Alibaba Cloud
Aliyun Container for Kubernetes

Best Practices

Issue: 20190613

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 due to product version upgrades , adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall 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 the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults " and "as available" basis. 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 bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, 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 content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, 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 the Alibaba Cloud website, product programs, or content 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 contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
	This warning information 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.
	This warning information 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 restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	 Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus , page names, and other UI elements.	Click OK.
`Courier font`	It is used for commands.	Run the `cd / d C :/ windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*

Style	Description	Example
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich {stand | slave}`

Contents

1 Cluster

1.1 ECS instance selection and cluster configurations

1.1.1 Select ECS instances

This topic describes the recommend ECS instances for creating a Kubernetes cluster.

Overall cluster ECS instance selection

Low performance ECS instances have the following disadvantages:

- The Worker nodes that run on low performance ECS instances can use only a limited number of network resources.
- If one container consumes most of the resources provided by a low performance ECS instance, the remaining resources become idle because they are insufficient for operations such as creating new containers or restoring failed containers. If you set multiple low performance ECS instances, an excessive amount of resources will be wasted.

High performance ECS instances have the following advantages:

- Large network bandwidth is available. For applications that require large bandwidth, resource usage is high.
- More container communication occurs within one ECS instance, reducing data transmission over networks.
- Images can be more efficiently pulled. For a cluster that uses high performance ECS instances, it only requires one attempt to pull an image and the pulled image then can be used by multiple containers. By contrast, for a cluster that uses low performance ECS instances, multiple attempts must be made to pull an image. Furthermore, scaling a cluster that uses low performance ECS instances takes much longer to perform.

Select the Master node specification

For Kubernetes clusters created through Alibaba Cloud Container Service, core components such as etcd, kube-apiserver, and kube-controller run on Mater nodes. These core components are critical for ensuring cluster stability. Generally, large clusters have higher requirements on the Master node specification.

 Note:

You can determine your cluster size by considering the following factors: the number of nodes, the number of pods, deployment frequency, and the number of visits. In this topic, only the number of nodes is used to determine the size of a cluster.

To select the Master node specification of a cluster of the standard size, see the following table. However, you can select the lower performance Master nodes for clusters in a test environment. The specifications recommended in the following table are designed to keep Master node loads low.

Number of nodes	Master node specification
1 to 5	4 cores, 8 GiB (We recommend that you do not select 2 cores with 4 GiB.)
6 to 20	4 cores, 16 GiB
21 to 100	8 cores, 32 GiB
100 to 200	16 cores, 64 GiB

Select the Worker node specification

- Determine the number of cores required by the cluster and the allowed core failure ratio.

  For example, assume a cluster has 160 cores in total. If the allowed core failure ratio is 10%, you must select at least ten 16-core ECS instances and ensure that the upper limit of the cluster load is 160*90%=144 cores. If the allowed core failure ratio is 20%, you must select at least five 32-core ECS instances and ensure that the upper limit of the cluster load is 160*80%=128 cores. In either of these two cases, if one ECS instance fails, the remaining ECS instances can still support the cluster services.
- Determine the CPU:memory ratio. If you run applications that consume large amount of memory resource, for example, Java applications, we recommend that you select an ECS instance with a CPU:memory ratio of 1:8.

Select the ECS Bare Metal Instance

We recommend that you select an ECS Bare Metal (EBM) Instance in the following two scenarios:

- You cluster requires 1000 cores for daily operation. In this case, you can use about ten or eleven EBM instances to build your cluster because one EBM instance has a minimum of 96 cores.
- You want to quickly scale out a large number of containers. For example, assume that you are prepared for a popular E-commerce product promotion. To handle the expected large amount of traffic, you can add EBM instances to your cluster because a single EBM instance can run multiple containers.

EBM instances provide the following benefits to your cluster:

- Ultra-high network performance. Remote Direct Memory Access (RDMA) technology is used. Furthermore, the Terway plugin is designed for you to get the most from your hardware and provides a container bandwidth higher than 9 Gbit/s across hosts.
- Zero jitter computing performance. EBM instances use chips developed by Alibaba Cloud to replace Hypervisor, meaning virtualization overhead or resource preemption concerns are no longer issues.
- High security. EBM instances use physical level encryption, support the Intel SGX encryption, provide a reliable computing environment, and support blockchain applications.

1.1.2 Recommended Kubernetes cluster configurations to run highly reliable applications

To help you guarantee that your applications stably and reliably run in Kubernetes, this topic describes the recommended Kubernetes cluster configurations.

Set the disk type and size

Select the disk type

- We recommend that you select the SSD disk type.
- For Worker nodes, we recommend that you select the Attach Data Disk check box when you create a cluster. This disk is provided exclusively for the `/ var / lib / docker` file to store local images. It is designed to allow the root disk to store a massive number of images. After your cluster has run for a period, many images you no longer require remain stored. To quickly solve this, we recommend that you take the machine offline, rebuild this disk, and then bring the machine back online.

Set the disk size

Kubernetes nodes require a large disk space because the Docker images, system logs , and application logs are stored in the disk. When creating a Kubernetes cluster, you need to consider the number of pods on each node, the log size of each pod, the image size, the temporary data size, and the space required for system reserved values.

We recommend that you reserve a space of 8 GiB for the ECS instance operation system because the operation system requires a disk space of at least 3 GiB. Kubernetes resource objects then use the remaining disk space.

Whether to build Worker nodes when creating your cluster

When you create a cluster, you can select either of the following Node Type:

- Pay-As-You-Go, indicates that you can build Worker nodes when creating a cluster.
- Subscription, indicates that you can purchase ECS instances as needed and add the instances to your cluster after you create you cluster.

Configure your cluster network settings

- If you want to connect your cluster with services outside Kubernetes, for example , Relational Database Service (RDS), we recommend that you use an existing VPC , rather than create a VPC. This is because VPCs are logically isolated. You can create a VSwitch and add the ECS instances that run Kubernetes to the VSwitch.
- You can select the Terway network plugin or the Flannel network plugin when creating a Kubernetes cluster. For more information, see *Do I select the Terway or Flannel plugin for my Kubernetes cluster network?*.
- We recommend that you do not set a small CIDR block of the pod network that only supports a minimal number of nodes. The CIDR block setting of the pod network is associated with the Pod Number for Node setting in Advanced Config. For example, if you set the CIDR block of the pod network to X.X.X.X/16, it means that the number of IP addresses assigned to your cluster is 256*256. Additionally, if you set the number of pods on each node to 128, it means that the maximum number of nodes supported by your cluster is 512.

Use multiple zones

Alibaba Cloud supports multiple regions and each region supports multiple zones. Zones are physical areas that have independent power grids and networks within a region. Using multiple zones enables disaster recovery across areas, but increases

network latency. When creating a Kubernetes cluster, you can choose to create a multi-zone cluster. For more information, see *Create a multi-zone Kubernetes cluster*.

Claim resources for each pod

When you use a Kubernetes cluster, a common problem is that too many pods are scheduled to one node. This scheduling of pods overloads the node, making it unable to provide services.

