on the configurations of cluster swarm-piggymetrics-cluster. For more information, see Create an ACK managed cluster. When you create the Kubernetes cluster, take note of the following limits:

- Select the same region and zone where cluster swarm-piggymetrics-cluster is deployed. This ensures successful migration of resources that cannot be migrated across regions, such as virtual private clouds (VPCs)
- Kubernetes clusters support only VPCs and do not support the classic network.
  - If cluster swarm-piggymetrics-cluster is deployed in a VPC, you must select the same VPC when you create the managed Kubernetes cluster.
  - If cluster swarm-piggymetrics-cluster is deployed in the classic network, you must migrate the cluster to a VPC. For more information, see Hybrid migration.

- When you create the managed Kubernetes cluster, you must select Install the CloudMonitor Agent on ECS Nodes. This allows you to view the monitoring information of the Elastic Compute Service (ECS) instances in the Cloud Monitor console after you switch traffic to the created Kubernetes cluster or perform canary releases in the created Kubernetes cluster.
- If cluster swarm-piggymetrics-cluster uses Log Service, we recommend that you select **Enable Log Service** and the Log Service project that is used by the Swarm cluster. This way, you do not need to migrate Log Service configurations.
- If cluster swarm-piggymetrics-cluster uses a Relational Database Service (RDS) instance, we recommend that you select the RDS instance to add the ECS instances of the managed Kubernetes cluster to the whitelist of the RDS instance. This ensures that the Kubernetes cluster can access the RDS instance.

Migrate node labels

If nodes in cluster swarm-piggymetrics-cluster are added with custom labels that set rules for node deployment, you must migrate these custom labels. To migrate custom labels, perform the following steps:

1. View the custom labels of nodes in the Swarm cluster.
   i. Log on to the Container Service for Swarm console. In the left-side navigation pane, click **Clusters**. On the Clusters page, find cluster swarm-piggymetrics-cluster and click **Manage** in the Actions column.
   ii. In the left-side navigation pane, click **Custom Labels**. On the page that appears, you can find the custom labels that are added to the nodes in cluster swarm-piggymetrics-cluster.
2. Add the same labels to the related nodes in the created managed Kubernetes cluster.
   i. Log on to the Container Service for Kubernetes (ACK) console. In the left-side navigation pane, choose **ClusterS > Nodes**. In the upper right corner of the **Nodes** page, click **Manage Labels**.
   ii. On the **Manage Labels** page, select a node and click **Add Label**. In the Add dialog box, enter the label name and value and click **OK**.

Check whether the created Kubernetes cluster can access other Alibaba Cloud services over a VPC

If cluster swarm-piggymetrics-cluster is deployed in the classic network, the created managed Kubernetes cluster that is deployed in a VPC may fail to access other Alibaba Cloud services such as RDS and Object Storage Service (OSS). For example, only ECS instances in the RDS whitelist can access the related RDS instance. The endpoint of an OSS bucket for the classic network is different from that for a VPC. Therefore, after you complete the preceding steps, you must check whether the created Kubernetes cluster can access other Alibaba Cloud services over a VPC.

1. Log on to the RDS console. On the Instances page, click the related RDS instance. In the left-side navigation pane of the details page, click **Data security**. In the **Whitelist Settings** section, check whether the IP addresses of the ECS instances in the created managed Kubernetes cluster are added to the whitelist of the RDS instance.
2. In the left-side navigation pane, click **Database Connection**. On the **Instance Connection** tab, you can obtain the value of the Internal Endpoint parameter, which is the endpoint of the RDS instance for VPCs.
3. Log on to the ECS console. Then, follow the instructions to check whether the ECS instance can connect to the RDS instance over the VPC.

- If the ECS instances can access the RDS instance, go to the next step.
- If the ECS instances cannot access the RDS instance, the possible reasons include:
  - The IP addresses of the ECS instances failed to be added to the whitelist of the RDS instance. The IP addresses of the ECS instances in the created Kubernetes cluster failed to be added to the whitelist of the RDS instance. Log on to the RDS console, choose **RDS Instance > Data Security**, and add the IP addresses of the ECS instances to the RDS whitelist again. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
  - If the issue still exists, see What do I do if I cannot connect an ECS instance to an ApsaraDB for RDS instance?.

  After you troubleshoot and fix the connection issue, log on to the ECS console again. Make sure that the ECS instances can access the RDS instance. Then, go to the next step.

4. Log on to the OSS console. Find the OSS bucket that is used by the Swarm cluster and view the endpoints for the classic network and VPCs.

If cluster swarm-piggymetrics-cluster uses the endpoint for the classic network to access the OSS bucket, you must configure the Swarm cluster to access the OSS bucket by using the endpoint for VPCs. This endpoint setting is also required in the next step when you migrate volumes.

Migrate volumes

In Swarm and Kubernetes, volumes are mounted to clusters. You can migrate volume configurations when you migrate cluster configurations. You can specify how to use the volumes when you migrate application configurations. Volumes in Swarm clusters correspond to persistent volumes (PVs) and persistent volume claims (PVCs) in Kubernetes clusters.

Swarm supports the following types of volumes: Network Attached Storage (NAS), cloud disk, and OSS. You can use them in Kubernetes **directly as volumes** or **through PVs and PVCs**. For more information, see Storage management overview. This section describes how to use volumes **through PVs or PVCs**.

1. Log on to the Container Service for Swarm console. In the left-side navigation pane, click **Volumes**. On the **Data Volume List** page, select cluster *swarm-piggymetrics-cluster* and find volumes of the NAS, disk, and OSS types.
2. In the created managed Kubernetes cluster, create PVs and PVCs for the three types of volumes that are mounted to the Swarm cluster. For more information, see the following topics:
   - Disk volume overview
   - NAS volume overview
   - OSS volume overview

Notes on volume migration

- A disk is a non-shared storage provided by Alibaba Cloud and can be mounted to only one pod at a time. If the Swarm and Kubernetes clusters write data to a disk at the same time, an error will occur. This means that you cannot migrate disks from Swarm to Kubernetes without service interruptions. We recommend that you use NAS or OSS for shared storage in Swarm clusters or migrate disks during off-peak hours. You must detach a disk from the Swarm cluster before you can mount it to the Kubernetes cluster. For more information, see Detach a data disk and View and delete data volumes.
- In Kubernetes clusters, the name of a PV or a PVC cannot contain uppercase letters or underscores (_). If the name of a volume in the Swarm cluster contains uppercase letters or underscores (_), convert the name based on the following rules:

- Change uppercase letters to lowercase letters.
- Change underscores (_) to hyphens (-).

- When you create a PV of the OSS type, set Endpoint to *VPC Endpoint* because Kubernetes supports only VPCs.
- When you create a PVC, convert the name of the related volume based on the preceding rules and use the result as the name of the PVC. When you use Kompose to convert a Swarm compose file to Kubernetes resource files, PVC names are generated in the same way.

Migrate configuration items

In Swarm clusters, you can create a configuration file and set configuration items to manage environment variables for multiple containers in a unified manner. When you deploy an application, you can use the configuration file to replace variables that start with a dollar sign ($) in the Swarm compose file with the actual values.

This is an advanced feature provided by the Container Service for Swarm console. ACK does not support this feature. The configuration files in the Swarm cluster are different from ConfigMaps in the Kubernetes cluster. This means that the configuration items in Swarm cannot be automatically migrated to Kubernetes. You must manually replace the variables in the Swarm compose file with the actual values. For more information, see Overview.

11.5. Migrate application configurations

11.5.1. Overview

This topic describes how to migrate application configurations from Swarm to Kubernetes after operations and maintenance (O&M) engineers have migrated cluster configurations. The following steps are required: set up the environment, pre-process and convert Swarm compose files, deploy Kubernetes resource files, manually migrate application configurations that cannot be automatically converted, and debug the application to fix potential issues.

Comparison between reserved instances, pay-as-you-go instances, and subscription instances

Swarm and Kubernetes both use various concepts in terms of applications, services, and access methods. For more information about the differences between these concepts, see Basic terms.

Prerequisites

An application is required to be migrated from Swarm to Kubernetes. A Kubernetes cluster with the same cluster configurations is created for the application. In this topic, an application named swarm-piggymetrics is used as an example and its configurations are migrated to the Kubernetes cluster named k8s-piggymetrics-cluster.

Based on PiggyMetrics, swarm-piggymetrics is an application built on a microservice architecture. PiggyMetrics is a Spring Cloud project on GitHub.

Procedure

1. Set up the environment.

2. Pre-process Swarm compose files.
3. Convert Swarm compose files.
4. Deploy Kubernetes resource files.
5. Manually migrate application configurations.
6. Debug application startup.
7. Migrate log configurations of applications.

Related topics

For more information about migration-related questions, see Troubleshooting.

For more information about the mapping of labels, see Application configuration labels, Application release labels, Network configuration labels, and Log configuration parameters.

11.5.2. Set up the environment

Before you migrate application configurations from a Swarm cluster to a Kubernetes cluster, you must set up the environment for migration. This topic describes how to set up the environment for migration.

Context

Kompose is an open source tool that is used to convert Swarm compose files into Kubernetes resource files. Alibaba Cloud has optimized Kompose to support labels that are specific to Alibaba Cloud.

After Kubernetes resource files are generated, you can Connect to ACK clusters by using kubectl and deploy these files in the Kubernetes cluster.

Kompose and kubectl are required to migrate application configurations. We recommend that you install Kompose and kubectl on a dedicated Elastic Compute Service (ECS) instance.

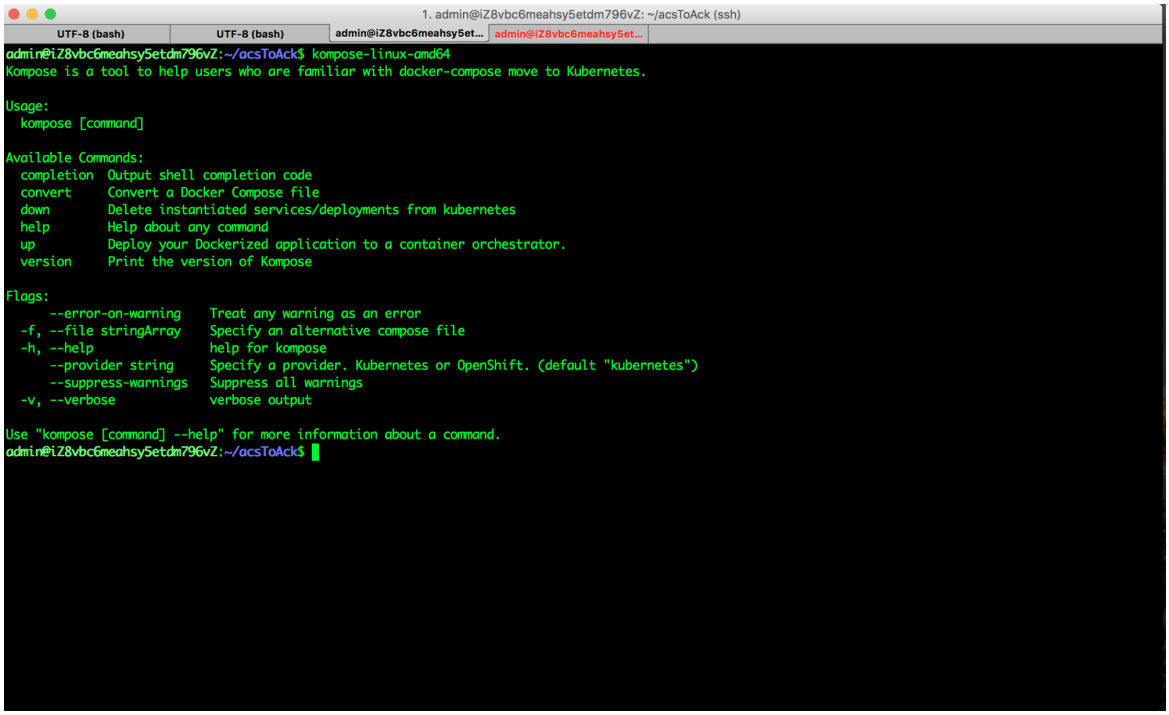
Procedure

1. Install Kompose.

   Kompose is a tool that is used to convert Swarm compose files into Kubernetes resource files. Alibaba Cloud has optimized Kompose to support labels that are specific to Alibaba Cloud. For more information about Kompose, see AliyunContainerService/kompose.

   Installation: Download the latest executable file AliyunContainerService/kompose/releases from GitHub based on your operating system.

   ```
   kompose-linux-amd64
   ```

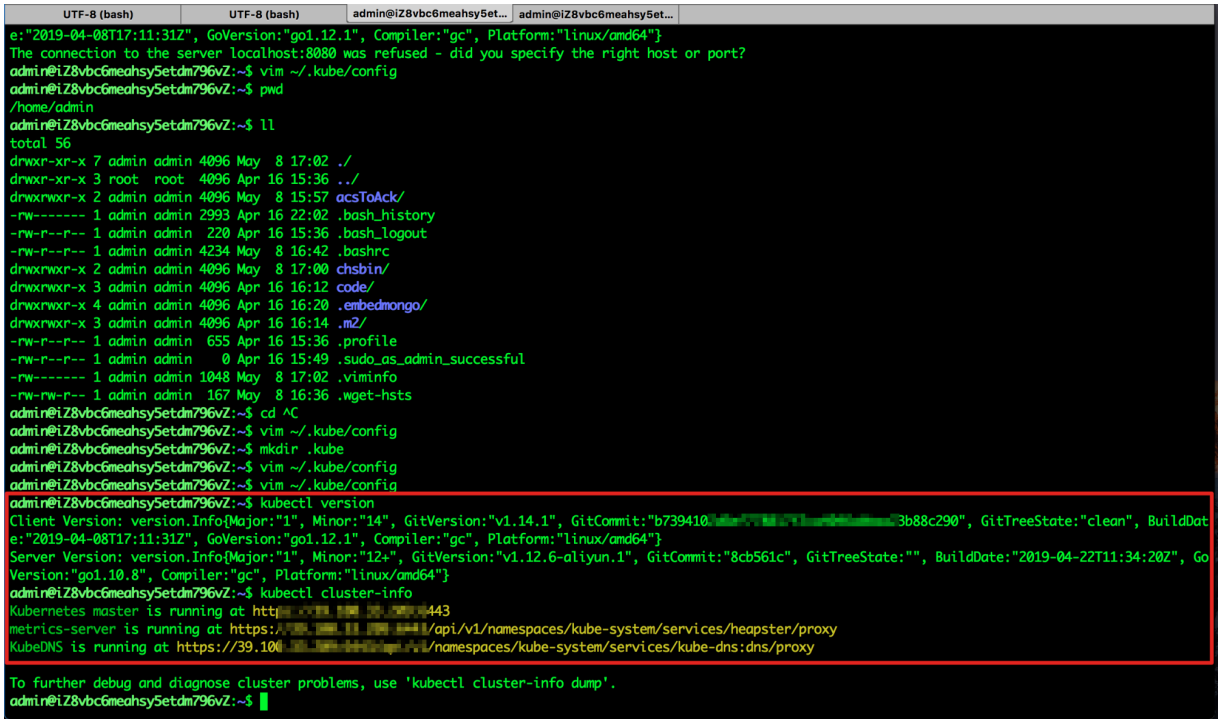


2. Set up the environment for kubectl.
   i. Download the latest kubectl client from the Kubernetes changelog page.
   ii. Install and set up the kubectl client. For more information, see Install and set up kubectl.
   iii. Configure cluster credentials.
      a. Log on to the Container Service for Kubernetes (ACK) console.
      b. In the left-side navigation pane, click **Clusters**.
      c. On the **Clusters** page, find the cluster that you want to manage and click the name or click **Details** in the **Actions** column.
      d. On the **Basic Information** tab, copy the KubeConfig content to *$HOME/.kube/config* on your on-premises machine.
   iv. After the setup is completed, run the following commands to check whether the installation is successful and cluster credentials are correctly configured.

```
kubectl version
kubectl cluster-info
```

Result

If the output is as shown in the following red box, it indicates that kubectl is installed and the migration environment is set up.



11.5.3. Pre-process Swarm compose files

This topic describes how to pre-process Swarm Compose files.

Procedure

1. Download a Swarm Compose file.
   i. Log on to the Container Service for Swarm console. In the left-side navigation pane, click **Applications**. On the Applications page, find the application that you want to migrate and click **Change Configuration** in the Actions column.
   ii. Copy the configurations and save them in a *YAML* file on the on-premises machine. In this topic, the file is named *swarm-piggymetrics.yaml*, which is the Swarm Compose file that you can pre-process.
2. Replace environment variables in the Swarm compose file with actual values.

   The Container Service for Swarm console allows you to use a configuration file to replace variables in a Swarm compose file with actual values. Kubernetes does not support this feature. Before you use Kompose to convert a Swarm compose file, you must manually replace the dollar signs (*$*) in the compose file with actual values. To replace variables in the Swarm compose file, perform the following steps:
   i. Log on to the Container Service for Swarm console. In the left-side navigation pane, click **Configurations**. On the page that appears, select the region where the configuration file is deployed and find the configuration file.
   ii. Replace the dollar signs (*$*) in the *swarm-piggymetrics.yaml* file with the actual values based on this configuration file.
3. Pre-process labels.

   You must pre-process the labels in the Swarm compose file. Otherwise, Kompose may report an error and stop the conversion as a result of invalid formats or unsupported labels. The following

text describes the two major methods of label processing. For more information about each label, see Application configuration labels, Application release labels, Network configuration labels, and Log configuration parameters.

- Change the values of the following labels from BOOLEAN to STRING. For example, change true or false to `"true"` or `"false"`.
  - aliyun.global
  - aliyun.latest_image
- Kubernetes Services with labels that contain the following value cannot be migrated. You must delete them and create new ones with the same configurations in the related Kubernetes cluster.
  - external

4. Modify the version number of the Swarm compose file.

   Kompose supports Swarm compose files of version 2. You must change the version declaration from version: '2.X' to version: '2' in the Swarm compose file. Otherwise, an error will occur during conversion.

11.5.4. Convert Swarm compose files

This topic describes how to convert Swarm compose files.

Procedure

1. Use Kompose to convert a Swarm compose file.

   After a Swarm compose file is pre-processed, run the following command to convert the file by using Kompose:

```
kompose-linux-amd64 convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolumeClaimOrHostPath
```

```
admin@iZ8vbc6meahsy5etdm796vZ:~/acsToAck$ kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolumeClaimOrHostPath
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported aliyun.routing.session_sticky label key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported aliyun.depends label key - ignoring
WARN Unsupported links key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported links key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported logging key - ignoring
WARN Unsupported devices key - ignoring
INFO Kubernetes file "gateway-service.yaml" created
INFO Kubernetes file "monitoring-service.yaml" created
INFO Kubernetes file "rabbitmq-service.yaml" created
INFO Kubernetes file "registry-service.yaml" created
INFO Kubernetes file "turbine-stream-service-service.yaml" created
INFO Kubernetes file "account-mongodb-deployment.yaml" created
INFO Kubernetes file "account-service-deployment.yaml" created
INFO Kubernetes file "auth-mongodb-deployment.yaml" created
INFO Kubernetes file "auth-service-deployment.yaml" created
INFO Kubernetes file "config-deployment.yaml" created
INFO Kubernetes file "gateway-deployment.yaml" created
INFO Kubernetes file "monitoring-deployment.yaml" created
INFO Kubernetes file "notification-mongodb-deployment.yaml" created
INFO Kubernetes file "notification-service-deployment.yaml" created
INFO Kubernetes file "rabbitmq-deployment.yaml" created
INFO Kubernetes file "registry-deployment.yaml" created
INFO Kubernetes file "statistics-mongodb-deployment.yaml" created
INFO Kubernetes file "statistics-service-deployment.yaml" created
INFO Kubernetes file "turbine-stream-service-deployment.yaml" created
admin@iZ8vbc6meahsy5etdm796vZ:~/acsToAck$
```

> **Note** If the conversion succeeds, Kubernetes resource files are generated. Warnings appear for labels that cannot be automatically converted. Use one of the following methods to process these labels:
>
> - Modify the Swarm compose file and use Kompose to convert the file.
> - Modify the generated Kubernetes resource files.
> - Ignore the warnings. Deploy the generated Kubernetes resource files. Then, log on to the Container Service for Kubernetes (ACK) console and manually migrate the configurations.

2. Modify the Swarm compose file.

   Kompose can convert most Swarm labels. A small number of Swarm labels cannot be automatically converted. Kompose will be optimized to support more labels. When warnings appear, modify the Swarm compose file based on the warning information and use Kompose to convert the file again. For more information about the exceptions that may occur during conversion and how to handle these exceptions, see Troubleshooting.

3. Modify the generated Kubernetes resource files.

   After a Swarm compose file is converted to Kubernetes resource files, modify these Kubernetes resource files to migrate the labels that cannot be converted by Kompose. These labels include:

   - aliyun.routing.port_
   - aliyun.global
   - external
   - environment: constraint
   - extra_hosts
   - net
   - dns

   For more information about each label, see Application configuration labels, Application release labels,Network configuration labels, and Log configuration parameters.