We recommend that you specify the resource request parameter and the resource limit parameter when configuring a pod in Kubernetes. This recommended configuration enables Kubernetes to select a node with sufficient resources according to the pod resource requirements during the pod deployment. The following example claims that the Nginx pod uses 1-core CPU and 1024 MiB memory, and the pod cannot use more than 2-core CPU or 4096 MiB 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 "
```

Kubernetes uses a static resource scheduling method, which means that instead of using the resources that have been used to calculate the remaining resources on each node, it uses allocated resources. Its calculation method is: `the remaining resources = the total resources - the resources that have been allocated`. If you manually run a resource-consuming program, Kubernetes is not aware of the resources that are being used by the program.

Therefore, you must claim resources for all pods. For the pods that do not have resource claims, after they are scheduled to a node, Kubernetes assumes that the resources used by them on the corresponding node are still available. Therefore, too many pods may be scheduled to this node.

Configure cluster operation and maintenance settings

- Enable Log Service

  When creating a cluster, select the Using Log Service check box.

- Configure cluster monitoring

  Alibaba Cloud Container Service is integrated with CloudMonitor. By configuring monitoring on nodes, you can implement real-time monitoring. By adding monitoring alarm rules, you can quickly locate the issues that cause abnormal resource usage.

  When you create a Kubernetes cluster through Container Service, two application groups are automatically created in CloudMonitor: one for Master nodes and one for Worker nodes. You can add alarm rules under these two groups and these rules apply to all machines in the groups. When subsequent nodes are added to the corresponding group, the alarm rules in the group are automatically applied.

  This means that you only need to configure alarm rules for the ECS resources.

   Note:

  - To monitor ECS instances, you need to set alarm rules for resources such as CPU, memory, and disk. We recommend that you set the `/ var / lib / docker` file on an exclusive disk.

Set an application to wait for its dependent application after it starts

Some applications may have some external dependencies. For example, an application may need to read data from a database (DB) or access the interface of another service. However, when the application starts, the DB or the interface may not be available. In traditional manual O&M, if the external dependencies of an application are unavailable when the application starts, the application exits directly.This is known as `failfast` . This strategy is not applicable for Kubernetes, because O&M in Kubernetes is automated and does not require manual intervention. For example, when you deploy an application, you do not need to manually select a node or start the application on the node. If the application fails, Kubernetes

automatically restarts it. Additionally, automatic capacity increase is supported through HPA when large loads occur.

For example, assume that application A depends on application B, and these two applications run on the same node. After the node restarts, application A starts, but application B has not started. In this case, the dependency of application A is unavailable. According to the strategy of failfast, application A exists and will not start even after application B starts. In this case, application A must be started manually.

In Kubernetes, you can set the system to check the dependency of the application during startup, and to implement polling to wait until the dependency is available. This can be implemented through*Init Container*.

Set the pod restart policy

When a bug in the code or excessive memory consumption causes application processes to fail, the pod in which the processes reside also fails. We recommend that you set a restart policy for the pod so that the pod can automatically restart after failure.

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

Available values of the restart policy parameter are:

- `Always` : indicates to always restart the pod automatically.
- `OnFailure` : indicates to automatically restart the pod when the pod fails (the exiting status of the process is not 0).
- `Never` : indicates to never restart the pod.

Configure the liveness probe and readiness probe

A running pod may not necessarily be able to provide services because processes in the pod may be locked. However, Kubernetes does not automatically restart the pod because the pod is still running. Therefore, you must configure the liveness probe in

each pod to determine whether the pod is alive, and whether it can provide services. Then, Kubernetes restarts the pod when the liveness probe detects any exception.

The readiness probe is used to detect whether the pod is ready to provide services. It takes some time for an application to initialize during startup. During the initialization, the application cannot provide services. The readiness probe can determine when the pod is ready to receive traffic from Ingress or Service. When the pod is faulty, the readiness probe stops new traffic being forwarded to the pod.

```
apiVersion :  v1
kind :  Pod
metadata :
  name :  tomcat
spec :
  containers :
 -  name :  tomcat
    image :  tomcat
    livenessPr  obe :
      httpGet :
        path : / index . jsp
        port :  8080
      initialDel  aySeconds :  3
      periodSeco  nds :  3
    readinessP  robe :
      httpGet :
        path : / index . jsp
        port :  8080
```

Set one process to run in each container

Users who are new to the container technology tend to use containers as virtual machines and put multiple processes into one container, such as monitoring process, log process, sshd process, and even the whole systemd. This causes the following two problems:

- It becomes complex to determine the resource usage of the pod as a whole, and it becomes difficult for the resource limit that you set to take effect.
- If only one process runs in a container, the container engine can detect process failures and it restarts the container upon each process failure. However, if multiple processes are put into a container, the container engine cannot determine the failure of any single process. Therefore, the engine does not restart the container when a single process fails even though the container does not work normally.

If you want to run multiple processes simultaneously, Kubernetes can help you easily implement that. For example, nginx and php-fpm communicate with each other

through a Unix domain socket. You can use a pod that contains two containers, and put the Unix socket into a shared volume of the two containers.

Avoid Single Point of Failure (SPOF)

If an application uses only one ECS instance, the application is unavailable during the period when Kubernetes restarts the instance upon an instance failure. This issue also occurs when you release an updated version of the application. Therefore, we recommend that you do not directly use pods in Kubernetes. Instead, deploy Deployment or StatefulSet applications and set more than two pods for each application.

1.2 Update expired certificates of a Kubernetes cluster

When cluster certificates expire, communication with the cluster API server by using kubectl or calling APIs is disabled, and the expired certificates on cluster nodes cannot be updated automatically through template deployment. To update the certificates, you can log on to each cluster node and run the container stating commands, `docker run`.

Update the expired certificates on a Master node

1. Log on to a Master node with the root permission.
2. Run the following command in any directory to update the expired certificates on the Master node:

```
$ docker run -it --privileged=true -v /:/alicoud-k8s
 -host --pid host --net host \
  registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:
 v1.0.0 /renew/upgrade-k8s.sh --role master
```

3. Repeat the preceding steps on each cluster Master node to update all the expired certificates.

Update the expired certificates on a Worker node

1. Log on to a Master node with the root permission.

2. Run the following command to obtain the cluster rootCA private key:

```
$ cat / etc / kubernetes / pki / ca . key
```

3. Run either of the following commands to obtain the cluster root private key encoded through base64:

   - If the cluster rootCA private key has a blank line, run the following command:

```
$ sed ' 1d ' / etc / kubernetes / pki / ca . key | base64 - w
  0
```

   - If the cluster rootCA private key does not have any blank line, run the following command:

```
$ cat / etc / kubernetes / pki / ca . key | base64 - w 0
```

4. Log on to a Worker node with the root permission.
5. Run the following command in any directory to update the expired certificates on the Worker node.

```
$ docker run - it -- privileged = true - v /:/ alicoud - k8s
 - host -- pid host -- net host \
   registry . cn - hangzhou . aliyuncs . com / acs / cert - rotate
 : v1 . 0 . 0 / renew / upgrade - k8s . sh -- role node --
 rootkey ${ base64CAKe y }
```

Note:

In step 3, you have obtained ${base64CAKey}, which is the cluster root private key encoded through base64.

6. Repeat the preceding steps on each cluster Worker node to update all the expired certificates.

1.3 Update the Kubernetes cluster certificates that are about to expire

This topic describes how to update the Kubernetes cluster certificates that are about to expire. You can use one of three methods to update the cluster certificates. You can update the cluster certificates in the Container Service console, update all the certificates by running a single command, or update Master and Worker node certificates separately by running different commands.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have connected to the Kubernetes cluster through kubectl. For more information, see *Connect to a Kubernetes cluster by using kubectl*.

Updates all certificates through the Container Service console

In the Container Service console, click the Update Certificate prompt of the target cluster. For more information, see *Update the Kubernetes cluster certificates that are about to expire*.

Run a command to update all certificates

Log on to a Master node and run the following command:

```
$ curl http://aliacs-k8s-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/public/cert-update/renew.sh | bash
```

Verify the results

1. Run the following command to view the status of Master nodes and Worker nodes:

```
$ kubectl get nodes
```

```
[root@ ~]# kubectl get nodes
NAME                          STATUS    ROLES     AGE       VERSION
cn-hangzhou.                  Ready     <none>    23d       v1.11.2
cn-hangzhou.                  Ready     <none>    23d       v1.11.2
cn-hangzhou.                  Ready     master    47d       v1.11.2
cn-hangzhou.                  Ready     master    47d       v1.11.2
cn-hangzhou.                  Ready     master    47d       v1.11.2
cn-hangzhou.                  Ready     <none>    47d       v1.11.2
cn-hangzhou.                  Ready     <none>    47d       v1.11.2
[root@ ~]#
```

2. Run the following command. When the value of the `SUCCESSFUL` parameter of each Master node is `1`, and the value of the `SUCCESSFUL` parameter of

each Worker node meets the number of cluster Worker nodes, all certificates are updated.

```
$ kubectl get job -nkube-system
```

```
[root@ ~]# kubectl get job -nkube-system
NAME                          DESIRED   SUCCESSFUL   AGE
aliyun-cert-renew-master-1    1         1            6m
aliyun-cert-renew-master-2    1         1            5m
aliyun-cert-renew-master-3    1         1            5m
aliyun-cert-renew-worker      4         4            4m
cert-job-2                    1         1            22h
cert-job-3                    1         1            22h
cert-job-4                    1         1            22h
cert-node-2                   4         4            19h
```

Manually update the certificates of each Master node

1. Copy the following code and paste it into any path to create a *job-master.yml* file:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: 1
  parallelism: 1
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                - ${hostname}
      containers:
      - command:
        - /renew/upgrade-k8s.sh
        - --role
        - master
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
          name: ${jobname}
      hostNetwork: true
```

```
      hostPID :  true
      restartPol  icy :  Never
      schedulerN  ame :  default - scheduler
      securityCo  ntext : {}
      toleration  s :
    -  effect :  NoSchedule
       key :  node - role . kubernetes . io / master
      volumes :
    -  hostPath :
         path : /
         type :  Directory
```

```
        name : ${ jobname }
```

2. Obtain the number of Master nodes in the cluster and the `hostname` of each Master node.

   - Method 1

     Run the following commands:

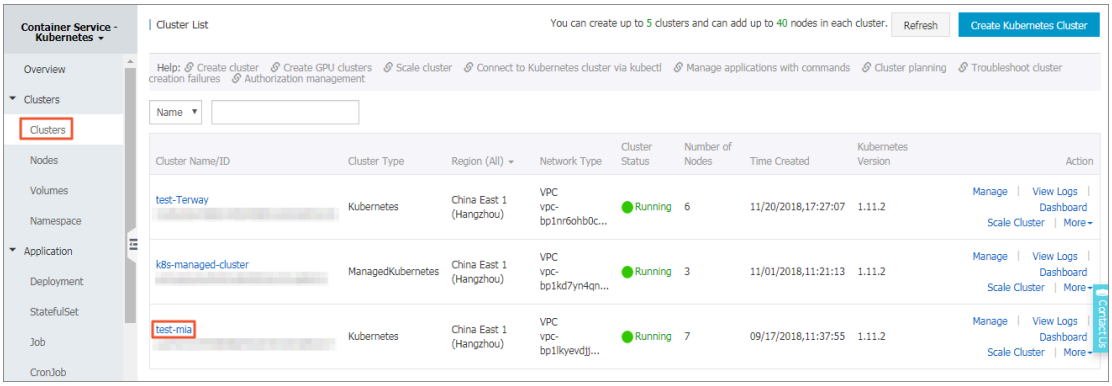
     ```
     $ kubectl get nodes
     ```

     ```
     [root@                    ~]# kubectl get nodes
     NAME                      STATUS   ROLES    AGE   VERSION
     cn-hangzhou.i             Ready    <none>   22d   v1.11.2
     cn-hangzhou.i             Ready    <none>   22d   v1.11.2
     cn-hangzhou.i             Ready    master   46d   v1.11.2
     cn-hangzhou.i             Ready    master   46d   v1.11.2
     cn-hangzhou.i             Ready    master   46d   v1.11.2
     cn-hangzhou.i             Ready    <none>   46d   v1.11.2
     cn-hangzhou.i             Ready    <none>   46d   v1.11.2
     [root@                    ~]#
     ```

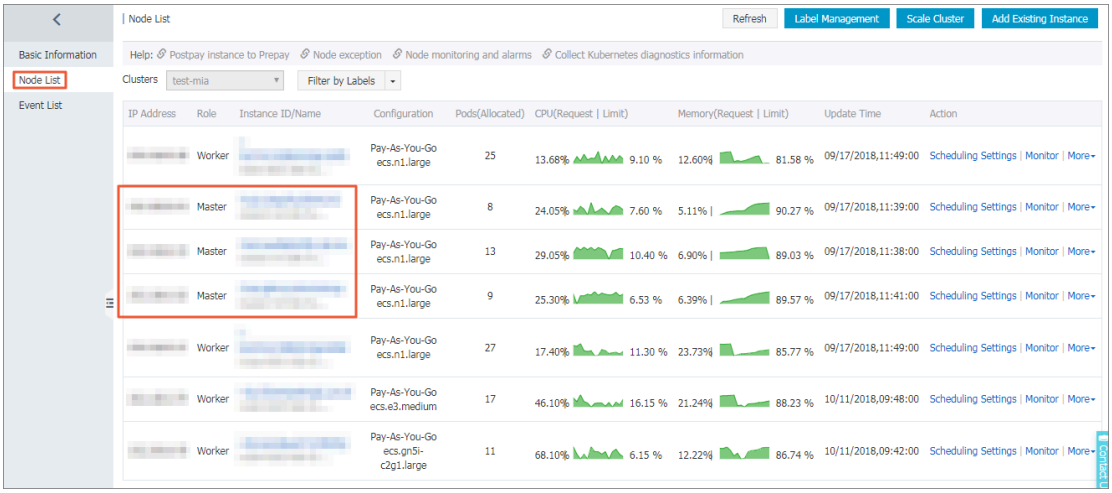
   - Method 2

     a. Log on to the *Container Service console*.

     b. In the left-side navigation pane under Kubernetes, click Clusters.

     c. Click the target cluster name, and then click Node List in the left-side navigation pane to view the number of Master nodes and the `hostname` of each Master node.



3. Run the following command to specify the `${ jobname }` and `${ hostname }` variables in the *job-master.yml* file:

```
$ sed 's/${ jobname }/cert-job-2/g; s/${ hostname }/hostname/g' job-master.yml > job-master2.yml
```

In this code line:

- `${ jobname }` is the Job and pod name. In this example, this variable is set to `cert-job-2`.
- `${ hostname }` is the Master name. In this example, `hostname` is set to a Master name obtained in step 2.

4. Run the following command to create a Job:

```
$ kubectl create -f job-master2.yml
```

5. Run the following command to view the Job status. When the value of the `SUCCESSFUL` parameter is `1`, the certificates of this Master node have been updated.

```
$ kubectl get job -nkube-system
```

6. Repeat step 3 to step 5 to update the certificates of the remaining Master nodes in the cluster.

```
[root@                    ~]# kubectl get job -nkube-system
NAME          DESIRED   SUCCESSFUL   AGE
cert-job-2    1         1            22m
cert-job-3    1         1            2m
cert-job-4    1         1            1m
[root@                    ~]#
```

Manually update Worker node certificates

1. Copy the following code and paste it into any path to create a *job-node.yml* file:

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: ${nodesize}
  parallelism: ${nodesize}
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: job-name
                operator: In
                values:
                - ${jobname}
            topologyKey: kubernetes.io/hostname
      containers:
      - command:
        - /renew/upgrade-k8s.sh
        - --role
        - node
        - --rootkey
        - ${key}
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
          name: ${jobname}
      hostNetwork: true
      hostPID: true
      restartPolicy: Never
      schedulerName: default-scheduler
      securityContext: {}
      volumes:
      - hostPath:
          path: /
          type: Directory
        name: ${jobname}
```

Note:

If a Worker node has a taint, you need to add `tolerations` for the taint in the *job-node.yml* file. More specifcially, you need to add the following code

between `securityContext: {}` and `volumes:` (If the number of Worker nodes that have taints is `n`, you need to add the following code `n` times):

```
      tolerations:
      - effect: NoSchedule
        key: ${key}
        operator: Equal
        value: ${value}
```

The method to obtain `${name}` and `${value}` is as follows:

a. Copy the following code and paste it into any path to create a *taint.tml* file:

```
{{printf "%-50s %-12s\n" "Node" "Taint"}}
{{-range .items}}
    {{-if $taint := (index .spec "taints")}}
        {{-.metadata.name}}{{"\t"}}
        {{-range $taint}}
            {{-.key}}={{.value}}:{{.effect}}{{"\t"}}
        {{-end}}
        {{-"\n"}}
    {{-end}}
{{-end}}
```

b. Run the following command to view the values of `${name}` and `${value}` for the Worker nodes that have taints:

```
$ kubectl get nodes -o go-template-file="taint.tml"
```

```
[root@                    ~]# kubectl get nodes -o go-template-file="taint.tml"
Node                                          Taint
cn-hangzhou.i-            key1=value1:NoSchedule
cn-hangzhou.i-            node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-            node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-            node-role.kubernetes.io/master=<no value>:NoSchedule
```

2. Run the following command to obtain the cluster CAKey:

```
$ sed ' 1d ' / etc / kubernetes / pki / ca . key | base64 - w
 0
```

3. Run the following command to specify the `${ jobname }`, `${ nodesize }`, and `${ key }` variables in the *job - node . yml* file:

```
$ sed ' s /${ jobname }/ cert - node - 2 / g ; s /${ nodesize }/
 nodesize / g ; s /${ key }/ key / g ' job - node . yml > job -
 node2 . yml
```

In this code line:

- `${ jobname }` is the Job and pod name. In this example, this variable is set to `cert - node - 2 .`
- `${ nodesize }` is the number of Worker nodes. For how to obtain this value, see step 2 in *Manually update the certificates of each Master node*. In this example, the `nodesize` variable is replaced with the number of the Worker nodes in the cluster.
- `${ key }` is the cluster CAKey. In this example, the `key` variable is replaced with the CAKey obtained in step 3 of *Manually update Worker node certificates*.

4. Run the following command to create a Job:

```
$ kubectl create - f job - node2 . yml
```

5. Run the following command to view the Job status. When the value of the `SUCCESSFUL` parameter is equal to the number of the cluster Worker nodes, all certificates have been updated.