11.5.5. Deploy Kubernetes resource files

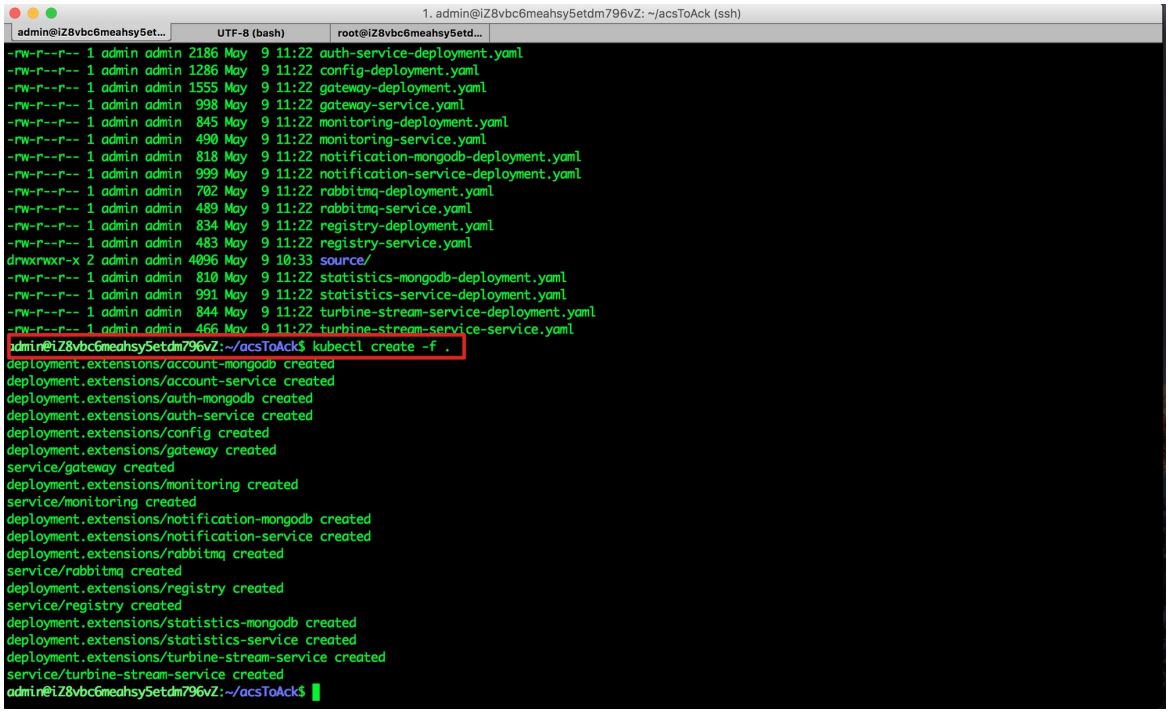
Context

After you generate Kubernetes resource files by using Kompose and preprocess the files, you can use the kubectl tool to deploy the Kubernetes resource files to a Container Service for Kubernetes (ACK) cluster. Make sure that cluster credentials are configured when you set up kubectl.

Procedure

1. Run the following command to deploy all Kubernetes resource files in the current directory:

```
kubectl create -f .            # Create resources based on filenames or standard input. The period (.) specifies that resources are created based on resource files in the current directory.
```



2. Log on to the ACK console. In the left-side navigation pane, choose **Applications > Deployments** to view the Deployment.
3. In the left-side navigation pane, click **Ingresses** or **Services** under **Ingresses and Load Balancing** to check the configurations of Ingresses or Services.

11.5.6. Manually migrate application configurations

For Swarm labels that are not supported by Kompose, you must log on to the Container Service for Kubernetes (ACK) console and perform manual migration. The following Swarm labels are not supported by Kompose. For more information about these labels, see Application configuration labels,Application release labels,Network configuration labels and Log configuration parameters.

- aliyun.auto_scaling
- hostname
- links
- external_links
- aliyun.lb.port_
- aliyun.log_ttl_

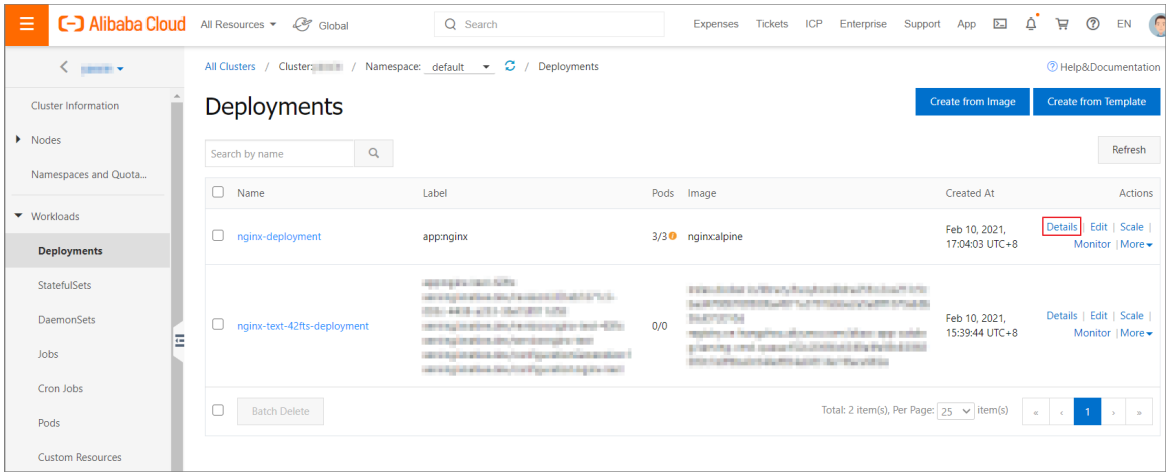
11.5.7. Debug application startup

This topic describes how to view the status of an application in the Container Service for Kubernetes (ACK) console after you migrate the application configurations. This topic also describes how to troubleshoot and fix application startup issues when the application configurations are migrated.

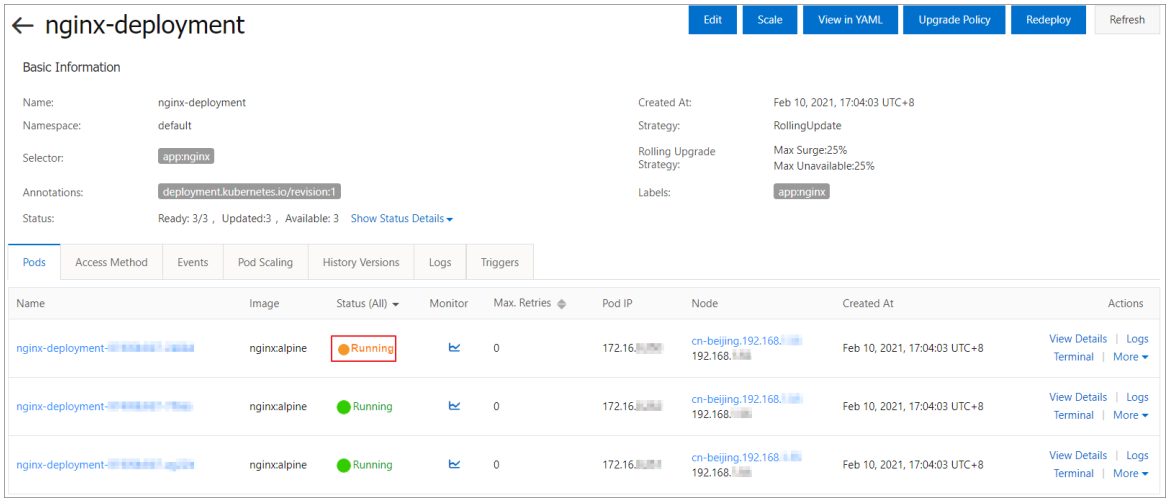
Check the application startup process

After you manually migrate application configurations or migrate application configurations by using Kompose, you can check whether the application starts up as expected in the ACK console. If any exceptions occur during the startup process, you can view application boot log in the console to locate the causes.

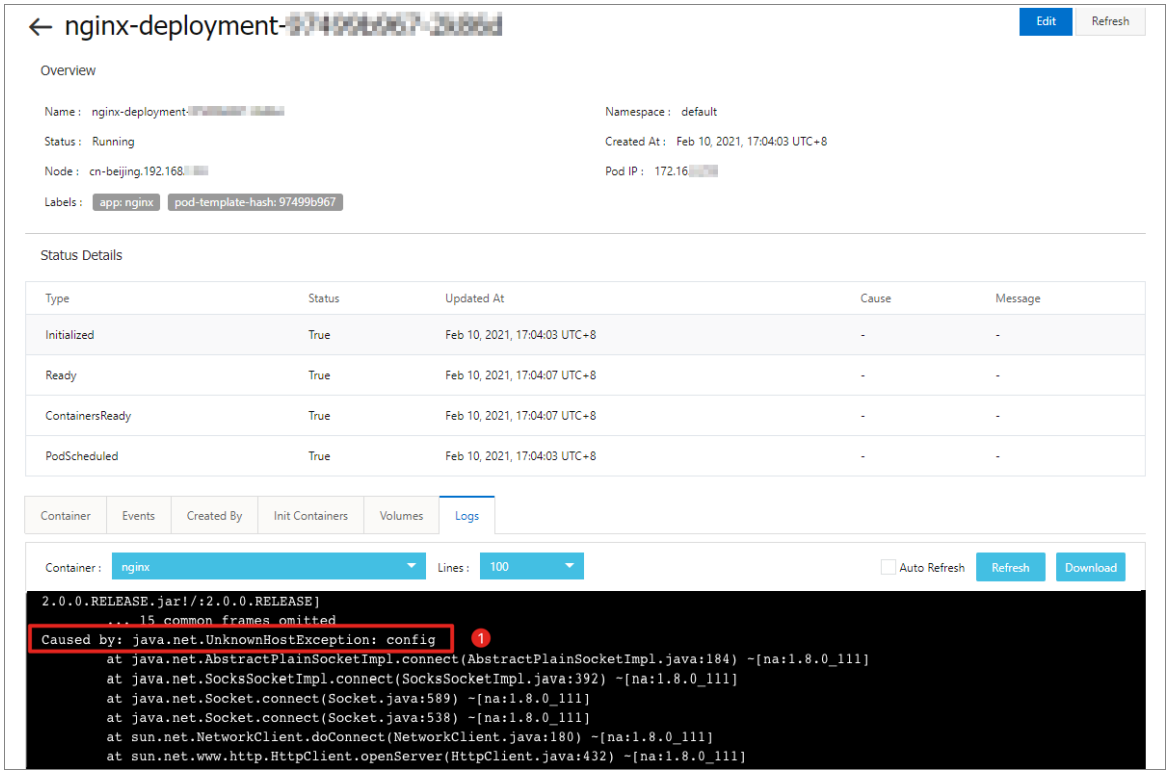
1.
2.
3.
4.
5. Select the corresponding namespace and click **Details** on the right side of the application that fails to start up.



6. Click the **Pods** tab, find the container that runs the application and click **Logs** in the Actions column.



7. In the **Logs** tab, view the log of the application.

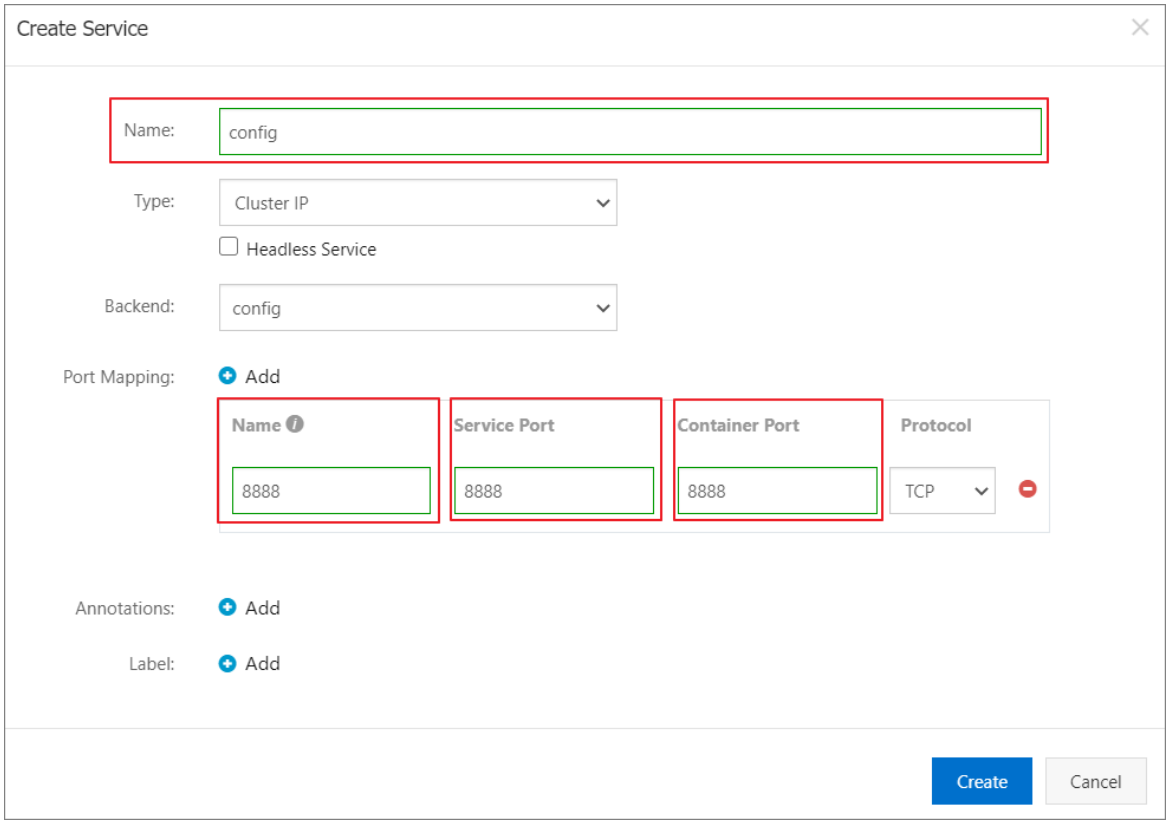


Fix application startup issues

Analyze and fix application startup issues successively. For example, the preceding log indicates that the nginx-deployment application fails to access config. To fix this issue, perform the following operations:

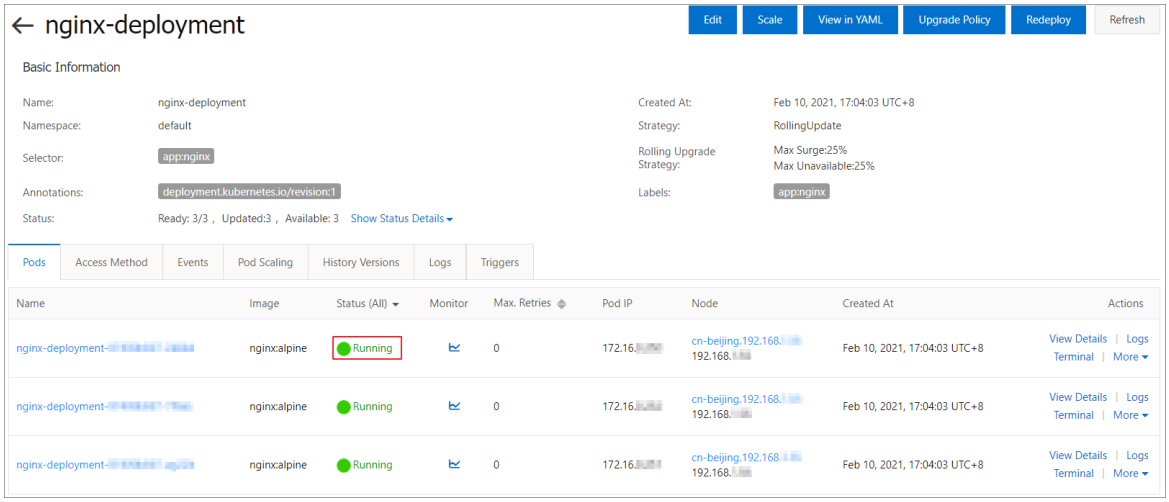
1. Manually create a Service in the ACK cluster to allow other applications to access config.

   The **Name** in the following figure indicates the Service name. The name must be the same as the hostname. You can specify **Port Mapping** as needed. For example, you can specify the same value for **Name**, **Service Port**, and **Container Port**. For more information, see Manage Services in the ACK console.



2. After you create the Service, deploy the application again to check whether the issue is fixed.

   If the state of the container is **Running** as shown in the following figure, it indicates that the container starts up as expected.



Exceptions may occur during the application startup process. For more information about these exceptions and how to fix them, see Troubleshooting.

11.5.8. Migrate log configurations of applications

If your applications use Log Service, make sure that application logs are collected to Log Service as expected from the related cluster of Container Service for Kubernetes (ACK) during the migration. You must also check whether services that are based on Log Service run as expected, such as Cloud Monitor.

1. Log on to the Log Service console. In the **Projects** section, find and click the **Log Service project**

that is used.

2. On the **Logstores** tab, you can view the Logstores that are generated for the Swarm and ACK clusters.

> **Note** The ACK cluster is configured to use the Log Service project that is used by the Swarm cluster when the ACK cluster is created. Kompose is also used to automatically migrate the log configurations. However, the Logstores for the Swarm cluster and the related ACK cluster are generated based on different rules. Therefore, application events of the related ACK cluster are logged in different Logstores under the same Log Service project.
>
> - The names of Logstores for the Swarm cluster are automatically generated in the format of *acslog-${app}-${name}*.
> - The names of Logstores for the ACK cluster are automatically generated in the format of *${name}*. Each Logstore for the ACK cluster corresponds to a Logstore for the Swarm cluster.
> - In Kubernetes, Logstore names cannot contain uppercase letters. Therefore, Kompose automatically converts uppercase letters to lowercase letters during the migration.

You must also migrate log configurations or switch Logstores for other Alibaba Cloud services that collect application events to the Log Service project of the Swarm cluster. The following list provides the Alibaba Cloud services that require configurations migration or conversion. For more information about how to convert and modify configurations related to logging, see Log Service - Real-time consumption.

- Real-time consumption: You can consume logs in real time by using the Log Service SDK, Storm Spout, Spark Streaming client, web console, and services such as Cloud Monitor and Application Real-Time Monitoring Service (ARMS).
- Real-time query: You can query and analyze logs in real time.
- Shipping and storage: You can deliver logs to various storage systems for processing.
- Secure log service: Log Service connects to cloud security services and uses an independent software vendor (ISV) to consume logs of cloud services.

11.5.9. Troubleshooting

This topic describes the exceptions that may occur when you migrate application configurations from a Container Service for Swarm cluster to a Container Service for Kubernetes (ACK) cluster. It also describes how to fix these exceptions.

Incorrect file version

Error message

```
FATA Version 2.1 of Docker Compose is not supported. Please use version 1, 2 or 3
```

Cause

The conversion is interrupted because Kompose supports only Compose files of versions 1, 2, and 3, except for versions 2.X.

Solution

Change `version: '2.X'` to `version: '2'` in the Compose file of the Container Service for Swarm cluster and use Kompose to convert the file again.

Key parsing errors

- Error message

```
ERRO Could not parse config for project source : Unsupported config option for account-db service: 'external'
```

Cause

The conversion is interrupted because Kompose cannot parse the external key.

Solution

If an exception of the **ERRO** or **FATA** severity occurs, delete the configuration that causes the exception from the Swarm Compose file. Then, use Kompose to convert the file again and manually migrate the configuration. For more information, see Application configuration labels, Application release labels, Network configuration labels, and Log configuration parameters.

- Error message

```
ERRO Could not parse config for project source : Unsupported config option for gateway service: 'net'
```

Cause

The conversion is interrupted because Kompose cannot parse the net key.

Solution

Delete the configuration that causes the exception from the Swarm Compose file and use Kompose to convert the file again. Then, manually migrate the configuration. For more information, see Application configuration labels, Application release labels, Network configuration labels, and Log configuration parameters.

Invalid value types

- Error message

```
ERRO Could not parse config for project source : Service 'auth-service' configuration key 'labels' contains an invalid type, it should be an array or object Unsupported config option for auth-service service: 'latest_image'
```

Cause

Kompose cannot convert the latest_image key because its value type is invalid.

Solution

Change the value type from BOOLEAN to STRING in the Swarm Compose file. For example, change true to `'true'`.

> **Note** This exception occurs to the following keys:
>
> - aliyun.latest_image: true
> - aliyun.global: true

- Error message

```
ERRO Could not parse config for project source : Cannot unmarshal '30' of type int into a string value
```

Cause

An invalid value type is detected. Check whether the value of the *aliyun.log_** key is 30 in the Compose file. The value must be enclosed in a pair of apostrophes (`''`) because the value must be a string, but not an integer.

Solution

Change 30 to `'30'` in the Swarm Compose file and use Kompose to convert the file again.

Unsupported keys

- Error message

```
WARN Unsupported hostname key - ignoring
```

Cause

In the Container Service for Swarm cluster, the hostName key specifies a hostname that can be used as a domain name to access services. However, the Swarm Compose file does not specify the corresponding container port and Kompose cannot automatically create a matching service for the ACK cluster. Therefore, you cannot access applications by using the hostname in the ACK cluster.

Solution

Deploy the Kubernetes resource files and then create a service in the ACK console. When you create the service, use hostName as the service name, and set the service port and container port to the same value.

- Error message

```
WARN Unsupported links key - ignoring
```

Cause

Similar to the hostName key, the links key specifies links names or aliases that can be used to access services. However, the Swarm Compose file does not specify the corresponding container port and Kompose cannot automatically create a matching service for the ACK cluster. Therefore, you cannot access applications by using links names or aliases in the ACK cluster.

Solution

Deploy the Kubernetes resource files and then create services in the ACK console. When you create the services, use the links names or aliases specified by the links key as the service names, and set the service port and container port to the same value.

Defects of key conversion

- Error message

```
WARN Handling aliyun routings label: [{rabbitmq.your-cluster-id.alicontainer.com  http 15672}]
```

Cause

In the Container Service for Swarm cluster, a test domain name provided by Container Service for Swarm is configured for simple routing. This test domain name is related to the ID of the Container Service for Swarm cluster. Kompose cannot automatically convert this test domain name to that of an ACK cluster because Kompose does not have the ID of the ACK cluster. You must manually modify the Ingress file and update the domain name.

Solution

Find the **-ingress.yaml* file of each Ingress and replace *host: your-cluster-id.alicontainer.com* with the test domain name of the ACK cluster. To obtain the test domain name, perform the following operations:

i. Log on to the ACK console. In the left-side navigation pane, choose **Clusters > Clusters**. On the Clusters page, find the Kubernetes-piggymetrics-cluster cluster and click **Manage** in the Actions column.

ii. On the **Basic Information** tab, the value in the Testing Domain field is **.c7f537c92438f415b943e7c2f8ca30b3b.cn-zhangjiakou.alicontainer.com*. Replace the value with *.your-cluster-id.alicontainer.com* in the Ingress file.

- Error message

```
WARN Handling aliyun routings label: [{gateway.your-cluster-id.alicontainer.com  http 4000} {gateway.swarm.piggymetrics.com  http 4000}]
```

Cause

In the Container Service for Swarm cluster, multiple domain names are configured for simple routing. However, Kompose can convert only one domain name. You must manually add Ingress rules for other domain names.

Solution

Modify the generated Kubernetes resource files. For more information, see Application configuration labels, Application release labels, Network configuration labels, and Log configuration parameters.