```
$ kubectl get job - nkube - system
```

```
[root@ ~]# kubectl get job -nkube-system
NAME          DESIRED   SUCCESSFUL   AGE
cert-job-2    1         1            1h
cert-job-3    1         1            47m
cert-job-4    1         1            46m
cert-node-2   4         4            1m
[root@ ~]#
```

1.4 Plan Kubernetes CIDR blocks under a VPC

Generally, you can select to create a Virtual Private Cloud (VPC) automatically and use the default network address when creating a Kubernetes cluster in Alibaba

Cloud. In some complicated scenarios, plan the Elastic Compute Service (ECS) address, Kubernetes pod address, and Kubernetes service address on your own. This document introduces what the addresses in Kubernetes under Alibaba Cloud VPC environment are used for and how to plan the CIDR blocks.

Basic concepts of Kubernetes CIDR block

The concepts related to IP address are as follows:

VPC CIDR block

The CIDR block selected when you create a VPC. Select the VPC CIDR block from 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16.

VSwitch CIDR Block

The CIDR block specified when you create a VSwitch in VPC. The VSwitch CIDR block must be the subset of the current VPC CIDR block, which can be the same as the VPC CIDR block but cannot go beyond that range. The address assigned to the ECS instance under the VSwitch is obtained from the VSwitch CIDR block. Multiple VSwitches can be created under one VPC, but the VSwitch CIDR blocks cannot overlap .

The VPC CIDR block structure is as follows.

Pod CIDR block

Pod is a concept in Kubernetes. Each pod has one IP address. You can specify the pod CIDR block when creating a Kubernetes cluster in Alibaba Cloud Container Service and the pod CIDR block cannot overlap with the For example, if the VPC CIDR block is 172.16.0.0/12, then the pod CIDR block of Kubernetes cannot use 172.16.0.0/16, 172.17.0.0/16, or any address that is included in 172.16.0.0/12.

Service CIDR block

Service is a concept in Kubernetes. Each service has its own address. The service CIDR block cannot overlap with the VPC CIDR block or pod CIDR block. The service address is only used in a Kubernetes cluster and cannot be used outside a Kubernetes cluster.

The relationship between Kubernetes CIDR block and VPC CIDR block is as follows.

How to select CIDR block

Scenario of one VPC and one Kubernetes cluster

This is the simplest scenario. The VPC address is determined when the VPC is created. Select a CIDR block different from that of the current VPC when creating a Kubernetes cluster.

Scenario of one VPC and multiple Kubernetes clusters

Create multiple Kubernetes clusters under one VPC. In the default network mode (Flannel), the pod message needs to be routed by using VPC, and Container Service automatically configures the route table to each pod CIDR block on the VPC route. The pod CIDR blocks of all the Kubernetes clusters cannot overlap, but the service CIDR blocks can overlap.

The VPC address is determined when the VPC is created. Select a CIDR block that does not overlap with the VPC address or other pod CIDR blocks for each Kubernetes cluster when creating a Kubernetes cluster.

In such a situation, parts of the Kubernetes clusters are interconnected. The pod of one Kubernetes cluster can directly access the pod and ECS instance of another Kubernetes cluster, but cannot access the

Scenario of VPC interconnection

You can configure what messages are to be sent to the opposite VPC by using route tables when two VPCs are interconnected. Take the following scenario as an example: VPC 1 uses the CIDR block 192.168.0.0/16 and VPC 2 uses the CIDR block 172.16.0.0/12. By using route tables, specify to send the messages of 172.16.0.0/12 in VPC 1 to VPC 2.

In such a situation, the CIDR block of the Kubernetes cluster created in VPC 1 cannot overlap with VPC 1 CIDR block or the CIDR block to be routed to VPC 2. applies to the scenario when you create a Kubernetes cluster in VPC 2. In this example, the pod CIDR block of the Kubernetes cluster can select a sub-segment under 10.0.0.0/8.

Note:
The CIDR block routing to VPC 2 can be considered as an occupied address. Kubernetes clusters cannot overlap with an occupied address.

To access the Kubernetes pod of VPC 1 in VPC 2, configure the route to the Kubernetes cluster in VPC 2.

Scenario of VPC to IDC

Similar to the scenario of VPC interconnection, if parts of the CIDR blocks in VPC route to IDC, the pod address of Kubernetes clusters cannot overlap with those addresses. pod address of Kubernetes clusters in IDC, configure the route table to leased line virtual border router (VBR) in IDC.

2 Network

2.1 Deploy a highly reliable Ingress controller

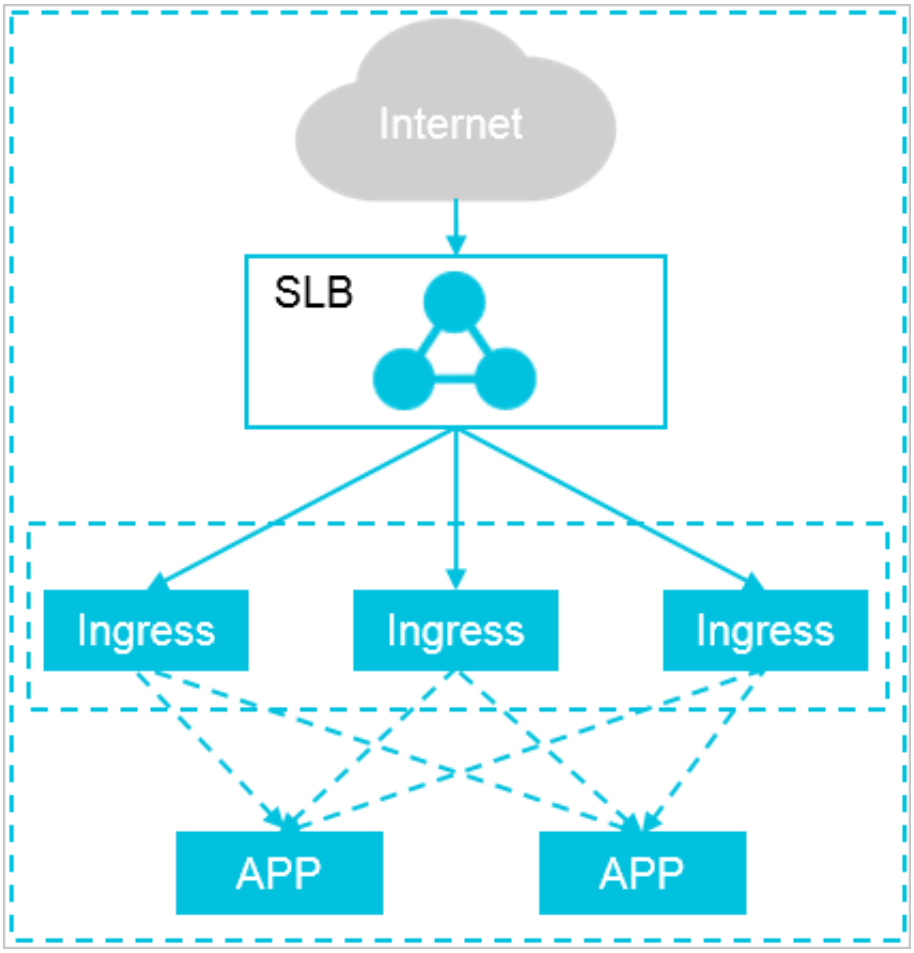
Ingress is a set of rules that authorize external access to the services in a Kubernetes cluster, providing Layer-7 Server Load Balancer capabilities. You can configure Ingress to provide externally accessible URLs, SLB, SSL, and name-based virtual hosts. Ingress requires high reliability because Ingress functions as the access layer through which external traffic goes into a cluster. This topic describes how to deploy a high-performance, highly reliable Ingress access layer.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have connected to the Master node by using SSH. For more information, see *Access Kubernetes clusters by using SSH*.

Highly reliable deployment architecture

To achieve high reliability, you must first resolve any SPOFs. Deploying multiple replicas is the general solution for this problem. Specifically, you can use the multi-node deployment architecture to deploy a highly reliable Ingress access layer in a Kubernetes cluster. We also recommend that you configure exclusive Ingress nodes to prevent service applications from competing for resources with the Ingress service because Ingress functions as the traffic access port of a cluster.



As shown in the preceding figure, multiple exclusive Ingress instances constitute an access layer that processes the inbound traffic to the cluster. Furthermore, the number of Ingress nodes can be scaled according to the traffic amount required by the backend services. If your cluster is of a moderate size, you can also deploy the Ingress service and other service applications in a hybrid way. However, we recommend that you limit the number of resources and isolate them for the Ingress and corresponding applications.

View the cluster pod replicas deployed by default and the Internet SLB address

After you create a cluster, a set of Nginx Ingress controller services that have two pod replicas are deployed within the cluster by default. The frontend of this set of services is mounted to an Internet SLB instance.

Run the following command to view the pods on which the Nginx Ingress controller services are deployed:

```
$ kubectl -n kube-system get pod | grep nginx-ingress-controller
nginx-ingress-controller-8648ddc696-2bshk   1/1     Running   0          3h
```

```
nginx - ingress - controller - 8648ddc696 - jvbs9
  1 / 1       Running     0            3h
```

Run the following command to view the Internet SLB address corresponding to the nginx-ingress-lb service:

```
$  kubectl  - n   kube - system   get   svc   nginx - ingress - lb
 NAME                TYPE            CLUSTER - IP      EXTERNAL - IP
       PORT ( S )                      AGE
 nginx - ingress - lb     LoadBalanc  er     172 . xx . x . xx     118
 . xxx . xxx . xx     80 : 32457 / TCP , 443 : 31370 / TCP     21d
```

To guarantee the high performance and availability of the cluster access layer for a growing cluster, you need to expand the Ingress access layer. You can use either of the following two methods:

Method 1: Expand the number of replicas

You can quickly scale the Ingress access layer by changing the number of the replicas of the Nginx Ingress controller deployment.

Run the following command to scale out the number of pod replicas to three:

```
$  kubectl  - n   kube - system   scale  -- replicas = 3   deployment
 / nginx - ingress - controller
 deployment . extensions / nginx - ingress - controller   scaled
```

Run the following command to view the pods on which the Nginx Ingress controller services are deployed:

```
$  kubectl  - n   kube - system   get   pod  |  grep   nginx - ingress
 - controller
 nginx - ingress - controller - 8648ddc696 - 2bshk
   1 / 1       Running     0            3h
 nginx - ingress - controller - 8648ddc696 - jvbs9
   1 / 1       Running     0            3h
 nginx - ingress - controller - 8648ddc696 - xqmfn
   1 / 1       Running     0            33s
```

Method 2: Deploy the Ingress service on a specified node

If you want the Nginx Ingress controller to run on target nodes of advanced configurations only, you can label the target nodes.

1. Run the following command to view the cluster nodes:

```
$  kubectl   get   node
 NAME                                  STATUS     ROLES      AGE
   VERSION
 cn - hangzhou . i - bp11bcmsna  8d4bpf17bc     Ready      master
   21d     v1 . 11 . 5
 cn - hangzhou . i - bp12h6biv9  bg24lmdc2o     Ready     < none >
   21d     v1 . 11 . 5
```

```
cn - hangzhou . i - bp12h6biv9  bg24lmdc2p     Ready     < none >
  21d     v1 . 11 . 5
cn - hangzhou . i - bp12h6biv9  bg24lmdc2q     Ready     < none >
  21d     v1 . 11 . 5
cn - hangzhou . i - bp181pofzy  yksie2ow03     Ready      master
  21d     v1 . 11 . 5
cn - hangzhou . i - bp1cbsg6rf  3580z6uyo7     Ready      master
  21d     v1 . 11 . 5
```

2. Run the following commands to add the label `node - role . kubernetes . io / ingress =" true "` to the Ingress node `cn - hangzhou . i - bp12h6biv9 bg24lmdc2o` and the Ingress node `cn - hangzhou . i - bp12h6biv9 bg24lmdc2p` :

> **Note:**
>
> - The number of the labeled nodes must be greater than or equal to the number of the cluster pod replicas so that multiple pods do not run on one node.
> - We recommend that you label Worker nodes only to deploy the Ingress service.

```
$  kubectl   label   nodes   cn - hangzhou . i - bp12h6biv9
 bg24lmdc2o   node - role . kubernetes . io / ingress =" true "
 node / cn - hangzhou . i - bp12h6biv9  bg24lmdc2o   labeled
```

```
$  kubectl   label   nodes   cn - hangzhou . i - bp12h6biv9
 bg24lmdc2p   node - role . kubernetes . io / ingress =" true "
 node / cn - hangzhou . i - bp12h6biv9  bg24lmdc2p   labeled
```

3. Run the following command to update your deployment and add the nodeSelector setting:

```
$  kubectl  - n   kube - system   patch   deployment   nginx -
 ingress - controller  - p  '{" spec ": {" template ": {" spec ": {"
 nodeSelect  or ": {" node - role . kubernetes . io / ingress ": "
 true "}}}}}'
 deployment . extensions / nginx - ingress - controller   patched
```

Result:

Run the following command to verify that the Ingress pods are deployed on the cluster nodes that are labeled by `node - role . kubernetes . io / ingress =" true "`:

```
$  kubectl  - n   kube - system   get   pod  - o   wide  |  grep
 nginx - ingress - controller
 nginx - ingress - controller - 8648ddc696 - 2bshk
   1 / 1       Running     0            3h      172 . 16 . 2 . 15
   cn - hangzhou . i - bp12h6biv9  bg24lmdc2p    < none >
 nginx - ingress - controller - 8648ddc696 - jvbs9
   1 / 1       Running     0            3h      172 . 16 . 2 . 145
   cn - hangzhou . i - bp12h6biv9  bg24lmdc2o    < none >
```

3 Storage

3.1 Use a static cloud disk when creating a stateful service

This topic describes typical scenarios in which a static cloud disk is needed for creating a stateful service, and the procedure for how to use one.

Scenarios and method

Scenarios for using cloud disks:

- You want to create applications that demand high disk I/O performance and do not require shared data. For example, MySQL, Redis, and other data storage services.
- You want logs to be written at high speed.
- You want your stored data to exist persistently. That is, the data still exist when the life cycle of the pod ends.

Scenario for using static cloud disks:

You have purchased a cloud disk.

Method of using static cloud disks:

Manually create a Persistent Volume (PV) and a Persistent Volume Claim (PVC).

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have created a cloud disk. For more information, see *Create a cloud disk*.
- You have connected to the Kubernetes cluster by using kubectl, see *Connect to a Kubernetes cluster by using kubectl*.

Limits

- Cloud disks are the non-shared storage devices provided by the Alibaba Cloud Storage Team. Each cloud disk can be mounted to only one pod.
- In a Kubernetes cluster, a cloud disk can be mounted only to a node that resides in the same zone as the cloud disk.

Create a PV

1. Create a *pv-static.yaml* file.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <your-disk-id>
  labels:
    alicloud-pvname: <your-disk-id>
    failure-domain.beta.kubernetes.io/zone: <your-zone>
    failure-domain.beta.kubernetes.io/region: <your-region>
spec:
  capacity:
    storage: 20Gi
  accessModes:
  - ReadWriteOnce
  flexVolume:
    driver: "alicloud/disk"
    fsType: "ext4"
    options:
      volumeId: "<your-disk-id>"
```

Note:

- `alicloud-pvname: <your-disk-id>`: indicates the PV name. This parameter must be set to the same value as that of the `volumeID` parameter, namely, the cloud disk ID.
- `failure-domain.beta.kubernetes.io/zone: <your-zone>`: indicates the zone in which the cloud disk resides. For example, `cn-hangzhou-b`.
- `failure-domain.beta.kubernetes.io/region: <your-region>`: indicates the region in which the cloud disk resides. For example, `cn-hangzhou`.

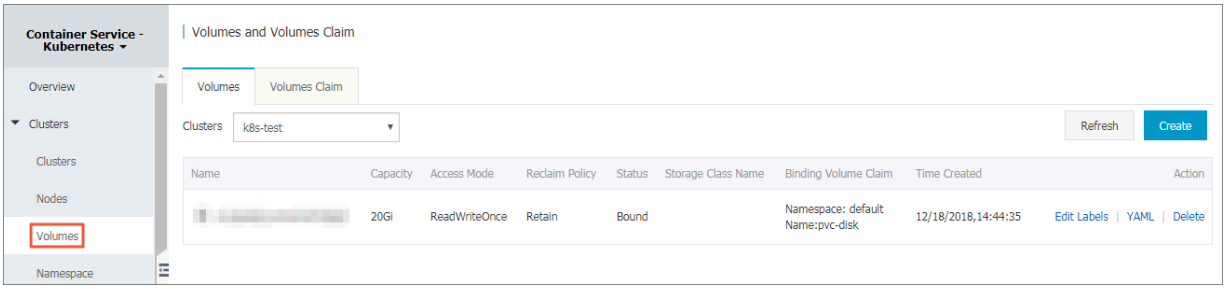
If you use a Kubernetes cluster that has multiple zones, you must set the `failure-domain.beta.kubernetes.io/zone` parameter and the `failure-domain.beta.kubernetes.io/region` parameter so that you can guarantee that your pod can be scheduled to the zone in which the cloud disk resides.