Application deployment failures

- Error message

```
error: error validating "logtail2-daemonset.yaml": error validating data: ValidationError (DaemonSet.status): missing required field "numberReady" in io.Kubernetes.api.extensions.v1beta1.DaemonSetStatus; if you choose to ignore these errors, turn validation off with --validate=false
```

Cause

Kompose automatically converts a service containing the **aliyun.global**: true key in the Container Service for Swarm cluster to a DaemonSet in the ACK cluster. However, the generated Kubernetes resource files contain the status field that records the intermediate status. This field causes the deployment failure.

status:

- currentNumberScheduled: 0
- desiredNumberScheduled: 0
- numberMisscheduled: 0

Solution

Delete the status field from the generated Kubernetes resource file **-daemonset.yaml* and then deploy the application again.

- Error message

```
error: error parsing auth-service-deployment.yaml: error converting YAML to JSON: yaml: l
ine 26: did not find expected key
```

Cause

The Kubernetes resource files cannot be parsed because the expected field is not found in the specified line. In most cases, the cause is invalid indentation. Kubernetes processes the current field as a sub-field of the previous field instead of processing it as a parallel field.

Solution

Correct the specified line of the Kubernetes resource files based on the field list and official documentation provided by ACK.

- Error message

```
error: error parsing auth-service-deployment.yaml: error converting YAML to JSON: yaml: l
ine 34: found character that cannot start any token
```

Cause

Characters that are not allowed for a token, such as tab characters, exist in the specified line of the Kubernetes resource files.

Solution

In the generated Kubernetes resource file **-daemonset.yaml*, replace tab characters in the specified line with space characters.

Application startup failures

- Error message

A container continues to restart and the health check fails.

```
Initialized: True
Ready: False
ContainersReady: False
PodScheduled: True
```

Cause

In ACK clusters, liveness probes and readiness probes are used to check whether containers are alive or ready. These probes are similar to aliyun.probe.url and aliyun.probe.cmd in Swarm. Kompose converts both aliyun.probe.url and aliyun.probe.cmd to liveness probes. In Container Service for Swarm clusters, aliyun.probe.url and aliyun.probe.cmd mark the container status only when an issue occurs. However, in ACK clusters, when a liveness probe detects an exception, it automatically stops container initialization and restarts the container.

Solution

i. Check whether the exception is caused by probe configurations.

   Delete liveness probes or readiness probes and check whether an application can start. If the application starts, it indicates that the probe configuration causes this issue.

ii. Modify the probe configuration.

Check the amount of time that is required to start the container. Then, adjust the settings of the liveness probe. Pay attention to the settings of the initialDelaySeconds, periodSeconds, and timeoutSeconds fields. For more information, see Create a stateless application by using a Deployment.

- Error message

A container continues to restart and the health check fails.

```
Initialized: True
Ready: False
ContainersReady: False
PodScheduled: True
```

The following shows the container log:

```
2019-05-22 06:42:09.245  INFO [gateway,,,] 1 --- [           main] c.c.c.ConfigServicePro
pertySourceLocator : Connect Timeout Exception on Url - http://config:8888. Will be tryin
g the next url if available
```

Cause

The request to `config:8888` timed out because the network mode of the pod is invalid. The network mode is set to `hostNetwork: true`. Therefore, the Elastic Compute Service (ECS) instances cannot parse the name of the Kubernetes service.

After you modify the configuration in the gateway-deployment.yaml file and run the kubectl apply command to redeploy the file, the issue may still exist.

This occurs because the modified **-deployment.yaml* file does not take effect.

Solution

To make sure that the modified file takes effect, log on to the ACK console, delete the corresponding application, and use kubectl to deploy the application again.

11.6. Appendix: Parameter comparisons

11.6.1. Application configuration labels

This topic describes the labels that are related to application configurations in clusters of Container Service for Swarm.

Swarm label	Description	Related Kubernetes parameter	Swarm configuration example	Kubernetes configuration example	How to migrate

Swarm label	Description	Related Kubernetes parameter	Swarm configuration example	Kubernetes configuration example	How to migrate
name	The name of a Service. This label does not have specific use and can be ignored.	Not supported in Kubernetes	-	-	No need to migrate
image	The container image.	image	-	-	Automatically converted by Kompose
ports	The container ports.	containerPort. If your Kubernetes cluster is accessed by external systems, you must create a NodePort type Service.	-	-	Automatically converted by Kompose
environment	The container environment variables.	env	-	-	Automatically converted by Kompose
volumes	The directories or volumes of the host, which are used by a container.	volumeMounts volumes	For more information, see the Swarm configuration example in volumes.	For more information, see the Kubernetes configuration example in volumes.	Automatically converted by Kompose
cap_add/ cap_drop	Grants a container the permissions to modify the kernel or revokes the permissions to modify the kernel from a container.	securityContext:capabilities	For more information, see the Swarm configuration example in cap_add/ cap_drop.	For more information, see the Kubernetes configuration example in cap_add/ cap_drop.	Automatically converted by Kompose

Swarm label	Description	Related Kubernetes parameter	Swarm configuration example	Kubernetes configuration example	How to migrate
privileged: true	Grants privileged permissions to the root account of a container. If you do not specify this parameter, the root account of a container has only permissions of a standard user. In a container that starts in privileged mode, you can view devices of the host. You can also mount devices to the container. You can start another Docker container in the container.	securityContext:privileged	For more information, see the Swarm configuration example in privileged: true.	For more information, see the Kubernetes configuration example in privileged: true.	Automatically converted by Kompose
mem_limit	The maximum amount of memory that can be used by a container.	resource:request/limits	For more information, see the Swarm configuration example in mem_limit.	For more information, see the Kubernetes configuration example in mem_limit.	Automatically converted by Kompose
cpu_shares	The maximum number of cores that can be used by a container.	resource:request/limits	For more information, see the Swarm configuration example in cpu_shares.	For more information, see the Kubernetes configuration example in cpu_shares.	Automatically converted by Kompose
kernel_memory	The same as the related parameter of the docker run command.	Not supported in Kubernetes	-	-	No need to migrate

Swarm label	Description	Related Kubernetes parameter	Swarm configuration example	Kubernetes configuration example	How to migrate
memswap_reservation	The same as the related parameter of the docker run command.	Not supported in Kubernetes	-	-	No need to migrate
memswap_limit	The same as the related parameter of the docker run command.	Not supported in Kubernetes	-	-	No need to migrate
shm_size	The same as the related parameter of the docker run command.	Not supported in Kubernetes	-	-	No need to migrate
oom-kill-disable	Specifies whether to disable the Out of Memory (OOM) killer for the container. It is the same as the --oom-kill-disable parameter of the `docker run` command.	Not supported in Kubernetes	-	-	No need to migrate

volumes

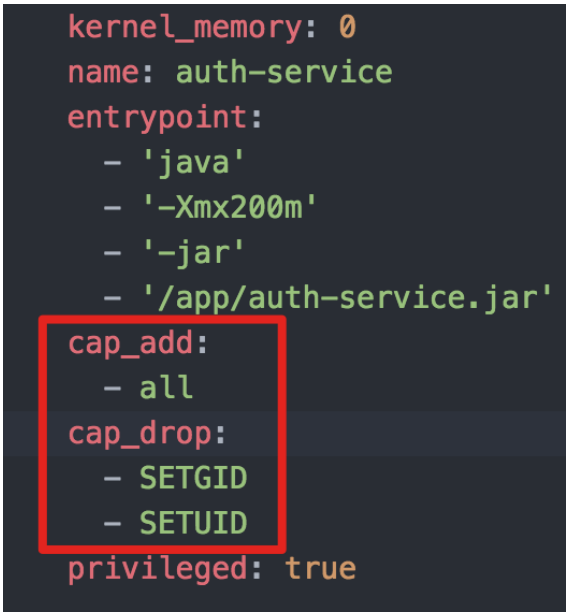
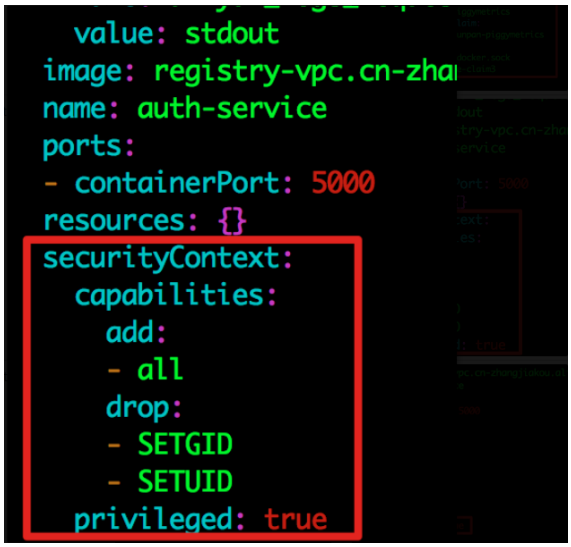
Swarm configuration example	Kubernetes configuration example

Swarm configuration example	Kubernetes configuration example
	 For more information, see auth-service-deployment.yaml.

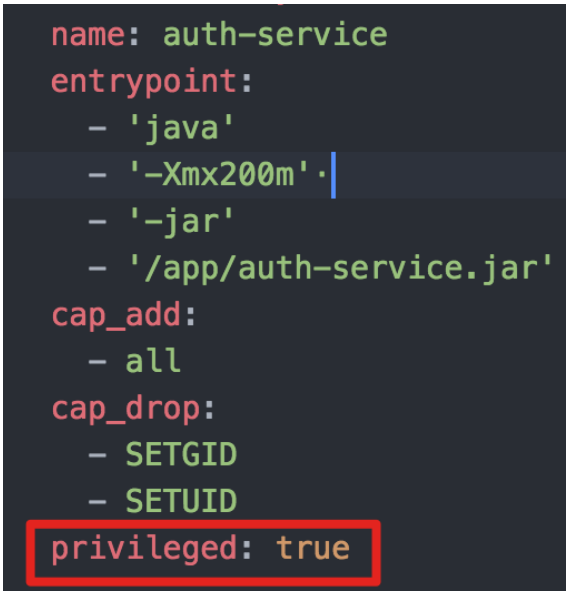
```
logging:
  options:
    max-file: '10'
    max-size: 10m
labels:
  aliyun.scale: '1'
  aliyun.log_store_requestlog: stdout  # 采集stdout日
  aliyun.log_ttl_requestlog: 30  # 设置requestlog日志
  aliyun.log.timestamp: false # Docker 在收集日志的时候
  aliyun.routing.session_sticky: "true"
  aliyun.routing.port_5000: auth-service.local
  aliyun.latest_image: true
  aliyun.depends: config
links:
  - auth-mongodb
volumes:
  - 'volume_name_piggymetrics:/data/oss:rw'
  - 'VN_NAS_PIGGYMETRICS:/data/nas:rw'
  - 'VN_YUNPAN_PIGGYMETRICS:/data/yunpan:rw'
  - '/var/run/docker.sock:/var/run/docker.sock:rw'
memswap_limit: 0
shm_size: 0
```

```
    securityContext:
      capabilities:
        add:
        - all
        drop:
        - SETGID
        - SETUID
      privileged: true
    volumeMounts:                                  1
    - mountPath: /data/oss
      name: volume-name-piggymetrics
    - mountPath: /data/nas
      name: vn-nas-piggymetrics
    - mountPath: /data/yunpan
      name: vn-yunpan-piggymetrics
    - mountPath: /var/run/docker.sock
      name: auth-service-claim3
  restartPolicy: Always
  volumes:                                         2
  - name: volume-name-piggymetrics
    persistentVolumeClaim:
      claimName: volume-name-piggymetrics
  - name: vn-nas-piggymetrics
    persistentVolumeClaim:
      claimName: vn-nas-piggymetrics
  - name: vn-yunpan-piggymetrics
    persistentVolumeClaim:
      claimName: vn-yunpan-piggymetrics
  - hostPath:
      path: /var/run/docker.sock
    name: auth-service-claim3
tus: {}
```

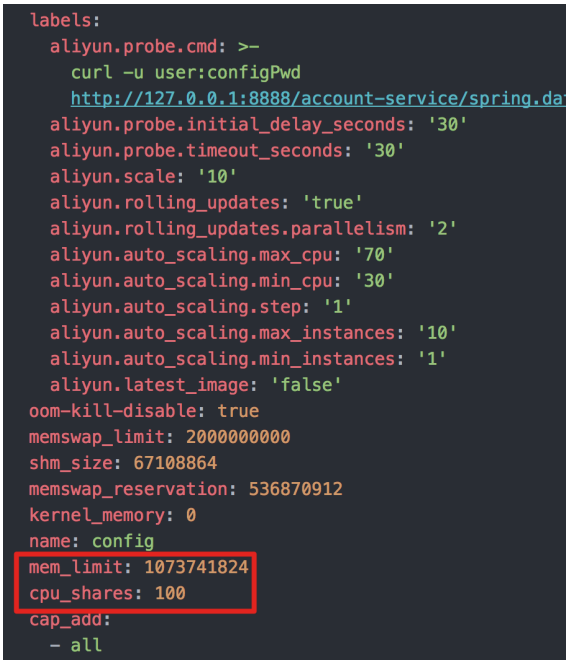
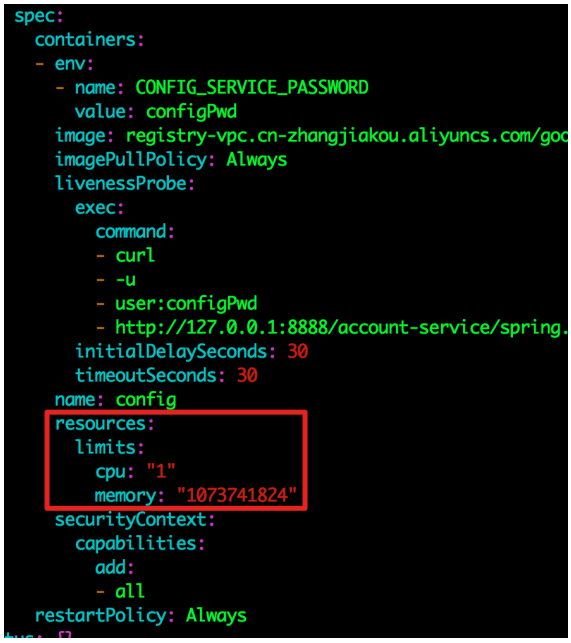
cap_add/cap_drop

Swarm configuration example	Kubernetes configuration example
	 For more information, see auth-service-deployment.yaml.

privileged: true

Swarm configuration example	Kubernetes configuration example
	 For more information, see auth-service-deployment.yaml.

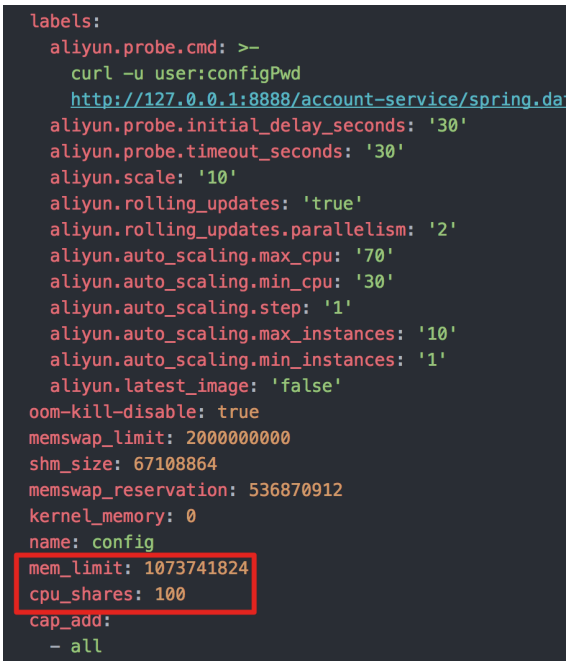
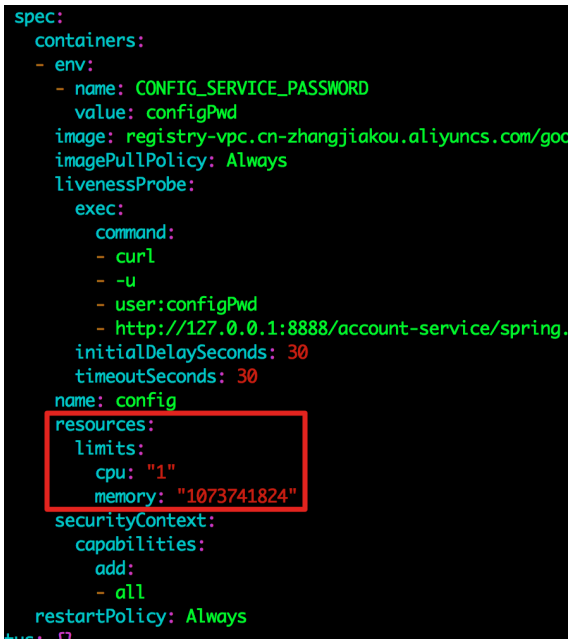
mem_limit

Swarm configuration example	Kubernetes configuration example
	 For more information, see config-deployment.yaml.

```
labels:
  aliyun.probe.cmd: >-
    curl -u user:configPwd
    http://127.0.0.1:8888/account-service/spring.da
  aliyun.probe.initial_delay_seconds: '30'
  aliyun.probe.timeout_seconds: '30'
  aliyun.scale: '10'
  aliyun.rolling_updates: 'true'
  aliyun.rolling_updates.parallelism: '2'
  aliyun.auto_scaling.max_cpu: '70'
  aliyun.auto_scaling.min_cpu: '30'
  aliyun.auto_scaling.step: '1'
  aliyun.auto_scaling.max_instances: '10'
  aliyun.auto_scaling.min_instances: '1'
  aliyun.latest_image: 'false'
oom-kill-disable: true
memswap_limit: 2000000000
shm_size: 67108864
memswap_reservation: 536870912
kernel_memory: 0
name: config
mem_limit: 1073741824
cpu_shares: 100
cap_add:
  - all
```

```
spec:
  containers:
  - env:
    - name: CONFIG_SERVICE_PASSWORD
      value: configPwd
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goo
    imagePullPolicy: Always
    livenessProbe:
      exec:
        command:
        - curl
        - -u
        - user:configPwd
        - http://127.0.0.1:8888/account-service/spring.
      initialDelaySeconds: 30
      timeoutSeconds: 30
    name: config
    resources:
      limits:
        cpu: "1"
        memory: "1073741824"
    securityContext:
      capabilities:
        add:
        - all
  restartPolicy: Always
```

cpu_shares

Swarm configuration example	Kubernetes configuration example
	 For more information, see config-deployment.yaml.

```
labels:
  aliyun.probe.cmd: >-
    curl -u user:configPwd
    http://127.0.0.1:8888/account-service/spring.da
  aliyun.probe.initial_delay_seconds: '30'
  aliyun.probe.timeout_seconds: '30'
  aliyun.scale: '10'
  aliyun.rolling_updates: 'true'
  aliyun.rolling_updates.parallelism: '2'
  aliyun.auto_scaling.max_cpu: '70'
  aliyun.auto_scaling.min_cpu: '30'
  aliyun.auto_scaling.step: '1'
  aliyun.auto_scaling.max_instances: '10'
  aliyun.auto_scaling.min_instances: '1'
  aliyun.latest_image: 'false'
oom-kill-disable: true
memswap_limit: 2000000000
shm_size: 67108864
memswap_reservation: 536870912
kernel_memory: 0
name: config
mem_limit: 1073741824
cpu_shares: 100
cap_add:
  - all
```

```
spec:
  containers:
  - env:
    - name: CONFIG_SERVICE_PASSWORD
      value: configPwd
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goo
    imagePullPolicy: Always
    livenessProbe:
      exec:
        command:
        - curl
        - -u
        - user:configPwd
        - http://127.0.0.1:8888/account-service/spring.
      initialDelaySeconds: 30
      timeoutSeconds: 30
    name: config
    resources:
      limits:
        cpu: "1"
        memory: "1073741824"
    securityContext:
      capabilities:
        add:
        - all
  restartPolicy: Always
```

11.6.2. Application release labels

This topic describes the labels that are related to application releases in clusters of Container Service for Swarm.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
restart	The policy for restarting the container.	restartPolicy	-	-	Automatically converted by Kompose.
aliyun.latest_image	Specifies whether to pull the latest image.	imagePullPolicy	aliyun.latest_image: true	For more information, see the sample Kubernetes configuration in aliyun.latest_image.	Migrate manually: After you use Kompose to convert the Swarm compose file, add the imagePullPolicy field to the Kubernetes resource file *-deployment.yaml of the application. The default value is Always.
depends_on	The dependencies of services.	Uses methods such as init container, liveness probe, and readiness probe to implement features such as health check and dependency check. For more information, see Handle service dependencies.	-	-	Depends on workload implementation.
aliyun.depends	The dependencies of services.	Uses methods such as init container, liveness probe, and readiness probe to implement features such as health check and dependency check. For more information, see Handle service dependencies.	-	-	Depends on workload implementation.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
environment:affinity:service!=db	The constraints of service deployments. For more information, see Service deployment constraints (affinity:service).	Uses labels such as nodeAffinity, podAffinity, and podAntiAffinity to configure affinity attributes of nodes and pods. For more information, see Affinity and anti-affinity.	-	-	Depends on workload implementation.
environment:constraint:group==1	Specifies that pods are scheduled to nodes with the *group:1* label.	Uses nodeSelector to filter nodes. For more information, see nodeSelector.	For more information, see the sample Swarm configuration in environment:constraint:group==1.	For more information, see the sample Kubernetes configuration in environment:constraint:group==1.	Migrate manually: After you use Kompose to convert the Swarm Compose file, add the nodeSelector field to the Kubernetes resource file **-deployment.yaml* of the application.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
aliyun.global	Specifies whether the service is for global use.	You can create DaemonSet type applications for global use. For more information, see DaemonSet.	For more information, see the sample Swarm configuration in aliyun.global.	For more information, see the sample Kubernetes configuration in aliyun.global.	Automatic conversion by using Kompose and manual modification are both required. Modify the configurations: After you use Kompose to convert the Swarm compose file, you must delete the status field in the generated Kubernetes resource file. The status field records intermediate statuses.
aliyun.scale	The number of containers for the service. You can use this field to scale out the service.	replicas	aliyun.scale: '10'	replicas: 10	Automatically converted by Kompose.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
aliyun.rolling_updates	Specifies whether to enable rolling update for the service.	strategy.type.RollingUpdate. For more information, see Strategy.	For more information, see the sample Swarm configuration in aliyun.rolling_updates and aliyun.rolling_updates.parallelism.	For more information, see the sample Kubernetes configuration in aliyun.rolling_updates and aliyun.rolling_updates.parallelism.	Migrate manually: After you use Kompose to convert the Swarm compose file, you must manually configure the rolling update policy.
aliyun.rolling_updates.parallelism	The number of containers that are concurrently updated.	maxUnavailable. For more information, see Strategy.			Migrate manually: After you use Kompose to convert the Swarm compose file, you must manually configure the rolling update policy.
aliyun.auto_scaling.*	The policy for automatically adjusting the number of containers for the service based on the container resource usage. For more information, see Container auto scaling.	This label does not have a mapping field in Kubernetes. To enable auto scaling of pods, you can create an application and enable Horizontal Pod Autoscaling (HPA) in the Container Service for Kubernetes (ACK) console. For more information, see HPA.	For more information, see the sample Swarm configuration in aliyun.auto_scaling. *.	For more information, see the sample Kubernetes configuration in aliyun.auto_scaling. *.	Migrate manually: After you deploy the Kubernetes resource file, configure HPA in the ACK console.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
aliyun.probe.cmd	The shell command for health check. For more information, see probe.	livenessProbe	For more information, see the sample Swarm configuration in aliyun.probe.cmd, aliyun.probe.initial_delay_seconds, and aliyun.probe.timeout_seconds.	For more information, see the sample Kubernetes configuration in aliyun.probe.cmd, aliyun.probe.initial_delay_seconds, and aliyun.probe.timeout_seconds.	Automatically converted by Kompose.
aliyun.probe.initial_delay_seconds	The delay duration for health check after the container starts. Unit: seconds. For more information, see probe.	livenessProbe			Automatically converted by Kompose.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
aliyun.probe.timeout_seconds	The timeout period of a health check. For more information, see probe.	livenessProbe			Automatically converted by Kompose.
aliyun.probe.url	The URL to which the HTTP or TCP request for health check is sent. For more information, see probe.	livenessProbe	For more information, see the sample Swarm configuration in aliyun.probe.url.	For more information, see the sample Kubernetes configuration in aliyun.probe.url.	Automatically converted by Kompose.