2. Run the following command to create a PV:

```
$ kubectl create -f pv-static.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Clusters > Volumes, and select the target cluster to see the created PV.



Create a PVC

Create a PVC for the cloud disk. Specifically, you need to set the `selector` field to filter for the created PV so that you can associate the PVC with the correct PV.

1. Create a *pvc-static.yaml* file.

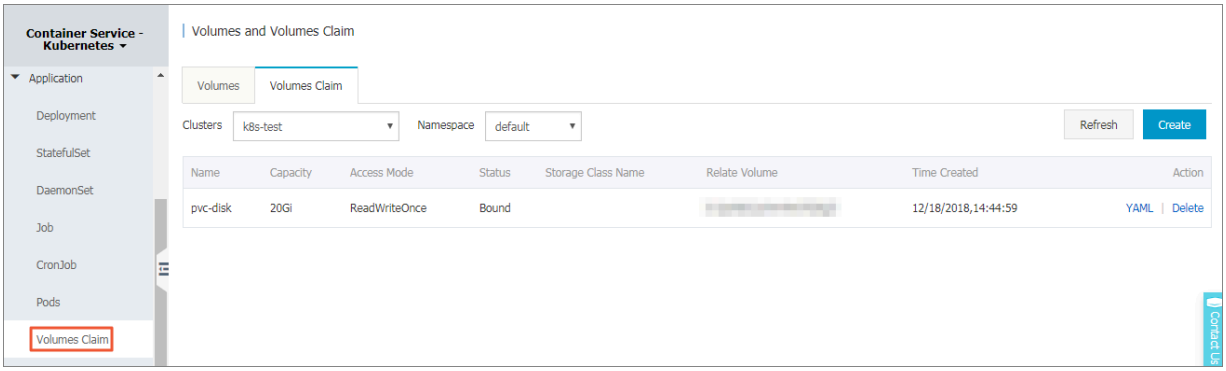
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-disk
spec:
  accessModes:
   - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  selector:
    matchLabels:
      alicloud-pvname: <your-disk-id>
```

2. Run the following command to create a PVC:

```
$ kubectl create -f pvc-static.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Volumes Claim, and select the target cluster and namespace to see the created PVC.



Create an application

1. Create a *static.yaml* file.

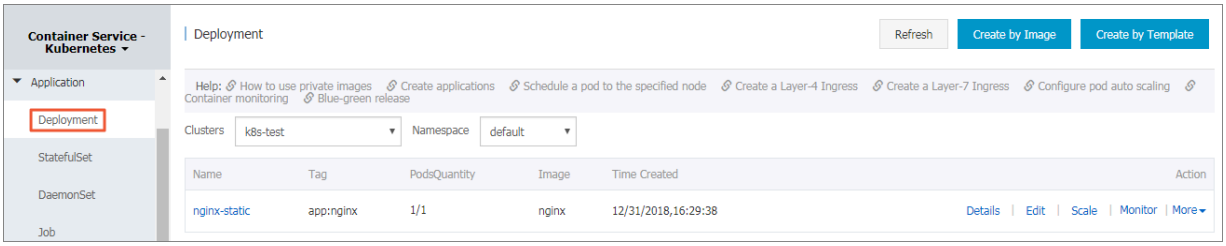
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-static
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - name: disk-pvc
            mountPath: "/data"
      volumes:
        - name: disk-pvc
          persistentVolumeClaim:
            claimName: pvc-disk
```

2. Run the following command to create a deployment:

```
$ kubectl create -f static.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Deployment, and select the target cluster and namespace to see the created deployment.



Persistent data storage on the static cloud disk

1. Run the following command to view the pod in which the created deployment resides:

```
$ kubectl get pod | grep static
```

```
nginx-static-78c7dcb9d7-g9lll     2/2       Running       0
          32s
```

2. Run the following command to check whether the new cloud disk is mounted to the `/data` path:

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll df |
grep data
/dev/vdf          20511312      45080    20449848     1%/data
```

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

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /
data
lost+found
```

4. Run the following command to create a file named `static` in the `/data` path:

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll touch /
data/static
```

5. Run the following command to view the files in the `/data` path:

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /
data
static
lost+found
```

6. Run the following command to remove the pod named `nginx-static-78c7dcb9d7-g9lll`:

```
$ kubectl delete pod nginx-static-78c7dcb9d7-g9lll
pod "nginx-static-78c7dcb9d7-g9lll" deleted
```

7. Open another kubectl interface and run the following command to view the process in which the preceding pod is removed and a new pod is created by Kubernetes:

```
$ kubectl get pod -w -l app=nginx
NAME                              READY     STATUS      RESTARTS
  AGE
nginx-static-78c7dcb9d7-g9lll     2/2       Running     0
         50s
nginx-static-78c7dcb9d7-g9lll     2/2     Terminating   g
0       72s
nginx-static-78c7dcb9d7-h6brd     0/2     Pending     0
  0s
nginx-static-78c7dcb9d7-h6brd     0/2     Pending     0
  0s
nginx-static-78c7dcb9d7-h6brd     0/2     Init:0/1
0       0s
nginx-static-78c7dcb9d7-g9lll     0/2     Terminating   g
0       73s
nginx-static-78c7dcb9d7-h6brd     0/2     Init:0/1
0       5s
```

```
nginx-static-78c7dcb9d7-g9lll     0/2     Terminating   0          78s
nginx-static-78c7dcb9d7-g9lll     0/2     Terminating   0          78s
nginx-static-78c7dcb9d7-h6brd     0/2     PodInitializing   0          6s
nginx-static-78c7dcb9d7-h6brd     2/2     Running   0          8s
```

8. Run the following command to view the new pod created by Kubernetes:

```
$ kubectl get pod
NAME                                READY     STATUS        RESTARTS   AGE
nginx-static-78c7dcb9d7-h6brd     2/2       Running       0          14s
```

9. Run the following command to verify that the created file named `static` in the */data* path has not been removed, indicating that data in the static cloud disk can be stored persistently:

```
$ kubectl exec nginx-static-78c7dcb9d7-h6brd ls /
data
static
lost+found
```

3.2 Use a dynamic cloud disk when creating a stateful service

This topic describes typical scenarios in which a dynamic cloud disk is needed for creating a stateful service, and the procedure for how to use one.

Scenarios and method

Scenario for using dynamic cloud disks:

You want to configure the system to automatically purchase cloud disks when you deploy an application, rather than manually purchase cloud disks before deploying the application.

Method of using a dynamic cloud disk:

1. Manually create a PVC and claim a specific StorageClass in the PVC.
2. Use the StorageClass to enable the system to automatically create a PV when you deploy an application.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.

- You have connected to the Kubernetes cluster by using kubectl, see *Connect to a Kubernetes cluster by using kubectl*.
- You have installed the provisioner plugin in the Kubernetes cluster. The plugin automatically creates a cloud disk according to a specific StorageClass.

Provisioner plugin

When you create a cluster through Alibaba Cloud Container Service for Kubernetes, the provisioner plugin is installed in the cluster by default.

Create a StorageClass

By default, Alibaba Cloud Container Service for Kubernetes creates four StorageClasses for a cluster during the cluster initialization, and the StorageClasses use the default settings. Furthermore, the four default StorageClasses are created only for a cluster that has a single zone. For a cluster that has multiple zones, you need to manually create a StorageClass. The following are the four StorageClasses created by default:

- *alicloud-disk-common* indicates to automatically create a basic cloud disk.
- *alicloud-disk-efficiency* indicates to automatically create an Ultra cloud disk.
- *alicloud-disk-ssd* indicates to automatically create an SSD cloud disk.
- *alicloud-disk-available* indicates a systematic method of disk selection. Specifically, the system first attempts to create an Ultra cloud disk. If the Ultra cloud disks in the specified zone are sold out, the system tries to create an SSD cloud disk. If the SSD cloud disks are sold out, the system tries to create a basic cloud disk.

1. Create a *storageclass.yaml* file.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-ssd-hangzhou-b
provisioner: alicloud/disk
reclaimPolicy: Retain
parameters:
  type: cloud_ssd
  regionid: cn-hangzhou
  zoneid: cn-hangzhou-b
  fstype: "ext4"
```

```
    readonly : " false "
```

Parameter setting

- `provisione r` : Set this parameter to alicloud/disk to specify that the StorageClass creates an Alibaba Cloud cloud disk by using the provisioner plugin.
- `reclaimPol icy` : Set a policy to reclaim the cloud disk. Available values of this parameter are `Delete` and `Retain` . The default setting is `Delete` .

   Note:
  If you maintain the default setting, namely, `Delete` , the data on the cloud disk cannot be restored after you remove the PVC because the cloud disk is also removed.
- `type` : Specify a cloud disk type by using one the following values: `cloud` , `cloud_effi ciency` , `cloud_ssd` , and `available` .
- `regionid` : (optional) Set the region in which the cloud disk is automatically created. This region must be the same as the region in which your cluster resides.
- `zoneid` : (optional) Set the zone in which a cloud disk is automatically created.
  - If you set this parameter for a single-zone cluster, the value must be the same as the zone in which the cluster resides.
  - If you set this parameter for a multi-zone cluster, multiple values can be set. For example,

    ```
    zoneid :  cn - hangzhou - a , cn - hangzhou - b , cn -
    hangzhou - c
    ```
- `fstype` : (optional) Set the type of the file system used for automatic cloud disk creation. The default setting is `ext4` .
- `readonly` : (optional) Set whether the automatically created cloud disk is read only. If you set this parameter to `true` , the cloud disk can only be read. If you set this parameter to `false` , the cloud disk can be read and written. The default setting is `false` .
- `encrypted` : (optional) Set whether to encrypt the automatically created cloud disk. If you set this parameter to `true` , the cloud disk is encrypted. If you set

this parameter to `false`, the cloud disk is not encrypted. The default setting is `false`.

2. Run the following command to create a StorageClass:

```
$ kubectl create -f storagecla ss.yaml
```

Create a PVC

1. Create a *pvc-ssd.yaml* file.

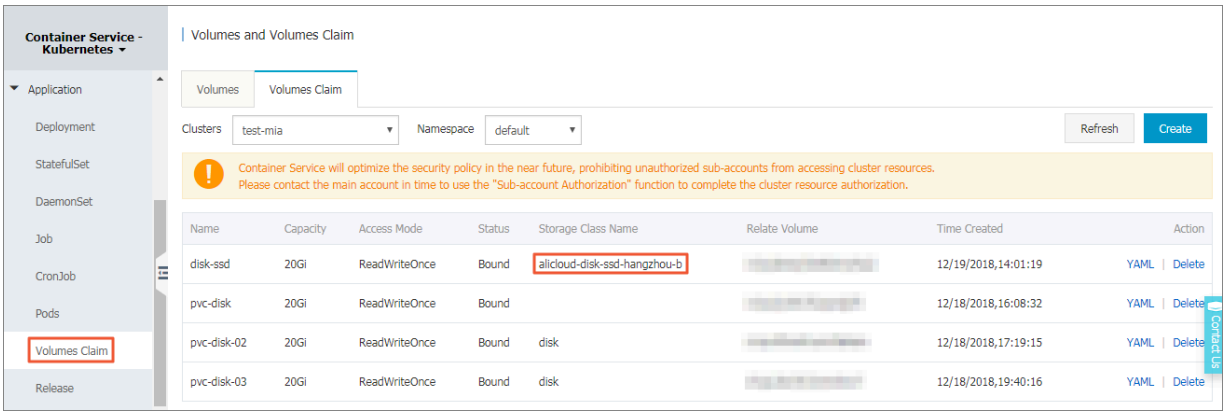
```
kind: PersistentVolumeClai m
apiVersion: v1
metadata:
  name: disk-ssd
spec:
  accessMode s:
   - ReadWriteO nce
  storageCla ssName: alicloud-disk-ssd-hangzhou-b
  resources:
    requests:
      storage: 20Gi
```

2. Run the following command to create a PVC:

```
$ kubectl create -f pvc-ssd.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Volumes Claim, and select the target cluster and namespace to see that the storage class name associated to the PVC is `alicloud-disk-ssd-hangzhou-b` specified in the `StorageCla ss`, and the PVC is associated with the volume.



Create an application

1. Create a *pvc-dynamic.yaml* file.

```
apiVersion: apps/v1
kind: Deployment
```

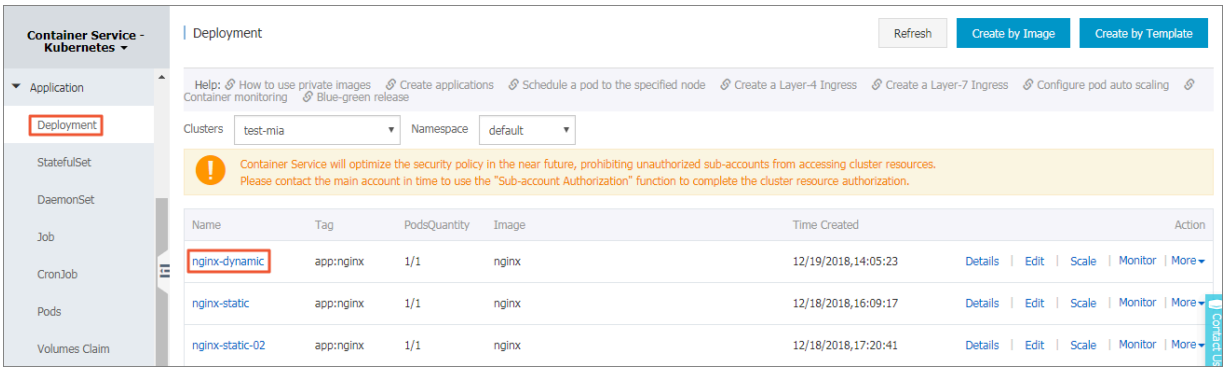
```
metadata :
  name :  nginx - dynamic
  labels :
    app :  nginx
spec :
  selector :
    matchLabel  s :
      app :  nginx
  template :
    metadata :
      labels :
        app :  nginx
    spec :
      containers :
      -  name :  nginx
         image :  nginx
         volumeMoun  ts :
          -  name :  disk - pvc
             mountPath : "/ data "
      volumes :
       -  name :  disk - pvc
          persistent  VolumeClai  m :
            claimName :  disk - ssd
```

2. Run the following command to create a deployment:

```
$  kubectl  create  - f  nginx - dynamic . yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Deployment, and select the target cluster and namespace to see the created deployment.



Persistent storage for a dynamic cloud disk

1. Run the following command to view the pod in which the created deployment resides:

```
$  kubectl  get  pod  |  grep  dynamic
```

```
nginx-dynamic-5c74594ccb-zl9pf       2/2       Running
  0          3m
```

2. Run the following command to check whether a new cloud disk is mounted to the */data* path:

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf df |
grep data
/dev/vdh          20511312     45080    20449848     1% /data
```

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

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /
data
lost+found
```

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

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf touch /
data/dynamic
```

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

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /
data
dynamic
lost+found
```

6. Run the following command to remove the pod named `nginx-dynamic-78c7dcb9d7-g9lll`:

```
$ kubectl delete pod nginx-dynamic-5c74594ccb-zl9pf
pod "nginx-dynamic-5c74594ccb-zl9pf" deleted
```

7. Open another kubectl interface and run the following command to view the process in which the preceding pod is removed and a new pod is created by Kubernetes:

```
$ kubectl get pod -w -l app=nginx
NAME                                READY     STATUS      RESTARTS
    AGE
nginx-dynamic-5c74594ccb-zl9pf       2/2       Running
0          6m48s
nginx-dynamic-5c74594ccb-zl9pf     2/2     Terminating
  0       7m32s
nginx-dynamic-5c74594ccb-45sd4     0/2     Pending     0
   0s
nginx-dynamic-5c74594ccb-45sd4     0/2     Pending     0
   0s
nginx-dynamic-5c74594ccb-45sd4     0/2     Init:0/1
0       0s
nginx-dynamic-5c74594ccb-zl9pf     0/2     Terminating
  0       7m32s
nginx-dynamic-5c74594ccb-zl9pf     0/2     Terminating
  0       7m33s
```

```
nginx - dynamic - 5c74594ccb - zl9pf     0 / 2     Terminatin  g
  0       7m33s
nginx - dynamic - 5c74594ccb - 45sd4     0 / 2     PodInitial
izing     0       5s
nginx - dynamic - 5c74594ccb - 45sd4     2 / 2     Running     0
   22s
```

8. Run the following command to view the pod newly created by Kubernetes:

```
$ kubectl get pod
NAME                                  READY     STATUS
RESTARTS     AGE
nginx - dynamic - 5c74594ccb - 45sd4       2 / 2       Running
  0            2m
```

9. Run the following command to verify that the created file named `dynamic` in the */ data* path has not been removed, indicating that data in the dynamic cloud disk can be stored persistently:

```
$ kubectl exec nginx - dynamic - 5c74594ccb - 45sd4 ls /
data
dynamic
lost + found
```

3.3 Use a StatefulSet service

This topic describes the typical scenarios in which a StatefulSet is needed for creating a stateful service, and the procedure for how to use one.

Background information

A StatefulSet with N replicas is typically used for applications that require one or more of the following conditions:

- A stable deployment order. Pods are deployed or expanded sequentially. That is, pods are deployed in the defined order of 0 to N-1. Before a new pod is deployed, all its predecessors must have been in Running and Ready status.
- A stable scaling order. Pods are deleted in the defined order of N-1 to 0. Before a pod is deleted, all its predecessors must be all Running and Ready.
- Stable and unique network identifiers. After a pod is rescheduled to any other node , its PodName and HostName remain unchanged.
- Stable and persistent storage implemented through a PVC. After a pod is rescheduled, it can still access the same persistent data.

Method of using a StatefulSet service

Set `volumeClai mTemplates` to enable the system to automatically create a PVC and a PV.

This topic describes how to:

- Deploy a StatefulSet service
- Scale a StatefulSet service
- Remove a StatefulSet service
- Persistent storage of a StatefulSet service

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have connected to the Master node of the Kubernetes cluster. For more information, see *Connect to a Kubernetes cluster by using kubectl*.