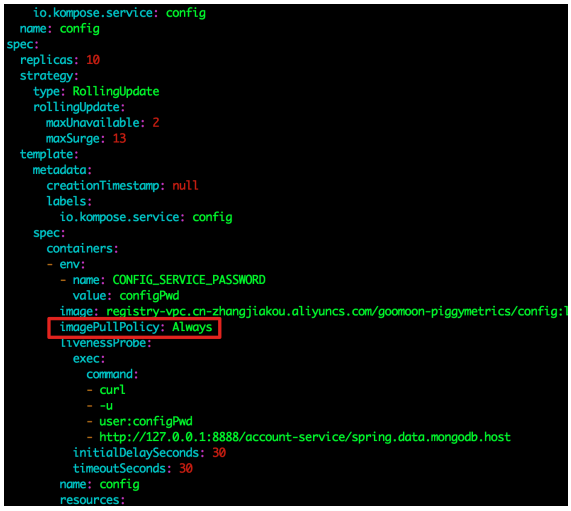
Note

- In Swarm, health check is performed to check the container status. However, the container can provide services to external systems regardless of the status.
- In Kubernetes, if a health check performed by liveness probe fails in a pod, Kubernetes forcibly restarts the pod. Therefore, you must set an appropriate delay period for the liveness probe. Otherwise, Kubernetes repeatedly restarts the pod and the application fails to start.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
extra_hosts	The entries to be added to the hosts file of the container.	The hostAliases parameter in Kubernetes. For more information, see Adding Additional Entries with HostAliases.	For more information, see the sample Swarm configuration in extra_hosts.	For more information, see the sample Kubernetes configuration in extra_hosts.	Migrate manually: After you use Kompose to convert the Swarm compose file, add the hostAliases parameter to the Kubernetes resource file *-deployment.yaml of the application.
entrypoint	The entry point to overwrite the default one.	command	For more information, see the sample Swarm configuration in entrypoint.	For more information, see the sample Kubernetes configuration in entrypoint.	Automatically converted by Kompose.
command	The command to overwrite the default one.	args	For more information, see the sample Swarm configuration in command.	For more information, see the sample Kubernetes configuration in command.	Automatically converted by Kompose.

aliyun.latest_image

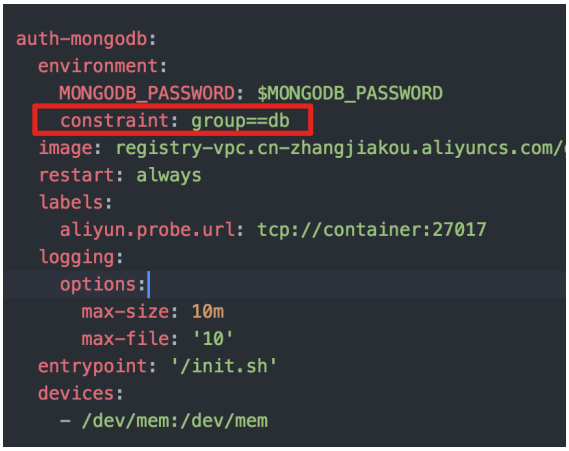
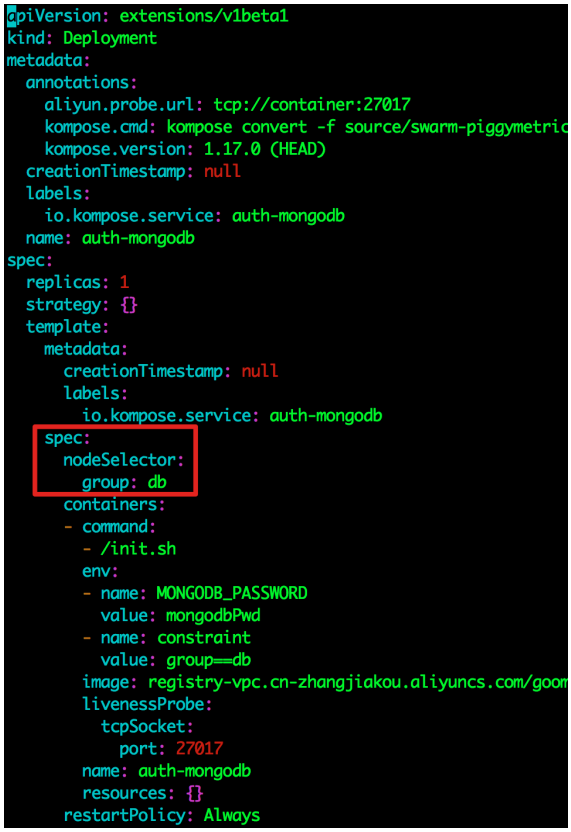
Sample Swarm configuration	Sample Kubernetes configuration

Sample Swarm configuration	Sample Kubernetes configuration
aliyun.latest_image: true	 For more information, see config-deployment.yaml.

```
    io.kompose.service: config
  name: config
spec:
  replicas: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 13
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: config
    spec:
      containers:
      - env:
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/config:1
        imagePullPolicy: Always
        livenessProbe:
          exec:
            command:
            - curl
            - -u
            - user:configPwd
            - http://127.0.0.1:8888/account-service/spring.data.mongodb.host
          initialDelaySeconds: 30
          timeoutSeconds: 30
        name: config
        resources:
```

environment: constraint:group==1

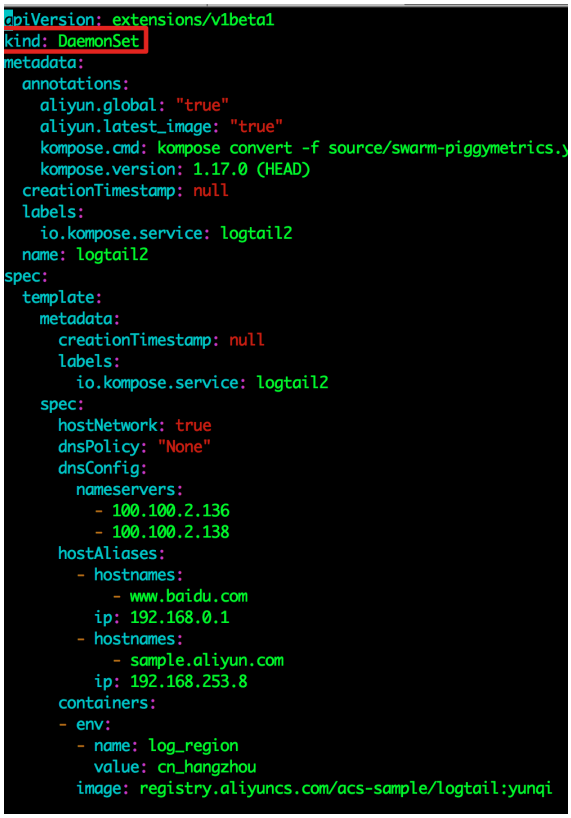
Sample Swarm configuration	Sample Kubernetes configuration

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see auth-mongodb-deployment.yaml.

```
auth-mongodb:
  environment:
    MONGODB_PASSWORD: $MONGODB_PASSWORD
    constraint: group==db
  image: registry-vpc.cn-zhangjiakou.aliyuncs.com/
  restart: always
  labels:
    aliyun.probe.url: tcp://container:27017
  logging:
    options:
      max-size: 10m
      max-file: '10'
  entrypoint: '/init.sh'
  devices:
    - /dev/mem:/dev/mem
```

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.probe.url: tcp://container:27017
    kompose.cmd: kompose convert -f source/swarm-piggymetric
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: auth-mongodb
  name: auth-mongodb
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: auth-mongodb
    spec:
      nodeSelector:
        group: db
      containers:
      - command:
        - /init.sh
        env:
        - name: MONGODB_PASSWORD
          value: mongodbPwd
        - name: constraint
          value: group==db
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goom
        livenessProbe:
          tcpSocket:
            port: 27017
        name: auth-mongodb
        resources: {}
      restartPolicy: Always
```

aliyun.global

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see logtail2-daemonset.yaml.

```
logtail2:
  image: registry.aliyuncs.com/acs-sample/logtail:yunqi
  labels:
    aliyun.latest_image: true
    aliyun.global: true
  devices:
    - /dev/mem:/dev/mem
  environment:
    - log_region=cn_hangzhou
  net: host
  extra_hosts:
    - "www.baidu.com:192.168.0.1"
    - "apollo.pro.sample.com:192.168.253.8"
  dns:
    - 100.100.2.136
    - 100.100.2.138
  cap_add:
    - SYS_RAWIO
```

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  annotations:
    aliyun.global: "true"
    aliyun.latest_image: "true"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.y
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: logtail2
  name: logtail2
spec:
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: logtail2
    spec:
      hostNetwork: true
      dnsPolicy: "None"
      dnsConfig:
        nameservers:
          - 100.100.2.136
          - 100.100.2.138
      hostAliases:
        - hostnames:
            - www.baidu.com
          ip: 192.168.0.1
        - hostnames:
            - sample.aliyun.com
          ip: 192.168.253.8
      containers:
      - env:
        - name: log_region
          value: cn_hangzhou
        image: registry.aliyuncs.com/acs-sample/logtail:yunqi
```

aliyun.rolling_updates and aliyun.rolling_updates.parallelism

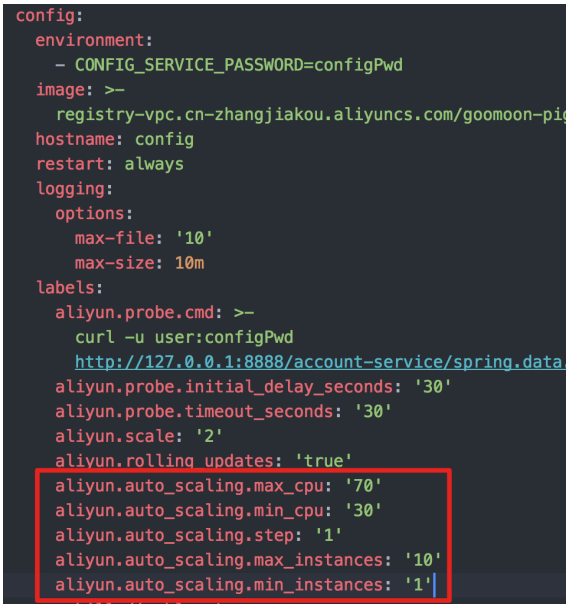
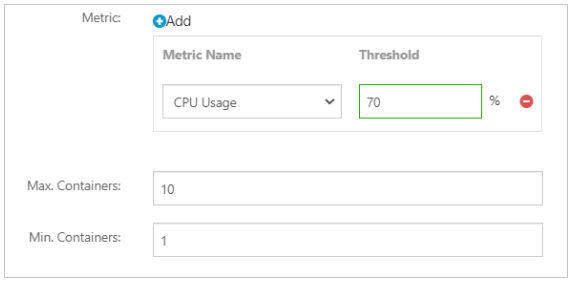
Sample Swarm configuration	Sample Kubernetes configuration

```
config:
  environment:
    - CONFIG_SERVICE_PASSWORD=configPwd
  image: >-
    registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-pig
  hostname: config
  restart: always
  logging:
    options:
      max-file: '10'
      max-size: 10m
  labels:
    aliyun.probe.cmd: >-
      curl -u user:configPwd
      http://127.0.0.1:8888/account-service/spring.data.
    aliyun.probe.initial_delay_seconds: '30'
    aliyun.probe.timeout_seconds: '30'
    aliyun.scale: '10'
    aliyun.rolling_updates: 'true'
    aliyun.rolling_updates.parallelism: '2'
    aliyun.auto_scaling.max_cpu: '70'
    aliyun.auto_scaling.min_cpu: '30'
    aliyun.auto_scaling.step: '1'
    aliyun.auto_scaling.max_instances: '10'
    aliyun.auto_scaling.min_instances: '1'
    aliyun.latest_image: 'false'
```

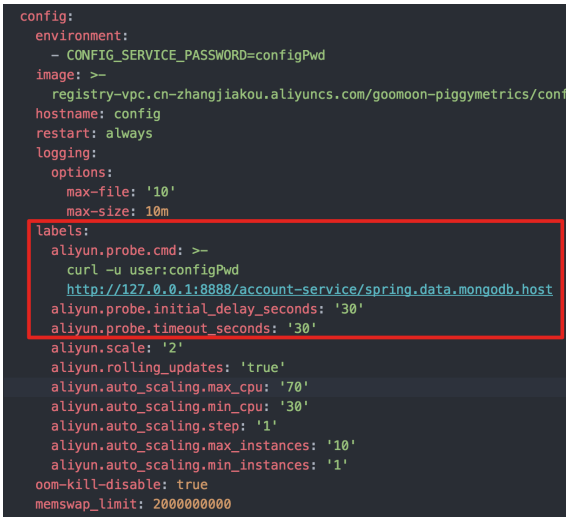
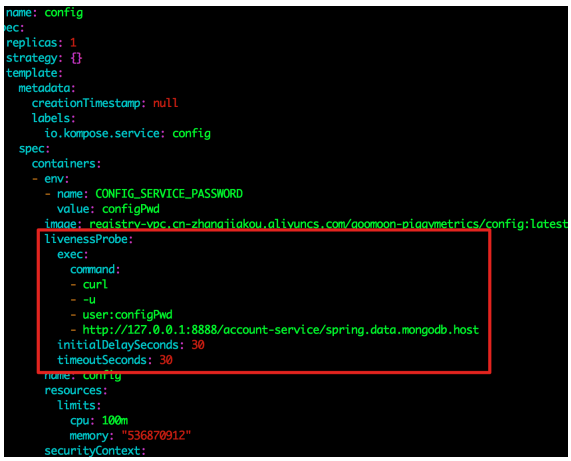
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.auto_scaling.max_cpu: "70"
    aliyun.auto_scaling.max_instances: "10"
    aliyun.auto_scaling.min_cpu: "30"
    aliyun.auto_scaling.min_instances: "1"
    aliyun.auto_scaling.step: "1"
    aliyun.latest_image: "false"
    aliyun.probe.cmd: curl -u user:configPwd http://127.
    aliyun.probe.initial_delay_seconds: "30"
    aliyun.probe.timeout_seconds: "30"
    aliyun.rolling_updates: "true"
    aliyun.rolling_updates.parallelism: "2"
    aliyun.scale: "10"
    kompose.cmd: kompose convert -f source/swarm-piggyme
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: config
  name: config
spec:
  replicas: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 13
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: config
    spec:
      containers:
```

For more information, see config-deployment.yaml.

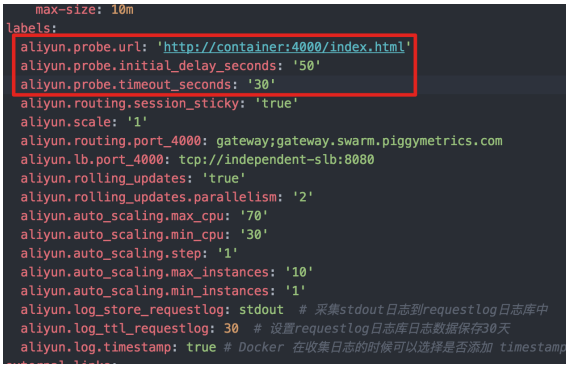
aliyun.auto_scaling. *

Sample Swarm configuration	Sample Kubernetes configuration
	

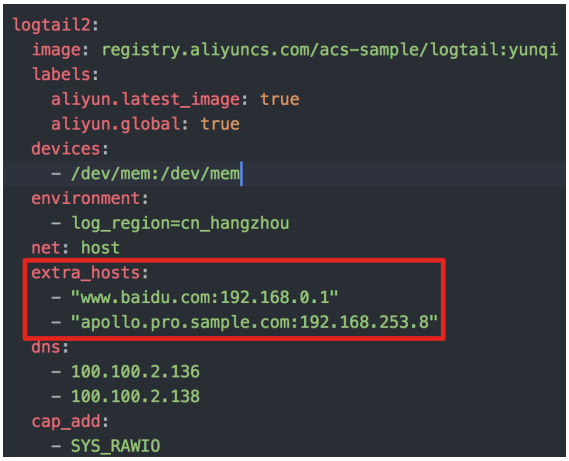
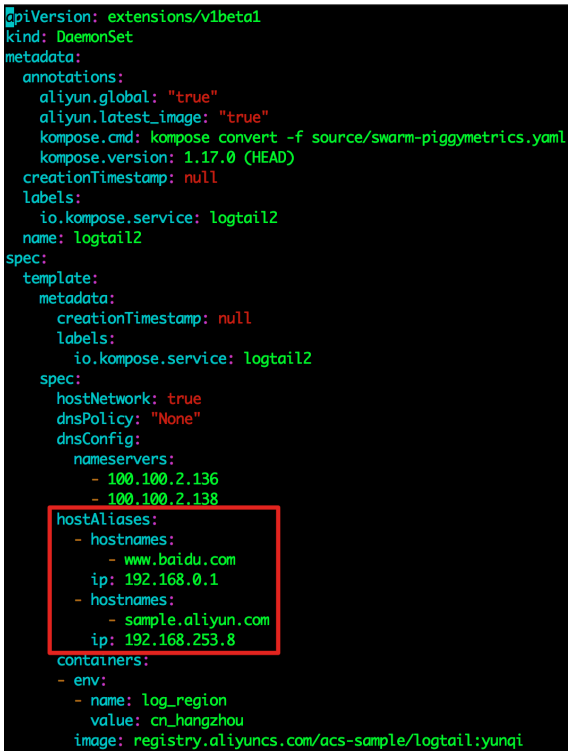
aliyun.probe.cmd, aliyun.probe.initial_delay_seconds, and aliyun.probe.timeout_seconds

Sample Swarm configuration	Sample Kubernetes configuration
 aliyun.latest_image: true	 For more information, see config-deployment.yaml.

aliyun.probe.url

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see gateway-deployment.yaml.

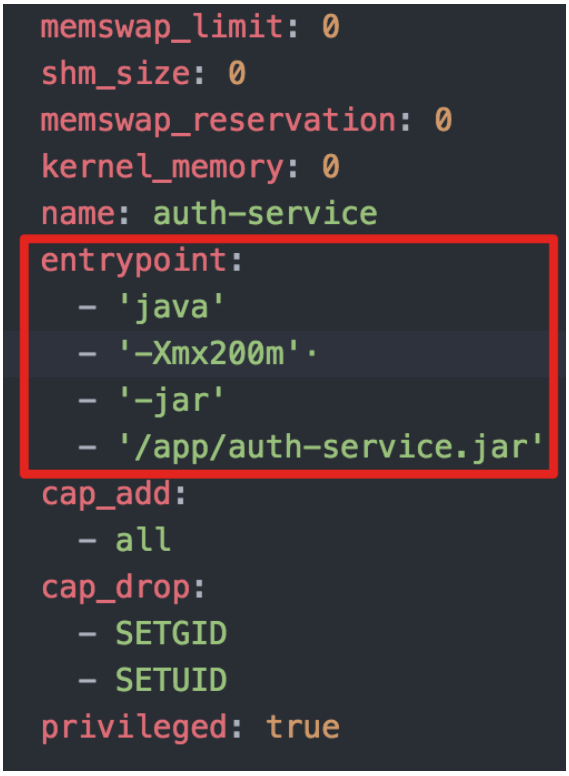
extra_hosts

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see logtail2-daemonset.yaml.

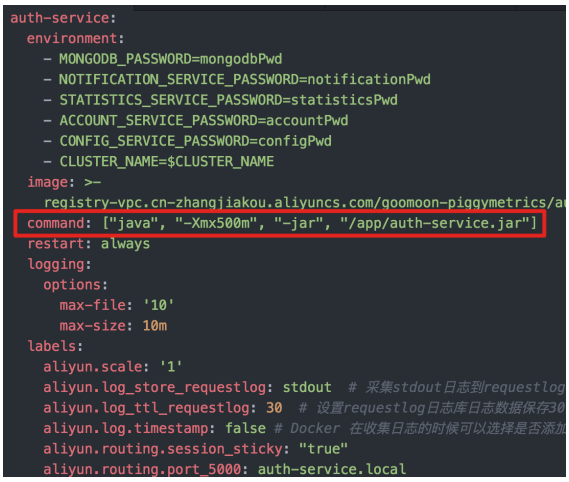
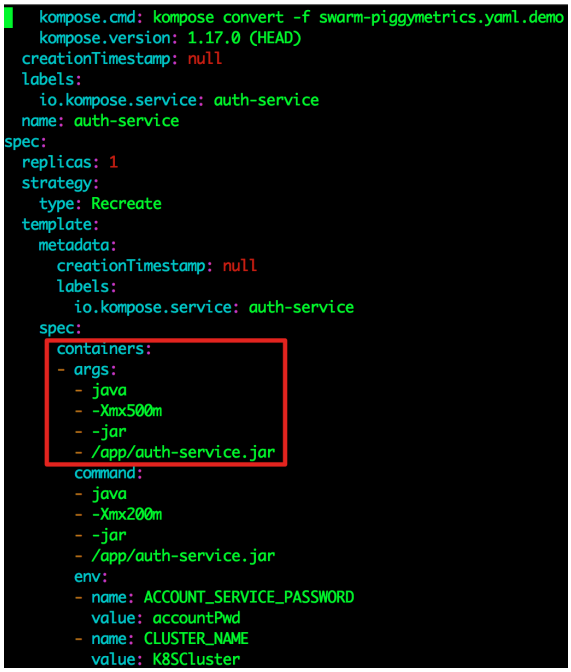
```
logtail2:
  image: registry.aliyuncs.com/acs-sample/logtail:yunqi
  labels:
    aliyun.latest_image: true
    aliyun.global: true
  devices:
    - /dev/mem:/dev/mem
  environment:
    - log_region=cn_hangzhou
  net: host
  extra_hosts:
    - "www.baidu.com:192.168.0.1"
    - "apollo.pro.sample.com:192.168.253.8"
  dns:
    - 100.100.2.136
    - 100.100.2.138
  cap_add:
    - SYS_RAWIO
```

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  annotations:
    aliyun.global: "true"
    aliyun.latest_image: "true"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: logtail2
  name: logtail2
spec:
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: logtail2
    spec:
      hostNetwork: true
      dnsPolicy: "None"
      dnsConfig:
        nameservers:
          - 100.100.2.136
          - 100.100.2.138
      hostAliases:
        - hostnames:
            - www.baidu.com
          ip: 192.168.0.1
        - hostnames:
            - sample.aliyun.com
          ip: 192.168.253.8
      containers:
      - env:
        - name: log_region
          value: cn_hangzhou
        image: registry.aliyuncs.com/acs-sample/logtail:yunqi
```

entrypoint

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see auth-service-deployment.yaml.

command

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see auth-service-deployment.yaml.

11.6.3. Network configuration labels

This topic describes the labels that are related to network configurations in clusters of Container Service for Swarm.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
net	The network mode. This parameter is available only in a Swarm compose file of version 1. It is the same as the --net parameter used in the Docker CLI. For more information, see net.	Some parameters can be directly converted into Kubernetes. For example, net: "host" can be directly converted to hostNetwork: true. Other values are not supported in Kubernetes. For more information, see Host namespaces.	For more information, see the sample Swarm configuration in net.	For more information, see the sample Kubernetes configuration in net.	Migrate manually: After you use Kompose to convert the Swarm compose file, you must manually delete unsupported parameters.
					Automatic conversion by using Kompose and manual modification are both required. Manual modification: • Common domain name: After you use Kompose to convert the Swarm compose file, find the Kubernetes resource file **-ingress.yaml* of each Ingress.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
aliyun.routing.port_	• The simple routing configuration in Swarm. This parameter specifies one or more domain names for accessing the Service. For more information, see routing. • If a domain name is suffixed with .local, it is only used for routing and load balancing among Services inside the cluster. For more information, see 集群内服务间路由和负载均衡.	The ingress parameter. Kompose converts only the first domain name. You must manually add rules for other domain names. In addition, you must replace the test domain name. For more information, see Advanced NGINX Ingress configurations.	For more information, see the sample Swarm configuration in aliyun.routing.port_.	For more information, see the sample Kubernetes configuration in aliyun.routing.port_.	◦ Replace host: your-cluster-id.alicontainer.com in this file with the test domain name of the Kubernetes cluster. For example, the test domain name of cluster k8s-piggymetrics-cluster is *.c7f537c92438f415b943e7c2f8ca30b3b.cn-zhangjiakou.alicontainer.com. Use this test domain name to replace .your-cluster-id.alicontainer.com in each Ingress resource file. ◦ If two or more domain names exist, Kompose converts only the first domain name. You must manually add rules for other domain names in each ingress resource file. • Domain name suffixed with .local: ◦ You must first deploy the Kubernetes resource files.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
					○ Create a Service in the Container Service for Kubernetes (ACK) console. A domain name suffixed with .local can be used for only communications within the Swarm cluster, and no port number needs to be specified. In Kubernetes, a Service name can contain only lowercase letters and hyphens (`-`). You cannot add the dot (.) in .local to the Service name in Kubernetes. You must modify the application code to migrate the domain name suffixed with .local to Kubernetes.
aliyun.routing.session_sticky	Specifies whether to maintain a sticky session when the routing parameter is set for routing requests. **Note** This label must be used together with `routing`.	Not supported in Kubernetes.	-	-	No need to migrate.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
hostname	The hostname used for accessing a Service across containers that run on different nodes based on Domain Name System (DNS) and Server Load Balancer (SLB). Typically, it is used by a Service to access another Service.	In Kubernetes, you can create a Service for accessing an application across pods. For more information, see Manage Services.	For more information, see the sample Swarm configuration in hostname.	For more information, see the sample Kubernetes configuration in hostname.	Migrate manually: 1. You must first deploy the Kubernetes resource files. 2. Create ClusterIP type Services in the ACK console. Kompose cannot automatically convert certain parameters such as hostname, links, and external_links in the Swarm
links	1. The names or aliases for accessing Services across containers that run on different nodes. 2. Similar to depends_on, this parameter can be used to determine the startup sequence of Services.	1. Create Services with the same names as those specified by the links parameter for accessing applications across pods. For more information, see Manage Services. 2. The startup sequence can be determined depending on the requirements of your workload.	For more information, see the sample Swarm configuration in links.	For more information, see the sample Kubernetes configuration in links.	

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
external_links	This parameter connects two Services to enable communication between each other. Compared with the links parameter, the external_links parameter allows you to connect a Service to containers that are created through other Swarm compose files or in other methods.	In a Kubernetes cluster, you can create a Service for accessing an application across pods. The pods must be managed by the Kubernetes cluster. Migration method: 1. Migrate the containers specified by external_links parameter to the Kubernetes cluster. 2. Create Services with the same names as those specified by the external_links parameter for accessing applications across pods. For more information, see Manage Services.	For more information, see the sample Swarm configuration in external_links.	For more information, see the sample Kubernetes configuration in external_links.	**Note** In Kubernetes, a Service name can contain only lowercase letters and hyphens (-). If parameters such as hostname, links, and external_links contain other characters such as dots (.), you must modify the application code to migrate these parameters to Kubernetes.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate

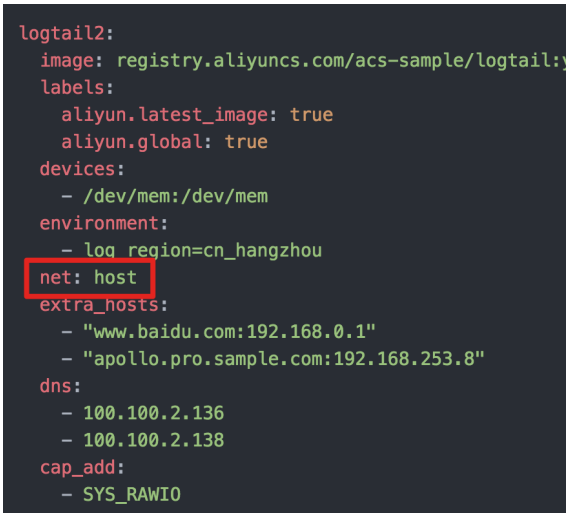
`external:`
 	
`host: $RDS_URL`
 	
`ports:`
 	
`- 3306`
	The address of the external system to which the Service is directly linked.
	• host: the domain name of the external system.
	• ports: the ports of the external system.
		The headless Service in Kubernetes. For more information, see Headless Services.	For more information, see the sample Swarm configuration in external: ***.	For more information, see the sample Kubernetes configuration in external: ***.	Migrate manually:
					1. Delete the external parameter from the Swarm compose file before you use Kompose to convert this file. This prevents the conversion from being interrupted.
					2. After you use Kompose to convert the Swarm compose file, create a Kubernetes resource file for deploying a headless Service, which is Service of the ExternalName type.
					3. Use kubectl to deploy this Kubernetes resource file together with other Kubernetes resource files.

					Migrate manually: After the Service is deployed, create a LoadBalancer type Service to access it through an SLB instance. For more information, see Use annotations to configure load balancing.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
aliyun.lb.port_*	The Service port to be exposed through an SLB instance to the public or internal networks in Network Address Translation (NAT) mapping mode. For more information, see Server Load Balancer routing.	Access the Service through an SLB instance, which is similar to load balancing in Swarm. For more information, see Use annotations to configure load balancing.	For more information, see the sample Swarm configuration in aliyun.lb.port_*.	For more information, see the sample Kubernetes configuration in aliyun.lb.port_*.	**Note** • The configuration in this example is for reference only. You must follow related Kubernetes documents to configure the Service based on the requirements of your workload. • When an SLB instance is attached to the Service port, production traffic may be immediately received. We recommend that you attach the SLB instance to a test port for testing before you attach the SLB instance to the Service port.

Swarm label	Description	Related Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
dns	The upstream DNS server of the container.	The DNS configurations that apply to pods. For more information, see Pod's DNS Config. **Note** If a DNS server is specified, the Service name may fail to be parsed.	For more information, see the sample Swarm configuration in dns.	For more information, see the sample Kubernetes configuration in dns.	Migrate manually: After you use Kompose to convert the Swarm compose file, add the dnsConfig parameter to the Kubernetes resource file **-deployment.yaml* of the application.

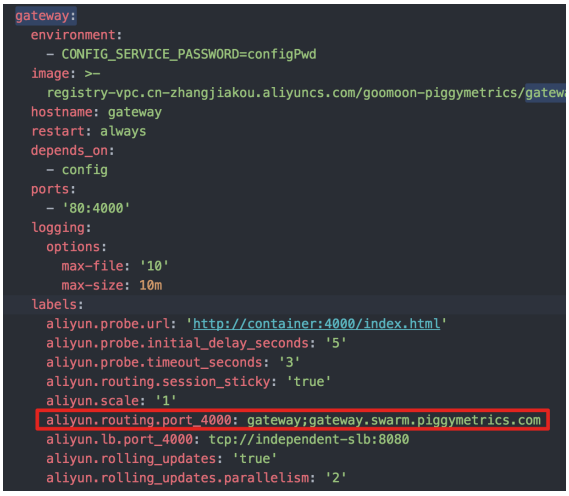
net

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see logtail2-daemonset.yaml.

```
logtail2:
  image: registry.aliyuncs.com/acs-sample/logtail:
  labels:
    aliyun.latest_image: true
    aliyun.global: true
  devices:
    - /dev/mem:/dev/mem
  environment:
    - log_region=cn_hangzhou
  net: host
  extra_hosts:
    - "www.baidu.com:192.168.0.1"
    - "apollo.pro.sample.com:192.168.253.8"
  dns:
    - 100.100.2.136
    - 100.100.2.138
  cap_add:
    - SYS_RAWIO
```

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  annotations:
    aliyun.global: "true"
    aliyun.latest_image: "true"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.y
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: logtail2
  name: logtail2
spec:
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: logtail2
    spec:
      hostNetwork: true
      containers:
      - env:
        - name: log_region
          value: cn_hangzhou
        image: registry.aliyuncs.com/acs-sample/logtail:yunqi
        name: logtail2
        resources: {}
        securityContext:
          capabilities:
            add:
            - SYS_RAWIO
      restartPolicy: Always
```

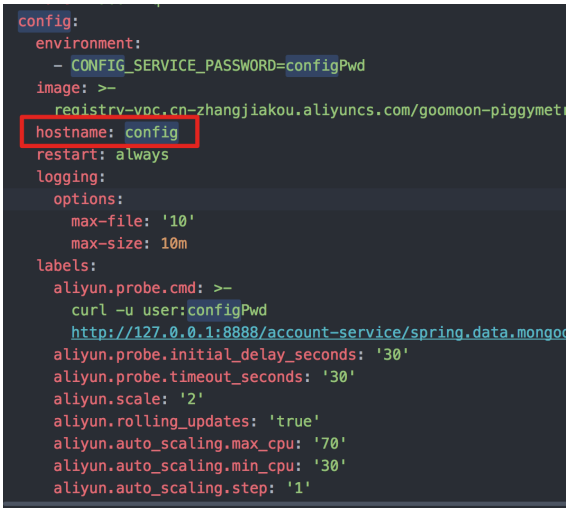
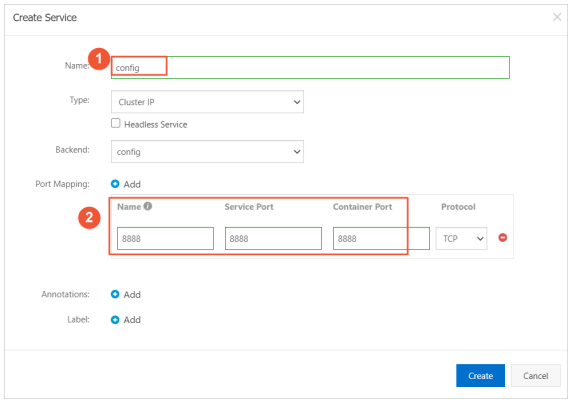
aliyun.routing.port_

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see gateway-ingress.yaml.

```
gateway:
  environment:
    - CONFIG_SERVICE_PASSWORD=configPwd
  image: >-
    registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/gatew
  hostname: gateway
  restart: always
  depends_on:
    - config
  ports:
    - '80:4000'
  logging:
    options:
      max-file: '10'
      max-size: 10m
  labels:
    aliyun.probe.url: 'http://container:4000/index.html'
    aliyun.probe.initial_delay_seconds: '5'
    aliyun.probe.timeout_seconds: '3'
    aliyun.routing.session_sticky: 'true'
    aliyun.scale: '1'
    aliyun.routing.port_4000: gateway;gateway.swarm.piggymetrics.com
    aliyun.lb.port_4000: tcp://independent-slb:8080
    aliyun.rolling_updates: 'true'
    aliyun.rolling_updates.parallelism: '2'
```

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: gateway
  name: gateway
spec:
  rules:
  - host: gateway.c7f537c92438f415b943e7c2f8ca30b3b.cn-zhangjiakou.alicontainer.com
    http:
      paths:
      - backend:
          serviceName: gateway
          servicePort: 80
  - host: gateway.swarm.piggymetrics.com
    http:
      paths:
      - backend:
          serviceName: gateway
          servicePort: 80
status:
  loadBalancer: {}
```

hostname

Sample Swarm configuration	Sample Kubernetes configuration
	

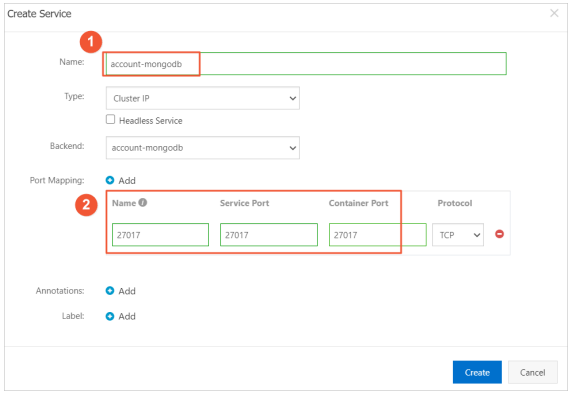
```
config:
  environment:
    - CONFIG_SERVICE_PASSWORD=configPwd
  image: >-
    registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymet
  hostname: config
  restart: always
  logging:
    options:
      max-file: '10'
      max-size: 10m
  labels:
    aliyun.probe.cmd: >-
      curl -u user:configPwd
      http://127.0.0.1:8888/account-service/spring.data.mongo
    aliyun.probe.initial_delay_seconds: '30'
    aliyun.probe.timeout_seconds: '30'
    aliyun.scale: '2'
    aliyun.rolling_updates: 'true'
    aliyun.auto_scaling.max_cpu: '70'
    aliyun.auto_scaling.min_cpu: '30'
    aliyun.auto_scaling.step: '1'
```


Note Create a Service with the same name (marked by 1 as shown in the figure) as the hostname in the ACK console. Use the container port as the Service port.

links

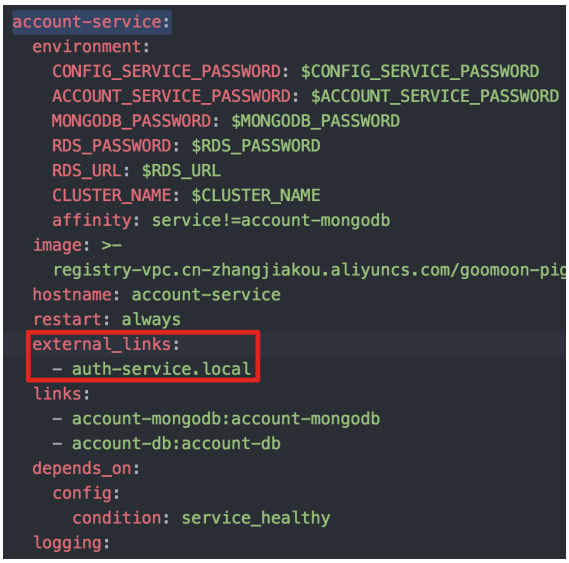
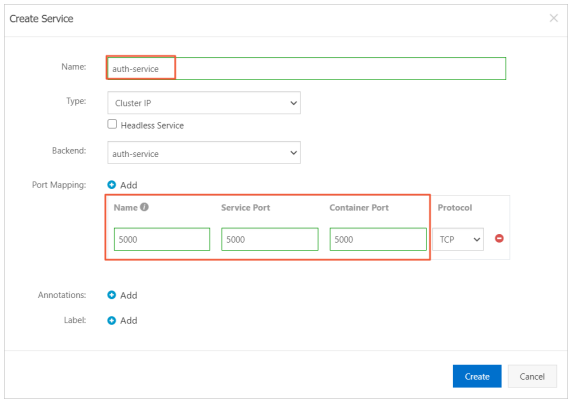
Sample Swarm configuration	Sample Kubernetes configuration

```
account-service:
  environment:
    CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE
    ACCOUNT_SERVICE_PASSWORD: $ACCOUNT_SERVI
    MONGODB_PASSWORD: $MONGODB_PASSWORD
    RDS_PASSWORD: $RDS_PASSWORD
    RDS_URL: $RDS_URL
    CLUSTER_NAME: $CLUSTER_NAME
    affinity: service!=account-mongodb
  image: >-
    registry-vpc.cn-zhangjiakou.aliyuncs.com
  hostname: account-service
  restart: always
  external_links:
    - auth-service.local
  links:
    - account-mongodb:account-mongodb
    - account-db:account-db
  depends_on:
    config:
      condition: service_healthy
```



> **Note** Create Services with the same names as those specified by the links parameter in the ACK console.

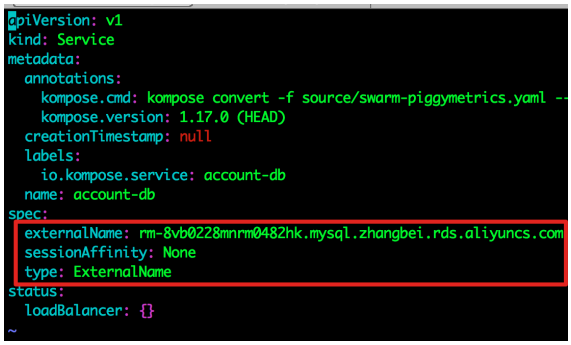
external_links

Sample Swarm configuration	Sample Kubernetes configuration
	 **Note** Create a Service with the same name as the one specified by the external_links parameter in the ACK console.

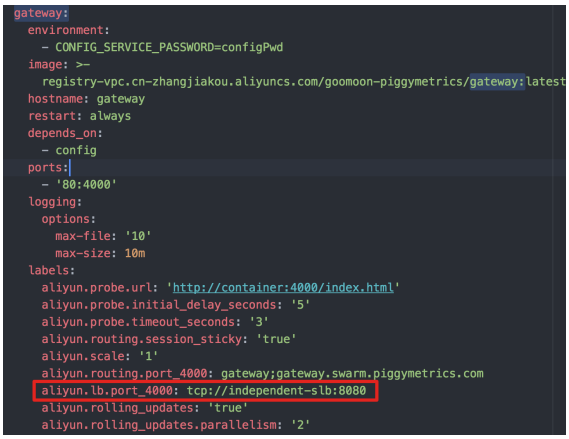
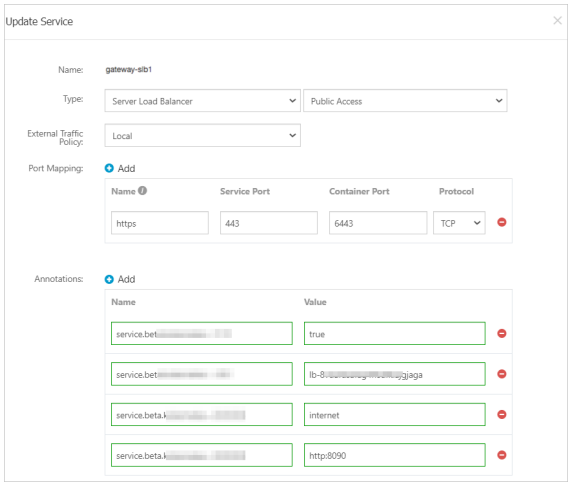
external: ***

The external: *** parameter includes the following fields:

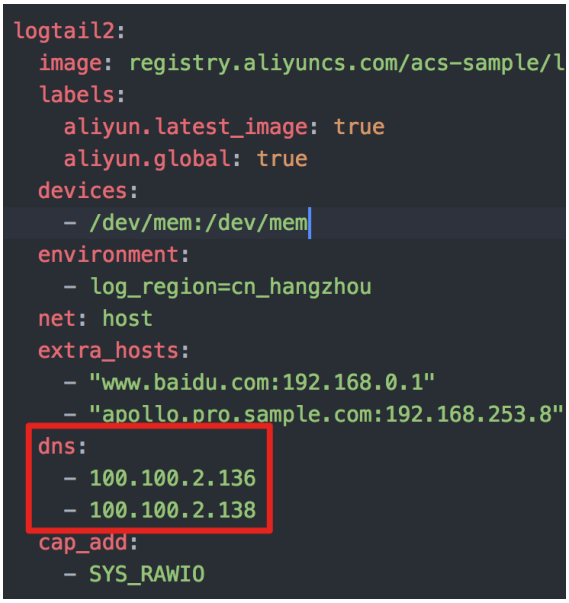
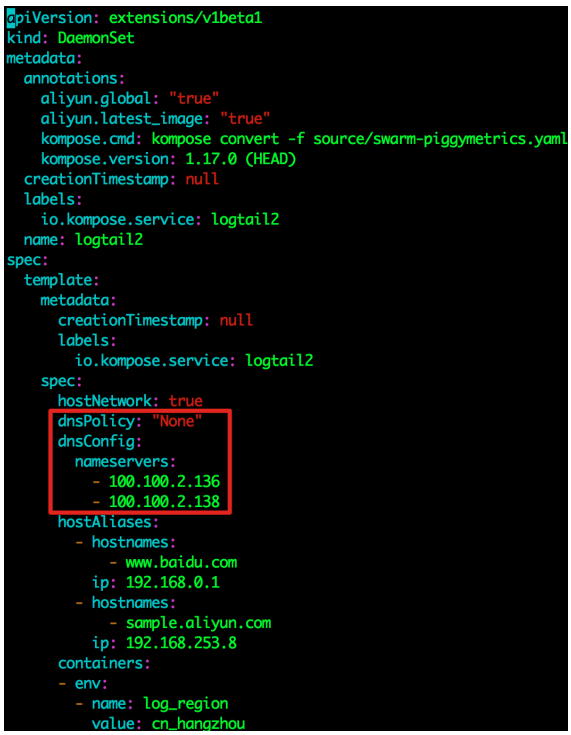
```
external:
     host: $RDS_URL
     ports:
       - 3306
```

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see account-db-service.yaml.

aliyun.lb.port_*

Sample Swarm configuration	Sample Kubernetes configuration
	

dns

Sample Swarm configuration	Sample Kubernetes configuration
	

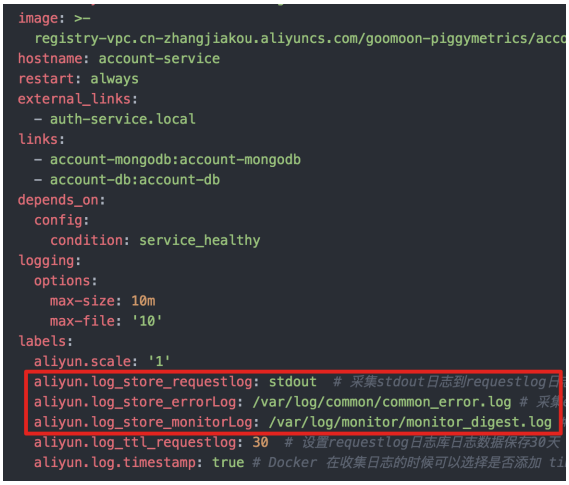
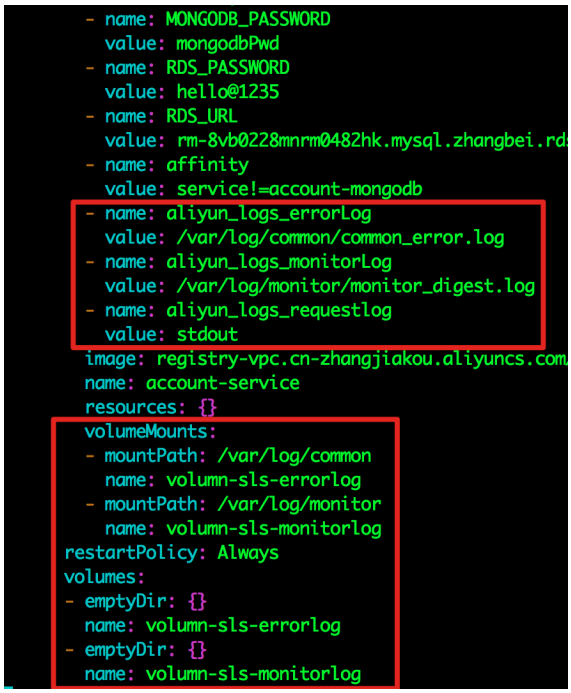
For more information, see logtail2-daemonset.yaml.

11.6.4. Log configuration parameters

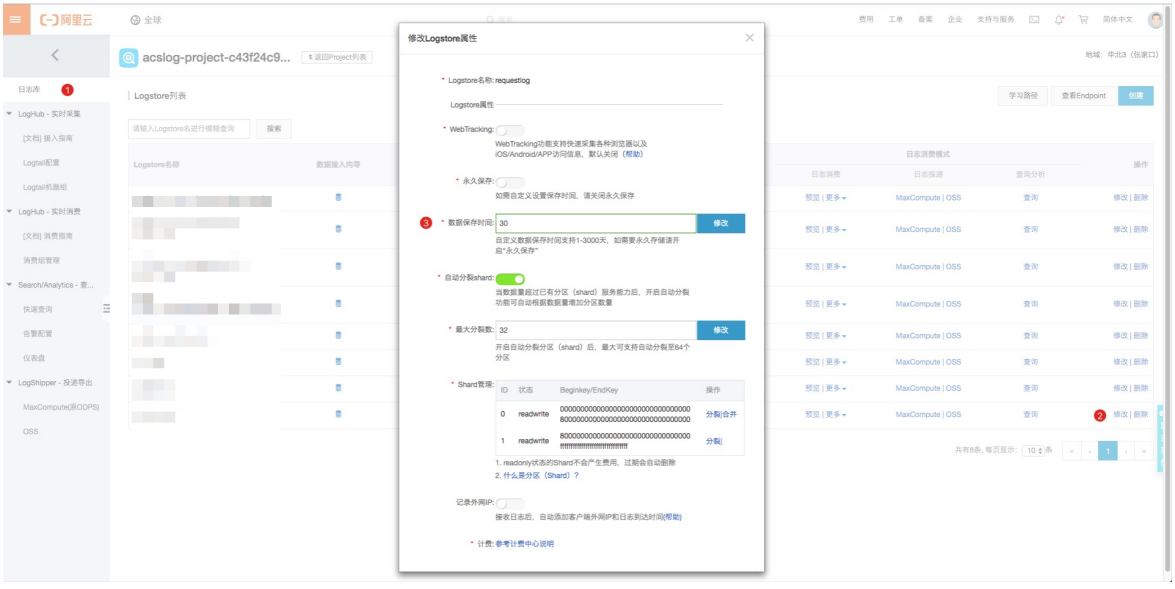
This topic describes the parameters that are related to log configuration.

Swarm parameter	Description	Corresponding Kubernetes parameter	Sample Swarm configuration	Sample Kubernetes configuration	How to migrate
aliyun.log_store_<logstore_name>	The Logstore for storing container logs.	The aliyun_logs environment variables. For more information, see Collect log data from containers by using Log Service.	For more information, see the sample Swarm configuration in aliyun.log_store_<logstore_name>.	For more information, see the sample Kubernetes configuration in aliyun.log_store_<logstore_name>.	Automatically converted by Kompose.
aliyun.log_ttl_<logstore_name>	The initial log retention period of the Logstore. Unit: days. Valid value: 1 to 365. Default is 2 days. This is a custom parameter specific to Log Service. For more information, see Enable Log Service. This parameter specifies the initial value. To change the value, go to the Log Service console.	This parameter does not have a mapping field or configuration in Kubernetes. You can change the log retention period in the Log Service console.	aliyun.log_ttl_requestlog: '30'	For more information, see the sample Kubernetes configuration in aliyun.log_ttl_<logstore_name>.	Migrate manually: 1. Deploy the Kubernetes resource files. Debug the application and check whether the pod status is Running. If yes, a Logstore is generated in Log Service. 2. Log on to the Log Service console, find the target Logstore, and change the log retention period based on your needs.
aliyun.log.timestamp	Specifies whether to add timestamps when Docker collects logs. For more information, see Enable Log Service.	Not supported in Kubernetes.	-	-	No need to migrate.

aliyun.log_store_<logstore_name>

Sample Swarm configuration	Sample Kubernetes configuration
	 For more information, see account-service-deployment.yaml.

aliyun.log_ttl_<logstore_name>

Sample Kubernetes configuration


11.7. Appendix: Parameter configuration examples

account-db-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: account-db
  namespace: default
spec:
  type: ExternalName
  externalName: rm-8vb0228mnrm0482hk.mysql.zhangbei.rds.aliyuncs.com
```

account-service-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.log_store_errorLog: /var/log/common/common_error.log
    aliyun.log_store_monitorLog: /var/log/monitor/monitor_digest.log
    aliyun.log_store_requestlog: stdout
    aliyun.scale: "1"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolu
meClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: account-service
  name: account-service
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: account-service
    spec:
      containers:
      - env:
        - name: ACCOUNT_SERVICE_PASSWORD
          value: accountPwd
        - name: CLUSTER_NAME
          value: K8SCluster
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        - name: MONGODB_PASSWORD
          value: mongodbPwd
        - name: RDS_PASSWORD
```

```
          value: hello@1235
        - name: RDS_URL
          value: rm-8vb0228mnrm0482hk.mysql.zhangbei.rds.aliyuncs.com
        - name: affinity
          value: service!=account-mongodb
        - name: aliyun_logs_errorlog
          value: /var/log/common/common_error.log
        - name: aliyun_logs_monitorlog
          value: /var/log/monitor/monitor_digest.log
        - name: aliyun_logs_requestlog
          value: stdout
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/account-servic
e
        name: account-service
        resources: {}
        volumeMounts:
        - mountPath: /var/log/common
          name: volumn-sls-errorlog
        - mountPath: /var/log/monitor
          name: volumn-sls-monitorlog
      restartPolicy: Always
      volumes:
      - emptyDir: {}
        name: volumn-sls-errorlog
      - emptyDir: {}
        name: volumn-sls-monitorlog
status: {}
```

auth-mongodb-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.probe.url: tcp://container:27017
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolu
meClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: auth-mongodb
  name: auth-mongodb
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: auth-mongodb
    spec:
      nodeSelector:
        group: db
      containers:
      - command:
        - /init.sh
        env:
        - name: MONGODB_PASSWORD
          value: mongodbPwd
        - name: constraint
          value: group==db
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/mongodb
        livenessProbe:
          tcpSocket:
            port: 27017
        name: auth-mongodb
        resources: {}
      restartPolicy: Always
status: {}
```

auth-service-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.latest_image: "true"
    aliyun.log_store_requestlog: stdout
    aliyun.routing.port_5000: auth-service.local
    aliyun.scale: "1"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolu
meClaimOrHostPath
```

```
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: auth-service
  name: auth-service
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: auth-service
    spec:
      containers:
      - args:
        - java
        - -Xmx500m
        - -jar
        - /app/auth-service.jar
        command:
        - java
        - -Xmx200m
        - -jar
        - /app/auth-service.jar
        env:
        - name: ACCOUNT_SERVICE_PASSWORD
          value: accountPwd
        - name: CLUSTER_NAME
          value: K8SCluster
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        - name: MONGODB_PASSWORD
          value: mongodbPwd
        - name: NOTIFICATION_SERVICE_PASSWORD
          value: notificationPwd
        - name: STATISTICS_SERVICE_PASSWORD
          value: statisticsPwd
        - name: aliyun_logs_requestlog
          value: stdout
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/auth-service:l
atest
        name: auth-service
        ports:
        - containerPort: 5000
        resources: {}
        securityContext:
          capabilities:
            add:
            - all
            drop:
            - SETGID
            - SETUID
```

```
          privileged: true
        volumeMounts:
        - mountPath: /data/oss
          name: volume-name-piggymetrics
        - mountPath: /data/nas
          name: vn-nas-piggymetrics
        - mountPath: /data/yunpan
          name: vn-yunpan-piggymetrics
        - mountPath: /var/run/docker.sock
          name: auth-service-claim3
        - mountPath: /data/tmp
          name: auth-service-claim4
      restartPolicy: Always
      volumes:
      - name: volume-name-piggymetrics
        persistentVolumeClaim:
          claimName: volume-name-piggymetrics
      - name: vn-nas-piggymetrics
        persistentVolumeClaim:
          claimName: vn-nas-piggymetrics
      - name: vn-yunpan-piggymetrics
        persistentVolumeClaim:
          claimName: vn-yunpan-piggymetrics
      - hostPath:
          path: /var/run/docker.sock
        name: auth-service-claim3
      - hostPath:
          path: /tmp
        name: auth-service-claim4
status: {}
```

config-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.auto_scaling.max_cpu: "70"
    aliyun.auto_scaling.max_instances: "10"
    aliyun.auto_scaling.min_cpu: "30"
    aliyun.auto_scaling.min_instances: "1"
    aliyun.auto_scaling.step: "1"
    aliyun.latest_image: "false"
    aliyun.probe.cmd: curl -u user:configPwd http://127.0.0.1:8888/account-service/spring.d
ata.mongodb.host
    aliyun.probe.initial_delay_seconds: "30"
    aliyun.probe.timeout_seconds: "30"
    aliyun.rolling_updates: "true"
    aliyun.rolling_updates.parallelism: "2"
    aliyun.scale: "3"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolu
meClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
```

```
    creationTimestamp: null
    labels:
      io.kompose.service: config
    name: config
spec:
    replicas: 10
    strategy:
      type: RollingUpdate
      rollingUpdate:
        maxUnavailable: 2
        maxSurge: 13
    template:
      metadata:
        creationTimestamp: null
        labels:
          io.kompose.service: config
      spec:
        containers:
        - env:
          - name: CONFIG_SERVICE_PASSWORD
            value: configPwd
          image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/config:latest
          imagePullPolicy: Always
          livenessProbe:
            exec:
              command:
              - curl
              - -u
              - user:configPwd
              - http://127.0.0.1:8888/account-service/spring.data.mongodb.host
            initialDelaySeconds: 30
            timeoutSeconds: 30
          name: config
          resources:
            limits:
              cpu: "1"
              memory: "1073741824"
          securityContext:
            capabilities:
              add:
              - all
        restartPolicy: Always
status: {}
```

gateway-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.auto_scaling.max_cpu: "70"
    aliyun.auto_scaling.max_instances: "10"
    aliyun.auto_scaling.min_cpu: "30"
    aliyun.auto_scaling.min_instances: "1"
    aliyun.auto_scaling.step: "1"
    aliyun.log_store_requestlog: stdout
    aliyun.probe.initial_delay_seconds: "50"
    aliyun.probe.timeout_seconds: "30"
    aliyun.probe.url: http://container:4000/index.html
    aliyun.rolling_updates: "true"
    aliyun.rolling_updates.parallelism: "2"
    aliyun.routing.port_4000: gateway;gateway.swarm.piggymetrics.com
    aliyun.scale: "1"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolu
meClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: gateway
  name: gateway
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: gateway
    spec:
      containers:
      - env:
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        - name: aliyun_logs_requestlog
          value: stdout
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/gateway:latest
        livenessProbe:
          httpGet:
            path: /index.html
            port: 4000
          initialDelaySeconds: 50
          timeoutSeconds: 30
        name: gateway
        ports:
        - containerPort: 4000
      restartPolicy: Always
status: {}
```

gateway-ingress.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: gateway
  name: gateway
spec:
  rules:
  - host: gateway.c7f537c92438f415b943e7c2f8ca30b3b.cn-zhangjiakou.alicontainer.com
    http:
      paths:
      - backend:
          serviceName: gateway
          servicePort: 80
  - host: gateway.swarm.piggymetrics.com
    http:
      paths:
      - backend:
          serviceName: gateway
          servicePort: 80
status:
  loadBalancer: {}
```

logtail2-daemonset.yaml

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  annotations:
    aliyun.global: "true"
    aliyun.latest_image: "false"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolu
meClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: logtail2
  name: logtail2
spec:
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: logtail2
    spec:
      hostNetwork: true
      dnsPolicy: "None"
      dnsConfig:
        nameservers:
        - 100.100.2.136
        - 100.100.2.138
      hostAliases:
      - ip: "192.168.0.1"
        hostnames:
        - "www.baidu.com"
      - ip: "192.168.253.8"
        hostnames:
        - "sample.aliyun.com"
      containers:
      - env:
        - name: log_region
          value: cn_hangzhou
        image: registry.aliyuncs.com/acs-sample/logtail:yunqi
        name: logtail2
        resources: {}
        securityContext:
          capabilities:
            add:
            - SYS_RAWIO
      restartPolicy: Always
```

swarm-piggymetrics.yaml

```
version: '2'
services:
  rabbitmq:
    image: >-
      registry-vpc.cn-hangzhou.aliyuncs.com/hd_docker_images/rabbitmq:latest
```

```
    hostname: rabbitmq
    restart: always
    ports:
      - '15672:15672'
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.scale: '1'
      aliyun.routing.port_15672: rabbitmq
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: rabbitmq
  config:
    environment:
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/config:latest
    hostname: config
    restart: always
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.probe.cmd: >-
        curl -u user:configPwd
        http://127.0.0.1:8888/account-service/spring.data.mongodb.host
      aliyun.probe.initial_delay_seconds: '30'
      aliyun.probe.timeout_seconds: '30'
      aliyun.scale: '10'
      aliyun.rolling_updates: 'true'
      aliyun.rolling_updates.parallelism: '2'
      aliyun.auto_scaling.max_cpu: '70'
      aliyun.auto_scaling.min_cpu: '30'
      aliyun.auto_scaling.step: '1'
      aliyun.auto_scaling.max_instances: '10'
      aliyun.auto_scaling.min_instances: '1'
      aliyun.latest_image: 'false'
    oom-kill-disable: true
    memswap_limit: 2000000000
    shm_size: 67108864
    memswap_reservation: 536870912
    kernel_memory: 0
    name: config
    mem_limit: 1073741824
    cpu_shares: 100
    cap_add:
      - all
  registry:
    environment:
```

```
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/registry:latest
    hostname: registry
    restart: always
    depends_on:
      - config
    ports:
      - '8761:8761'
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.scale: '1'
      aliyun.routing.port_8761: eureka
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: registry
  gateway:
    environment:
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/gateway:latest
    hostname: gateway
    restart: always
    depends_on:
      - config
    ports:
      - '80:4000'
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.probe.url: 'http://container:4000/index.html'
      aliyun.probe.initial_delay_seconds: '50'
      aliyun.probe.timeout_seconds: '30'
      aliyun.routing.session_sticky: 'true'
      aliyun.scale: '1'
      aliyun.routing.port_4000: gateway;gateway.swarm.piggymetrics.com
      aliyun.lb.port_4000: tcp://independent-slb:8080
      aliyun.rolling_updates: 'true'
      aliyun.rolling_updates.parallelism: '2'
      aliyun.auto_scaling.max_cpu: '70'
      aliyun.auto_scaling.min_cpu: '30'
      aliyun.auto_scaling.step: '1'
      aliyun.auto_scaling.max_instances: '10'
      aliyun.auto_scaling.min_instances: '1'
      aliyun.log_store_requestlog: stdout  # collect stdout logs to the requestlog Logstore
      aliyun.log_ttl_requestlog: 30  # set the requestlog Logstore to retain logs for 30 da
ys
      aliyun.log.timestamp: true # set whether to add timestamps when Docker collects logs
```

```
      aliyun.log.timestamp: true # set whether to add timestamps when Docker collects logs
    external_links:
      - auth-service.local
    oom-kill-disable: true
    name: gateway
  auth-service:
    environment:
      - MONGODB_PASSWORD=mongodbPwd
      - NOTIFICATION_SERVICE_PASSWORD=notificationPwd
      - STATISTICS_SERVICE_PASSWORD=statisticsPwd
      - ACCOUNT_SERVICE_PASSWORD=accountPwd
      - CONFIG_SERVICE_PASSWORD=configPwd
      - CLUSTER_NAME=$CLUSTER_NAME
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/auth-service:latest
    command: ["java", "-Xmx500m", "-jar", "/app/auth-service.jar"]
    restart: always
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.scale: '1'
      aliyun.log_store_requestlog: stdout  # collect stdout logs to the requestlog Logstore
      aliyun.log_ttl_requestlog: 30  # set the requestlog Logstore to retain logs for 30 da
ys
      aliyun.log.timestamp: false # set whether to add timestamps when Docker collects logs
      aliyun.routing.session_sticky: "true"
      aliyun.routing.port_5000: auth-service.local
      aliyun.latest_image: 'true'
      aliyun.depends: config
    links:
      - auth-mongodb
    volumes:
      - 'volume_name_piggymetrics:/data/oss:rw'
      - 'VN_NAS_PIGGYMETRICS:/data/nas:rw'
      - 'VN_YUNPAN_PIGGYMETRICS:/data/yunpan:rw'
      - '/var/run/docker.sock:/var/run/docker.sock:rw'
      - '/tmp:/data/tmp'
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: auth-service
    hostname: auth-service
    entrypoint:
      - 'java'
      - '-Xmx200m'
      - '-jar'
      - '/app/auth-service.jar'
    cap_add:
      - all
    cap_drop:
      - SETGID
      - SETUID
```

```
    privileged: true
  auth-mongodb:
    environment:
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/mongodb
    restart: always
    labels:
      aliyun.probe.url: tcp://container:27017
    logging:
      options:
        max-size: 10m
        max-file: '10'
    entrypoint: '/init.sh'
    devices:
      - /dev/mem:/dev/mem
  account-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
      ACCOUNT_SERVICE_PASSWORD: $ACCOUNT_SERVICE_PASSWORD
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      RDS_PASSWORD: $RDS_PASSWORD
      RDS_URL: $RDS_URL
      CLUSTER_NAME: $CLUSTER_NAME
      affinity: service!=account-mongodb
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/account-service
    hostname: account-service
    restart: always
    external_links:
      - auth-service.local
    links:
      - account-mongodb:account-mongodb
      - account-db:account-db
    depends_on:
      config:
        condition: service_healthy
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.scale: '1'
      aliyun.log_store_requestlog: stdout # collect stdout logs to the requestlog Logstore
      aliyun.log_store_errorLog: /var/log/common/common_error.log# collect error logs to th
e requestlog Logstore
      aliyun.log_store_monitorLog: /var/log/monitor/monitor_digest.log # collect monitor_di
gest logs to the requestlog Logstore
      aliyun.log_ttl_requestlog: 30  # set the requestlog Logstore to retain logs for 30 da
ys
      aliyun.log.timestamp: true # set whether to add timestamps when Docker collects logs
  account-mongodb:
    environment:
      INIT_DUMP: account-service-dump.js
```

```
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/mongodb
    restart: always
    logging:
      options:
        max-size: 10m
        max-file: '10'
  statistics-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      STATISTICS_SERVICE_PASSWORD: $STATISTICS_SERVICE_PASSWORD
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/statistics-service
    hostname: statistics-service
    external_links:
      - auth-service.local
    restart: always
    depends_on:
      config:
        condition: service_healthy
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.log_store_requestlog: stdout # collect stdout logs to the requestlog Logstore
      aliyun.log_ttl_requestlog: 30  # set the requestlog Logstore to retain logs for 30 da
ys
      aliyun.log.timestamp: true # set whether to add timestamps when Docker collects logs
  statistics-mongodb:
    environment:
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/mongodb
    hostname: statistics-mongodb
    restart: always
    logging:
      options:
        max-size: 10m
        max-file: '10'
  notification-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      NOTIFICATION_SERVICE_PASSWORD: $NOTIFICATION_SERVICE_PASSWORD
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/notification-service
    hostname: notification-service
    external_links:
      - auth-service.local
    restart: always
    depends_on:
```

```
      config:
        condition: service_healthy
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.log_store_requestlog: stdout # collect stdout logs to the requestlog Logstore
      aliyun.log_ttl_requestlog: 30  # set the requestlog Logstore to retain logs for 30 da
ys
      aliyun.log.timestamp: true # set whether to add timestamps when Docker collects logs
  notification-mongodb:
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/mongodb
    hostname: notification-mongodb
    restart: always
    environment:
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
    logging:
      options:
        max-size: 10m
        max-file: '10'
  monitoring:
    environment:
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/monitoring:latest
    hostname: monitoring
    restart: always
    depends_on:
      - config
    ports:
      - '9000:9000'
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.scale: '1'
      aliyun.routing.port_9000: hystrix
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: monitoring
  turbine-stream-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/turbine-stream-service
    hostname: turbine-stream-service
    restart: always
    depends_on:
      config:
```

```
        condition: service_healthy
    ports:
      - '8989:8989'
    logging:
      options:
        max-size: 10m
        max-file: '10'
  logtail2:
    image: registry.aliyuncs.com/acs-sample/logtail:yunqi
    labels:
      aliyun.latest_image: 'false'
      aliyun.global: 'true'
    devices:
      - /dev/mem:/dev/mem
    environment:
      - log_region=cn_hangzhou
    net: host
    extra_hosts:
      - "www.baidu.com:192.168.0.1"
      - "sample.aliyun.com:192.168.253.8"
    dns:
      - 100.100.2.136
      - 100.100.2.138
    cap_add:
      - SYS_RAWIO
```

11.8. Regression testing

This topic describes how to use regression testing to check whether your applications run as expected in a cluster of Container Service for Kubernetes (ACK).

Set a testing domain name

In an ACK cluster, you can configure Service ports to enable communication among Services in the cluster. You can configure Ingresses to enable external access to Services in an ACK cluster.

When you configure an Ingress, you can set multiple domain names such as production domain names and testing domain names. Production domain names are domain names that have obtained Internet Content Provider (ICP) numbers. You must enter the fully-qualified domain name. Testing domain names are provided by ACK. You can enter the prefix of the testing domain name. Perform the following steps to create and configure an Ingress:

1.
2.
3.
4.
5. In the upper-right corner of the Ingresses page, click **Create**.

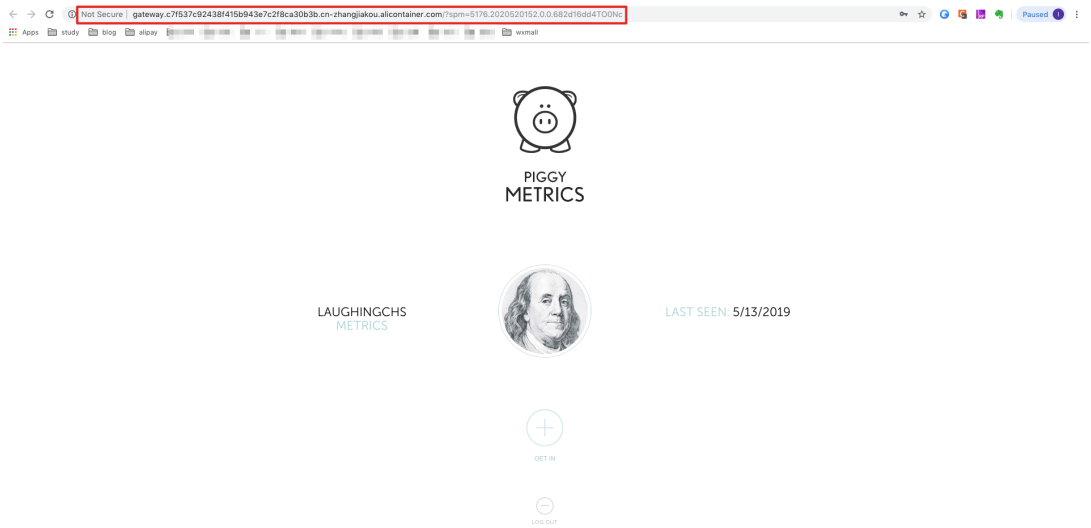
   For more information about how to create and configure an Ingress, see Create an Ingress.
6. After the Ingress is created, you can find its domain name and endpoint on the Ingresses page.

> **Note** The endpoint of the ingress is the public IP address of the Server Load Balancer (SLB) instance that is attached to the ACK cluster.

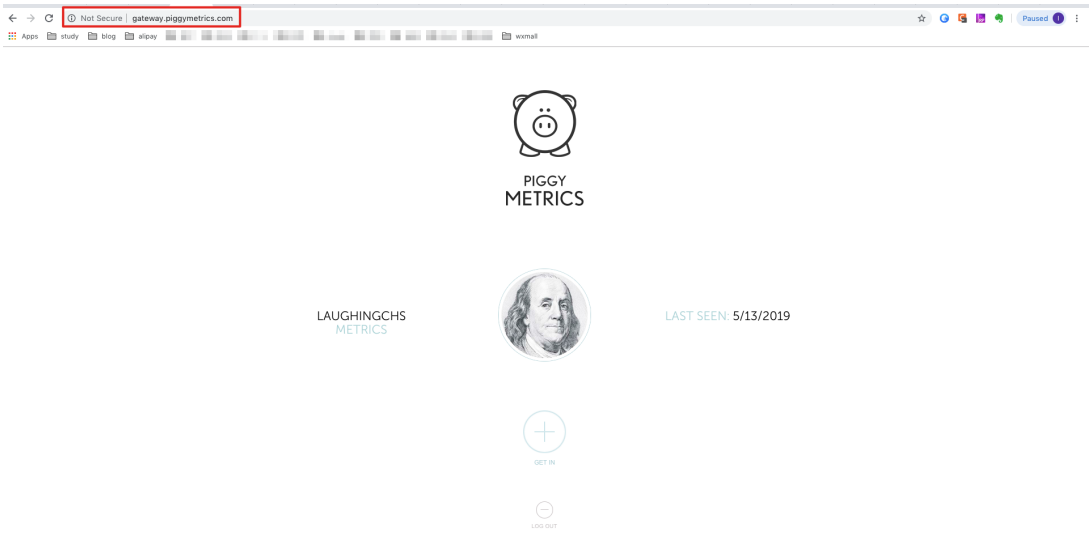
Test whether your applications run as expected

You can use the testing domain name or production domain name to access applications deployed in the related ACK cluster. If you have not updated DNS settings to point the production domain name to the public IP address of the attached SLB instance, you must bind the public IP address to the production domain name on your on-premises machine. The following examples show the domain names that can be used:

Testing domain name



Production domain name



> **Note** Perform regression testing based on the testing domain name or production domain name to check whether your applications run as expected in the newly created ACK cluster.

Check application events

The Logstores for a Swarm cluster and the related ACK cluster are generated based on different rules. For more information, see Migrate log configurations of applications. Therefore, application events of a Swarm cluster and the related ACK cluster are logged in different Logstores under the same Log Service project. During the test process, you can check whether application events are logged to the Logstore created for the ACK cluster and whether the events indicate that applications run as expected.

1. Log on to the Log Service console. In the **Projects** section, click the Log Service project that is used.
2. On the details page of the Log Service project, click the **Logstores** tab. In the Logstore list, you can find the Logstores for the Swarm and ACK clusters.
3. Select the Logstore for the ACK cluster and click **Search** to view the application events. This allows you to check whether the applications run as expected.

Check the monitoring metrics of the applications

If application events can be logged to the Logstore for the ACK cluster, you need to check whether the related monitoring services, such as Cloud Monitor, Application Real-Time Monitoring Service (ARMS), message delivery, and storage services are correctly configured based on the Logstore. For more information, see Log Service - Real-time consumption. You must check whether each log consumption scenario is correctly configured based on your actual requirements.

In this example, an application named swarm-piggymetrics is created to demonstrate how to use host monitoring and log monitoring of Cloud Monitor to monitor applications.

1. View host monitoring metrics. For more information, see How do I view the monitoring data within a specific period of time in the CloudMonitor console?.

   When you create an ACK cluster, you can install the **Cloud Monitor agent**. The Cloud Monitor agent can be used to monitor the system load and network connections on each node in the ACK cluster. It also allows you to check whether the test traffic is received by the newly create ACK cluster.

2. View log monitoring metrics.

   You can use log monitoring or ARMS to conduct fine-grained application monitoring. In this example, a log monitoring task named **findAccountCount_k8s** is created based on the existing monitoring rules of the Swarm cluster to monitor the events of nodes in the ACK cluster. Parsing rules are also created to parse the logged events. You can check whether the test traffic is received by the ACK cluster through charts.

   > **Note** The preceding figure shows the steady increase of account queries over a short time period. This proves that the newly created ACK cluster k8s-piggymetrics-cluster handles test traffic and provides external services.

11.9. Route traffic to a Kubernetes service

11.9.1. Overview

This topic describes how to route a percentage of production traffic to a Container Service for Kubernetes (ACK) cluster to verify the cluster functions that are not included in regression testing.

Prerequisites

The regression testing is completed. For more information, see Regression testing.

Route traffic to the NodePort service

You can use a Server Load Balancer (SLB) instance attached to a Container Service for Swarm cluster or an SLB instance attached to an ACK cluster to route traffic in the following ways:

- Route traffic by using an SLB instance attached to an ACK cluster: Add the endpoint of the SLB instance to the client or Domain Name System (DNS) record. This requires you to update the client configurations or modify the DNS settings.
  - High cost of traffic redirection

    You must update the client configurations or modify the DNS settings.
  - High risk of rollback
    - If issues are found in the ACK cluster, you must update the client configurations or modify the DNS settings again.
    - It may take a long time period to update the client configurations or cache the DNS settings. This poses negative impacts on the service performance.
- Route traffic by using an SLB instance attached to a Container Service for Swarm cluster: Add your Kubernetes service to a VServer group of the SLB instance that is attached to the Container Service for Swarm cluster. Then, set weights to specify the percent of traffic to be routed to the service.
  - Low cost of traffic redirection
    - You only need to configure the SLB instance attached to the Container Service for Swarm cluster in the SLB console.
    - You can set weights to specify the percentage of traffic to be routed to different services in the SLB console.
  - Low risk of rollback
    - You only need to configure the SLB instance attached to the Container Service for Swarm cluster in the SLB console.
    - The new configurations immediately take effect.

Procedure

1. Create a NodePort service in an ACK cluster.
2. Verify the NodePort Service.
3. Configure the SLB instance attached to the Swarm cluster.

How to roll back

After production traffic is routed to the Kubernetes service, you can modify the VServer group to which you add the Kubernetes service when issues are found. You can set the weight of the Elastic Compute Service (ECS) instance where the service runs to *0* or remove the ECS instance.

Procedure

1. Log on to the SLB console. On the **Instances** page, click the SLB instance that you want to manage.

2. On the page that appears, click the **VServer Groups** tab. Select the VServer group to which you add the Kubernetes service and click **Edit** in the Actions column.
3. On the **Edit VServer Group** page, find the ECS instance where the Kubernetes service runs, click **Delete** in the Actions column, and click **OK**.

Summary

You can configure **VServer groups** of an SLB instance that is attached to a Container Service for Swarm cluster to route a percentage of production traffic to the NodePort service deployed in an ACK cluster. After traffic is redirected, you can verify the functions of the service.

Compared with traffic redirection by updating the client configurations or modifying the DNS settings, this method only requires you to modify the SLB configurations in the SLB console. The configuration changes take effect immediately and you can specify the percentage of traffic to be sent to different services. You can also roll back when an error occurs. This reduces the cost of migration and minimizes the risk.

11.9.2. Create a NodePort service in an ACK cluster

This topic describes how to create a NodePort service in the Kubernetes-piggymetrics-cluster cluster. The service is used to handle production traffic forwarded from the SLB instance attached to a Container Service for Swarm cluster.

Prerequisites

A managed Kubernetes cluster is created. For more information, see Create an ACK managed cluster.

Procedure

1. Log on to the Container Service for Kubernetes console. In the left-side navigation pane, choose **Ingresses and Load Balancing > Services**. On the Services page, find the Kubernetes-piggymetrics-cluster cluster and click **Create** in the upper-right corner.
2. In the **Create Service** dialog box, set the required parameters and click **Create**. For more information, see Manage Services.

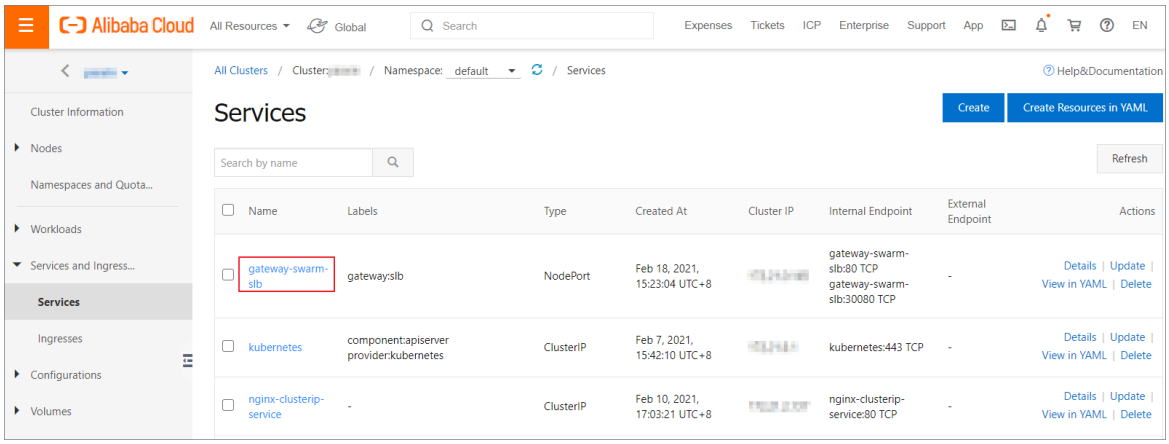
> **Note**
>
> - Name: Enter the service name. gateway-swarm-slb is entered in this example.
> - Type: Select the service type, which specifies how the service is accessed. *Node Port* is selected in this example.
>
>   Node Port: enables access to the service through the IP address and static port (NodePort) on each node. A NodePort service can be used to route requests to a ClusterIP service. The ClusterIP service is automatically created by the system. You can access the NodePort service from outside the cluster by sending requests to `<NodeIP>:<NodePort>`.
> - Backend: Select the backend instance to be bound to the service. The created gateway service is selected in this example.
> - Port Mapping: Set the service port and container port. The container port must be the same as that exposed by the backend pod.
>
>   The node port range is from 30000 to 32767.

11.9.3. Verify the NodePort Service

In the Kubernetes-piggymetrics-cluster cluster, the NodePort Service is used to handle production traffic forwarded from the Server Load Balancer (SLB) instance that is attached to a Container Service for Swarm cluster. This topic describes how to verify the NodePort Service.

1.
2.
3.
4.
5. Check whether the Service is created.

   In the list of Services, you can verify that the type of the gateway-swarm-slb Service is *NodePort*.

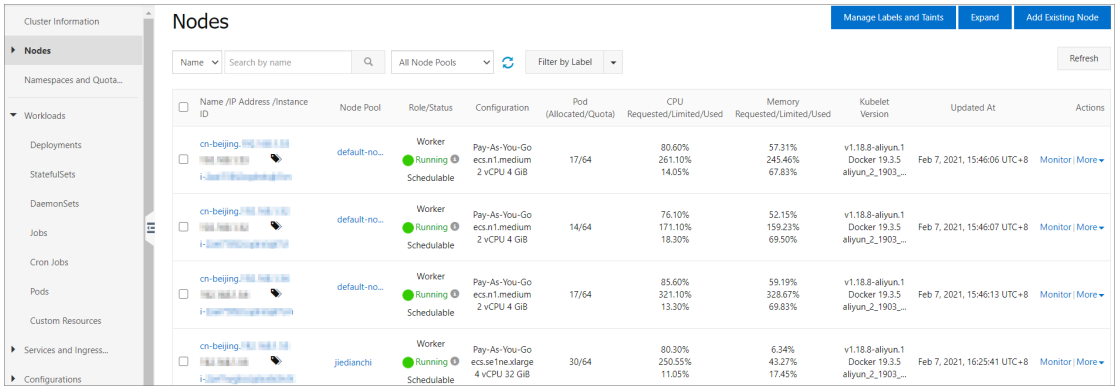


6. Check whether the Service functions as expected.

You can log on to an Elastic Compute Service (ECS) instance in the Container Service for Kubernetes (ACK) cluster. Then, send a request to the NodePort Service to check whether the Service is accessible. In this example, the Service port is 30080 and the IP address of the ECS instance is 192.168.XX.X0.

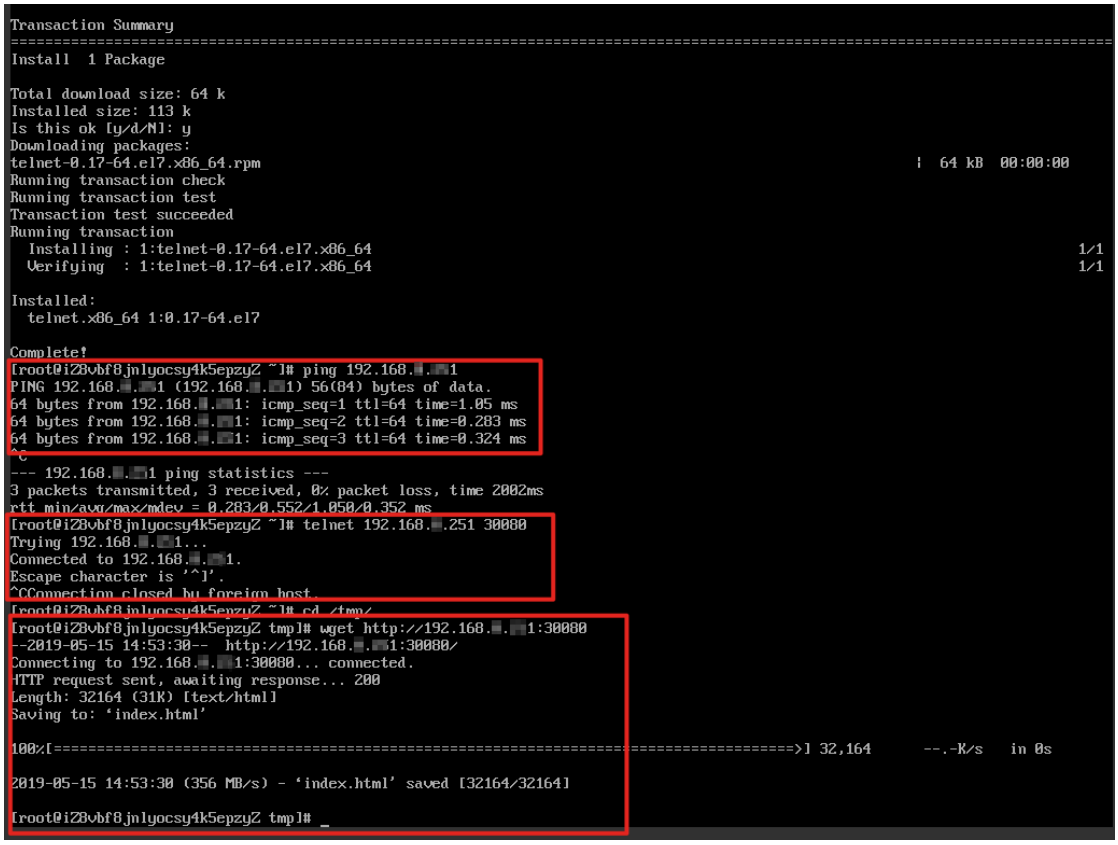
i.

ii. On the **Nodes** page, select the Kubernetes-piggymetrics-cluster cluster and click the ID of the ECS instance that you want to manage.



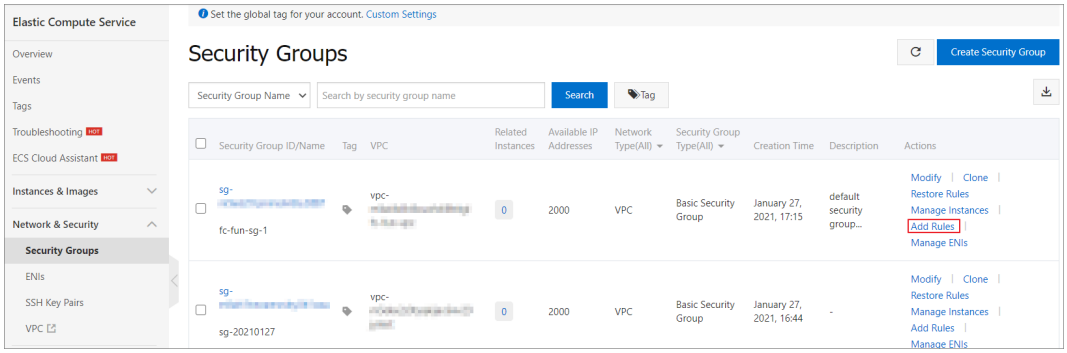
iii. On the Instance Details page, click **Connect** to log on to the instance.

iv. After you log on to the ECS instance 192.168.XX.X0, run the **ping**, **telnet**, and **wget** commands to access 192.168.XX.X1:30080 to check whether the NodePort Service functions as expected.

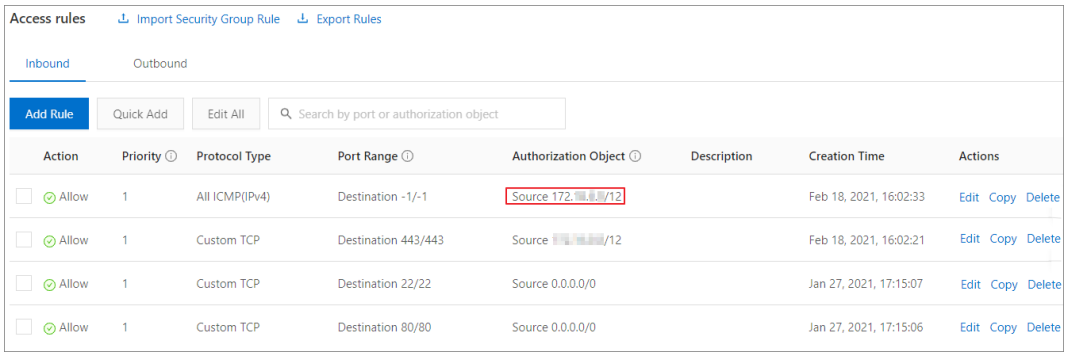


ACK automatically creates a security group for ECS instances in the same cluster. By default, only IP addresses within the pod CIDR blocks can access the ECS instances. You cannot access the ECS instances from outside the cluster. This limit is not applicable when you access the ECS instances from outside the cluster through an SLB instance. In this example, an ECS instance in the ACK cluster is used to verify the NodePort Service. To view the security groups, perform the following operations:

a. Log on to the ECS console. In the left-side navigation pane, choose **Network & Security > Security Groups**. On the Security Groups page, find the security group that you want to manage and click **Add Rules** in the Actions column.



b. On the **Security Group Rules** page, click the **Inbound** tab and view the IP addresses and ports included in the security group.



After you verify that the NodePort Service functions as expected, you must modify the listener of the SLB instance in the Container Service for Swarm cluster and attach the NordPort Service to the SLB instance. For more information, see Configure the SLB instance attached to the Swarm cluster.

11.9.4. Configure the SLB instance attached to the Swarm cluster

This topic describes how to configure the SLB instance attached to the Swarm cluster to redirect a percentage of traffic to the related cluster of Container Service for Kubernetes (ACK).

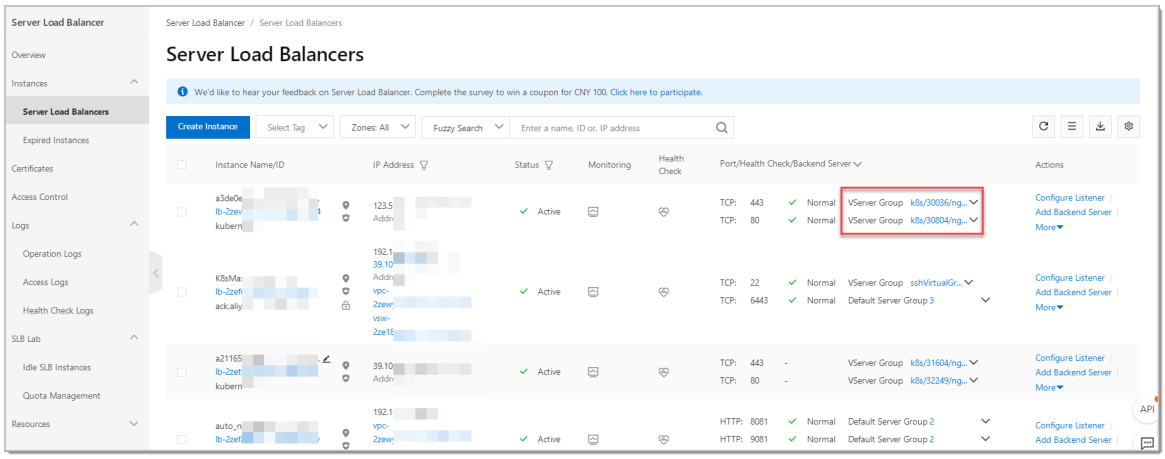
After a Server Load Balancer (SLB) instance receives requests, the requests are forwarded to the backend servers of the SLB instance. To mount a server to the listener of an SLB instance, you can add the server to one of the following groups: **default server group**, **primary/secondary server group**, or **VServer group**.

The default server group and primary/secondary server group require that servers in the same group use the same listening port. However, servers in a Swarm cluster use port 9080 to receive traffic from the SLB instance. The previously created NodePort Service can use only a port from 30000 to 32767 to receive traffic from the SLB instance. In this case, you can add the server that hosts the NodePort Service to only a VServer group to route traffic to the Service. Perform the following steps:

Check the backend servers of the attached SLB instance

Check whether **VServer groups** are created to mount servers to the SLB instance that is attached to the Swarm cluster.

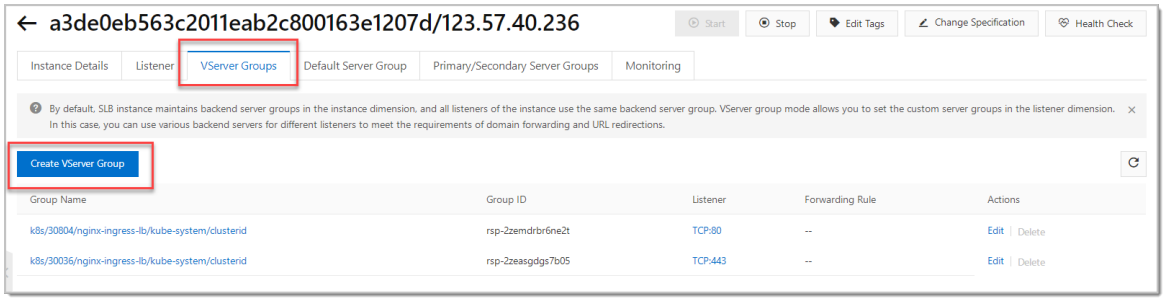
1. Log on to the SLB console. In the left-side navigation pane, choose **Instances > Instances**.
2. On the **Instances** page, find the attached SLB instance and check its backend server type.



   - If the backend server type is *default server*, go to the next step.
   - If the backend server type is *VServer group*, go to the Configure the listener and switch traffic to the VServer group step.

Create a VServer group

1. On the **Instances** page, click the attached SLB instance. Click the **Listener** tab. Then, click the **VServer Groups** tab.
2. Click **Create VServer Group** to go to the **Create VServer Group** page.



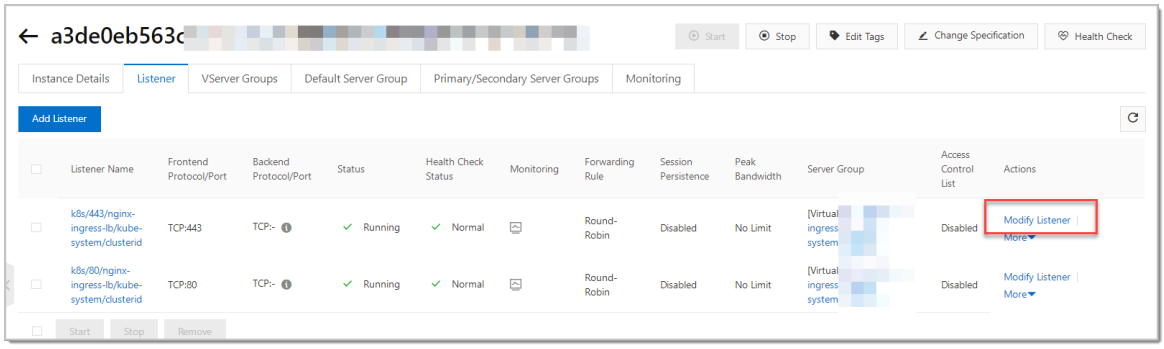
3. Click **Add** to open the **My Servers** dialog box.
4. Set the parameters and click **Create** to create a VServer group. For more information, see Create a vServer group.

> **Note**
>
> - When you add servers to the VServer group, select all servers in the Swarm cluster. This ensures that the backend servers can handle all production traffic when you switch traffic to this VServer group. You can add one or more servers in the related ACK cluster to the VServer group.
> - The port and weight settings of the servers from the Swarm cluster must be the same as those of the previously configured **default server group** or **primary/secondary server group**.

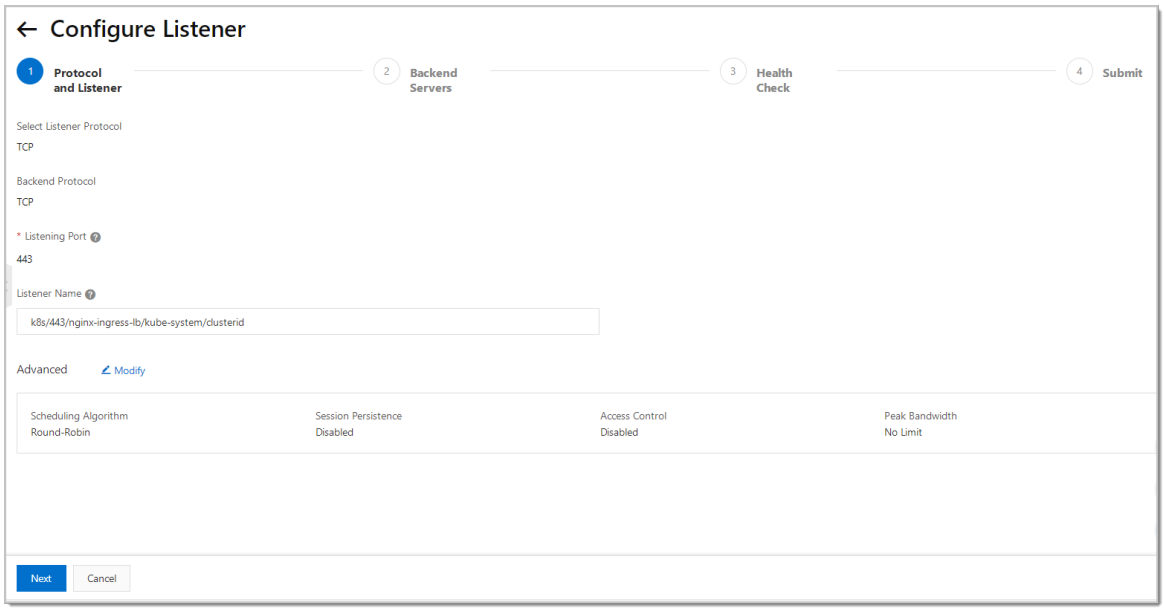
Configure the listener and switch traffic to the VServer group

After the VServer group is created, you must configure the listener to switch traffic from the **default server group** or **primary/secondary server group** to the VServer group.

1. On the **Instances** page, click the attached SLB instance to go to the **Listener** tab. On the **Listener** tab, find the port that you want to configure and click **Modify Listener** in the Actions column.



2. On the **Configure Listener** page, modify the configurations in the **Protocol and Listener** step and click **Next**.



3. In the **Backend Servers** section, click **VServer Group** and select *swarm&K8s* as the VServer group. Check whether the public and internal IP addresses and port settings of servers from the Swarm cluster are correct. Then, click **Next** to save the modifications.

4. Modify the configurations in the **Health Check** step based on your requirements and click **Next**.
5. Check whether all configurations are modified based on your requirements in the **Confirm** step and click **Submit**.
6. Go to the **Listener** tab. The server group of the related listener is now changed to *[Virtual]swarm&k8s*.

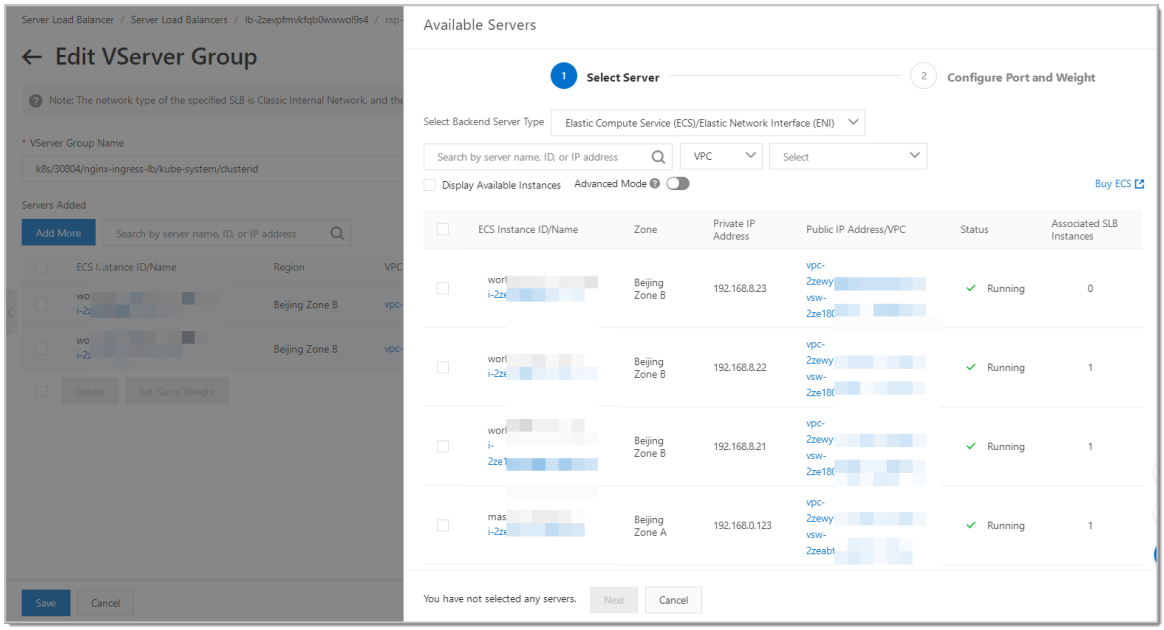
Check whether requests are properly handled

Monitor the service performance for a period of time and check whether requests are properly handled. You can use **CloudMonitor > Host Monitoring**, **CloudMonitor > Log Monitoring**, or other monitoring services to check how inbound traffic is handled.

Add servers in the ACK cluster to the VServer group

If the backend server type of the SLB instance is *VServer group*, you need to add servers in the ACK cluster to the VServer group. The routes a percentage of production traffic to the ACL cluster.

1. On the **Instances** page, click the attached SLB instance to go to the **Listener** tab. On the **Listener** tab, find the port that you want to use and click the related server group in the Server Group column.
2. On the **VServer Groups** tab, find the VServer group that you want to use and click **Edit** in the Actions column. On the **Edit VServer Group** page, click **Add More** to add servers of the ACK cluster to the VServer group.



> **Note**
>
> - You can add one or more servers of the ACK cluster to the VServer group. This allows you to implement canary releases based on your requirements.
> - The port settings of the servers from the ACK cluster must be the same as those of the NodePort Service.
>
> The port range is 30000 to 32767. In this example, port 30080 is used, which points to the gateway-swarm-slb Service in the ACK cluster.
> - You can also set weights to specify the percentage of traffic sent to the ACK cluster.

Check whether traffic is routed to the ACK cluster

Monitor the service performance for a period of time and check whether requests are properly handled. Then, check whether traffic is routed to the ACK cluster based on the weight settings of servers in the VServer group.

> **Note** In this topic, the SLB instance attached to the Swarm cluster is configured to route a percentage of traffic to the related ACK cluster. This way, you can check whether applications run as expected in the ACK cluster. Meanwhile, the Swarm cluster still handles production traffic. For more information about how to delete the Swarm cluster, see Delete a Swarm cluster.

11.10. Switch all production traffic to an ACK cluster

This topic describes how to switch all production traffic from a Container Service for Swarm cluster to a Container Service for Kubernetes (ACK) cluster after you validate the functions of the ACK cluster. You can switch traffic by modifying the DNS record or updating the client configurations.

Prerequisites

To reduce the risk of service disruption, make sure that the services in the ACK cluster are running as expected before you switch all production traffic to the cluster. You can perform regression testing to verify the functions of the services by using the test domain name or a production domain name. The production domain name must have an Internet Content Provider (ICP) number. For more information, see Regression testing.

Switch traffic by modifying the DNS record

If your client connect to the services through a production domain name, you can modify DNS records to switch production traffic to the ACK cluster without service disruption.

Procedure

1. Obtain the IP address of the default Server Load Balancer (SLB) instance attached to the ACK cluster. The IP address is *39.XX.XX.112* in this example.

   Log on to the ACK console. In the left-side navigation pane, choose **Ingresses and Load Balancing > Ingresses**. On the Ingress page, select the *k8s-piggymetrics-cluster* cluster and the *default* namespace. The endpoint of the SLB instance attached to the ACK cluster is *39.XX.XX.112*.

2. Add the IP address *39.XX.XX.112* as a backend IP address for the production domain name *piggymetrics.com* at your DNS provider.
3. Monitor the service performance and check whether traffic sent to the ACK cluster is properly handled. For more information, see Regression testing.

   > **Note** You can also use your own monitoring system to check whether services in the ACK cluster are running as expected.

4. After you make sure that traffic sent to the ACK cluster is properly handled, log on to the platform of your DNS provider that hosts the production domain name. Remove the IP address of the SLB instance attached to the Container Service for Swarm cluster from the backend IP address list.

   In this example, the IP address of the SLB instance attached to the Container Service for Swarm cluster is *39.XX.XX.230*. Remove the IP address from the backend IP address list for the production domain name piggymetrics.com.

Switch traffic by updating the client configurations

The test domain name is related to the cluster ID. Therefore, the test domain name of the Container Service for Swarm cluster is different from that of the ACK cluster. The client may access the Container Service for Swarm cluster through the test domain name or use the host IP to access the IP address of the SLB instance attached to the cluster. In this case, you cannot switch traffic to the ACK cluster without service disruption. We recommend that you perform the switchover during off-peak hours.

Procedure

1. Obtain the IP address of the SLB instance that is attached to the ACK cluster. The IP address is *39.XX.XX.112* in this example.

   Log on to the ACK console. In the left-side navigation pane, choose **Ingresses and Load Balancing > Ingresses**. On the Ingress page, select the *k8s-piggymetrics-cluster* cluster and the *default* namespace. The endpoint of the SLB instance attached to the ACK cluster is *39.XX.XX.112*.
2. Add *39.XX.XX.112* to the list of IP addresses that the client accesses by updating the configuration or code of the client.
3. Monitor the service performance and check whether traffic sent to the ACK cluster is properly handled. For more information, see Regression testing.

   > **Note** You can also use your own monitoring system to check whether services in the ACK cluster are running as expected.

4. After you make sure that traffic sent to the ACK cluster is properly handled, remove the IP address of the SLB instance attached to the Container Service for Swarm cluster from the list of IP addresses that the client accesses. You can complete this task by updating the configuration or code of the client.

   In this example, the IP address of the SLB instance attached to the Container Service for Swarm cluster is *39.XX.XX.230*. Remove this IP address from the list of IP addresses that the client accesses.

11.11. Delete a Swarm cluster

This topic describes how to delete a cluster of Container Service for Swarm.

Check whether user traffic is routed to the Swarm cluster

1. Check whether user traffic is sent to the Server Load Balancer (SLB) instance.

   Before you delete a Swarm cluster, make sure that no user traffic is sent to the SLB instance that is attached to the Swarm cluster. SLB allows you to view monitoring data by SLB instance or by listener. You can perform the following steps to view monitoring data and check whether user traffic is sent to the SLB instance:

   i. Log on to the SLB console.
   ii. In the left-side navigation pane, choose **CLB (FKA SLB) > Instances**.
   iii. On the **Instances** page, click the SLB instance that is attached to the Swarm cluster. Then, you are redirected to the details page of the SLB instance.
   iv. On the **Monitoring** tab, you can view monitoring data and create alert rules.

2. Check whether user traffic is forwarded to the applications that are deployed in the Swarm cluster.

   We recommend that you monitor the applications that are deployed in the Swarm cluster for a period of time and make sure that no user traffic is received before you delete the Swarm cluster. The Swarm cluster may receive network traffic due to client updates or Domain Name Service (DNS) caching. In this case, if you delete the Swarm cluster, your business may be interrupted.

Delete a Swarm cluster

After you make sure that the related Kubernetes cluster is handling external requests as expected and no user traffic is received by the Swarm cluster, you can delete the Swarm cluster. Perform the following steps to delete the Swarm cluster:

1. Back up the configurations of the Swarm cluster.

   To ensure that you can restore the Swarm cluster in case an unknown error occurs in the Kubernetes cluster, we recommend that you back up at least the following configuration items of the Swarm cluster:

   - Node labels, network settings, volume settings, ConfigMap settings, and SLB configurations.
   - Orchestration templates and log files of the applications that are deployed in the Swarm cluster.

2. Release resources that are used by the Swarm cluster.

   > **Notice** When you delete a Swarm cluster, all resources that are used by the cluster are automatically released or deleted, such as Elastic Compute Service (ECS) instances, SLB instances, applications, and Services. Exercise caution when you delete the Swarm cluster.

   i. Log on to the Container Service for Kubernetes (ACK) console.
   ii. In the left-side navigation pane, choose **Clusters**.
   iii. Choose **More** > **Delete** in the Actions column of the Swarm cluster that you want to delete. For more information, see Delete a cluster.

3. Release disks.

   A disk can be attached only to one ECS instance. If you want to migrate the data stored in a data disk to the Kubernetes cluster, you can stop the ECS instance that uses the disk and then migrate the ECS instance to the Kubernetes cluster. For more information, see Migrate cluster configurations.

If you want to migrate a data disk to the Kubernetes cluster without the data stored in the disk, you can create a new disk for the Kubernetes cluster. After the Swarm cluster is deleted, the disk that is mounted to the Swarm cluster is not automatically released. The disk changes to the **Unattached** state.

> **Note** To avoid data loss, we recommend that you back up the data stored in a disk before you release the disk.

To release a disk that is mounted to a Swarm cluster, perform the following operations in the ECS console:

i. Log on to the ECS console.
ii. In the left-side navigation pane, choose **Storage & Snapshots > Disks**.
iii. Choose **More > Release** in the Actions column of the disk that you want to release.