Deploy a StatefulSet service

Note:

`volumeClai mTemplates` : indicates a template of PVCs of the same type. If you set this field, the system creates PVCs according to the number of the replicas that are set for the StatefulSet service. That is, the number of the PVCs and that of the replicas are the same. Furthermore, these PVCs share the same settings except for names.

1. Create a *statefulse t . yaml* file.

Note:

You need to set the `storageCla ssName` parameter to `alicloud - disk - ssd` , indicating that an Alibaba Cloud SSD cloud disk is used.

```
apiVersion :  v1
kind :  Service
metadata :
  name :  nginx
  labels :
    app :  nginx
spec :
  ports :
  -  port :  80
     name :  web
  clusterIP :  None
  selector :
    app :  nginx
```

```
---
 apiVersion :  apps / v1beta2
 kind :  StatefulSe  t
 metadata :
   name :  web
 spec :
   selector :
     matchLabel  s :
       app :  nginx
   serviceNam  e : " nginx "
   replicas :  2
   template :
     metadata :
       labels :
         app :  nginx
     spec :
       containers :
      -  name :  nginx
         image :  nginx
         ports :
        -  containerP  ort :  80
           name :  web
         volumeMoun  ts :
        -  name :  disk - ssd
           mountPath : / data
   volumeClai  mTemplates :
  -  metadata :
       name :  disk - ssd
     spec :
       accessMode  s : [ " ReadWriteO  nce " ]
       storageCla  ssName : " alicloud - disk - ssd "
       resources :
         requests :
           storage :  20Gi
```

2. Run the following command to deploy a StatefulSet service:

```
$  kubectl   create  - f   statefulse  t . yaml
```

3. Open another kubectl interface and run the following command to check that the pods are deployed in order:

```
$  kubectl   get   pod  - w  - l   app = nginx
 NAME           READY     STATUS      RESTARTS     AGE
 web - 0          0 / 1       Pending     0            0s
 web - 0          0 / 1       Pending     0            0s
 web - 0          0 / 1       ContainerC  reating      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       ContainerC  reating      0
 0s
 web - 1          1 / 1       Running     0            7s
```

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

```
$  kubectl   get   pod
 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
NAME              STATUS     VOLUME                      CAPACITY
  ACCESS   MODES     STORAGECLA SS          AGE
disk - ssd - web - 0     Bound      d - 2zegw7et6x  c96nbojuoo
20Gi         RWO             alicloud - disk - ssd     7m
disk - ssd - web - 1     Bound      d - 2zefbrqggv  kd10xb523h
20Gi         RWO             alicloud - disk - ssd     6m
```

Scale a StatefulSet service

Scale out a StatefulSet service

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

```
$ kubectl scale sts web -- replicas = 3
statefulse t . apps / web scaled
```

2. Run the following command to view the pods:

```
$ kubectl get pod
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:

```
$ kubectl get pvc
NAME              STATUS     VOLUME                      CAPACITY
  ACCESS   MODES     STORAGECLA SS          AGE
disk - ssd - web - 0     Bound      d - 2zegw7et6x  c96nbojuoo
20Gi         RWO             alicloud - disk - ssd     35m
disk - ssd - web - 1     Bound      d - 2zefbrqggv  kd10xb523h
20Gi         RWO             alicloud - disk - ssd     34m
disk - ssd - web - 2     Bound      d - 2ze4jx1zym  n4n9j3pic2
20Gi         RWO             alicloud - disk - ssd     27m
```

Scale in a StatefulSet service

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

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

```
statefulse  t . apps / web   scaled
```

2. Run the following command to view the pod and verify that the number of pods is reduced to two:

```
$  kubectl   get   pod
NAME                          READY     STATUS      RESTARTS
AGE
web - 0                         1 / 1       Running     0
   38m
web - 1                         1 / 1       Running     0
   38m
```

3. Run the following command to view the PVCs and verify that the number of PVCs and PVs remains unchanged after the number of pods is changed:

```
$  kubectl   get   pvc
NAME               STATUS     VOLUME                      CAPACITY
  ACCESS   MODES     STORAGECLA  SS          AGE
disk - ssd - web - 0     Bound      d - 2zegw7et6x  c96nbojuoo
20Gi         RWO             alicloud - disk - ssd     39m
disk - ssd - web - 1     Bound      d - 2zefbrqggv  kd10xb523h
20Gi         RWO             alicloud - disk - ssd     39m
disk - ssd - web - 2     Bound      d - 2ze4jx1zym  n4n9j3pic2
20Gi         RWO             alicloud - disk - ssd     31m
```

Rescale out a StatefulSet service

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

```
$  kubectl   scale   sts   web  -- replicas = 3
statefulse  t . apps / web   scaled
```

2. Run the following command to view the pods:

```
$  kubectl   get   pod
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 and verify that the newly created pods still use the original PVCs and PVs after the StatefulSet service is scaled out:

```
$  kubectl   get   pvc
NAME               STATUS     VOLUME                      CAPACITY
  ACCESS   MODES     STORAGECLA  SS          AGE
disk - ssd - web - 0     Bound      d - 2zegw7et6x  c96nbojuoo
20Gi         RWO             alicloud - disk - ssd     1h
disk - ssd - web - 1     Bound      d - 2zefbrqggv  kd10xb523h
20Gi         RWO             alicloud - disk - ssd     1h
```

```
disk - ssd - web - 2     Bound      d - 2ze4jx1zym  n4n9j3pic2
20Gi         RWO             alicloud - disk - ssd     1h
```

Remove a StatefulSet service

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

2. Run the following command to remove the pod named `web - 1` :

```
$ kubectl delete pod web - 1
pod " web - 1 " deleted
```

3. Run the following command to view the pods and verify that the recreated pod shares the same name with the removed pod:

```
$ kubectl get pod
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
```

4. Run the following command to view the PVCs and verify that the recreated pod uses the same PVC as removed the pod:

```
$ kubectl get pvc
NAME               STATUS     VOLUME                      CAPACITY
  ACCESS   MODES     STORAGECLA  SS          AGE
disk - ssd - web - 0     Bound      d - 2zegw7et6x  c96nbojuoo
20Gi         RWO             alicloud - disk - ssd     1h
disk - ssd - web - 1     Bound      d - 2zefbrqggv  kd10xb523h
20Gi         RWO             alicloud - disk - ssd     1h
disk - ssd - web - 2     Bound      d - 2ze4jx1zym  n4n9j3pic2
20Gi         RWO             alicloud - disk - ssd     1h
```

5. Open a new kubectl interface and run the following command to view the process of pod removal and pod recreation:

```
$ kubectl get pod - w - l app = nginx
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     Terminatin  g     0       89s
web - 1     0 / 1     Terminatin  g     0       89s
web - 1     0 / 1     Terminatin  g     0       90s
web - 1     0 / 1     Terminatin  g     0       90s
web - 1     0 / 1     Pending     0       0s
web - 1     0 / 1     Pending     0       0s
web - 1     0 / 1     ContainerC  reating     0       0s
```

```
web - 1     1 / 1     Running     0       20s
```

Persistent storage of a StatefulSet service

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

```
$ kubectl exec web - 1 ls / data
lost + found
```

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

```
$ kubectl exec web - 1 touch / data / statefulse t
```

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

```
$ kubectl exec web - 1 ls / data
lost + found
statefulse t
```

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

```
$ kubectl delete pod web - 1
pod " web - 1 " deleted
```

5. Run the following command to view the files in the */ data* path and verify that the created file named *statefulse t* has not been removed, indicating that data in the cloud disk can be stored persistently:

```
$ kubectl exec web - 1 ls / data
lost + found
statefulse t
```

3.4 Use a NAS file system when creating a stateful service

This topic describes typical scenarios in which a NAS file system is needed for creating a stateful service, and the procedure for how to use one.

Scenarios and method

If a NAS file system is mounted to multiple pods, the pods share the data in the NAS file system. After a pod modifies the data stored in the NAS file system, the applicatio n supported by the pods is required to automatically update the modified data for the other pods.

Scenarios for using a NAS file system

- You want to create or run applications that demand high disk I/O performance.
- You need a storage service that has higher read and write performance than OSS.

- You want to share files across hosts. For example, you want to use a NAS file system as a file server.

Method of using a NAS file system:

1. Manually create a NAS file system and add a mount point to it.
2. Manually create a PV and a PVC.

This topic describes how to use Alibaba Cloud NAS services in the PV/ PVC mode by using the `flexvolume` plugin provided by Alibaba Cloud.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have connected to the Kubernetes cluster by using kubectl, see *Connect to a Kubernetes cluster by using kubectl*.
- You have created a NAS file system in the NAS console. For more information, see *Create a file system*. You must make sure that the NAS file system and your Kubernetes cluster are in the same zone.
- You have added a mount point for your Kubernetes cluster in the created NAS file system. For information, see *Add a mount point*. You must make sure that the NAS file system and your cluster are in the same VPC.

Create a PV

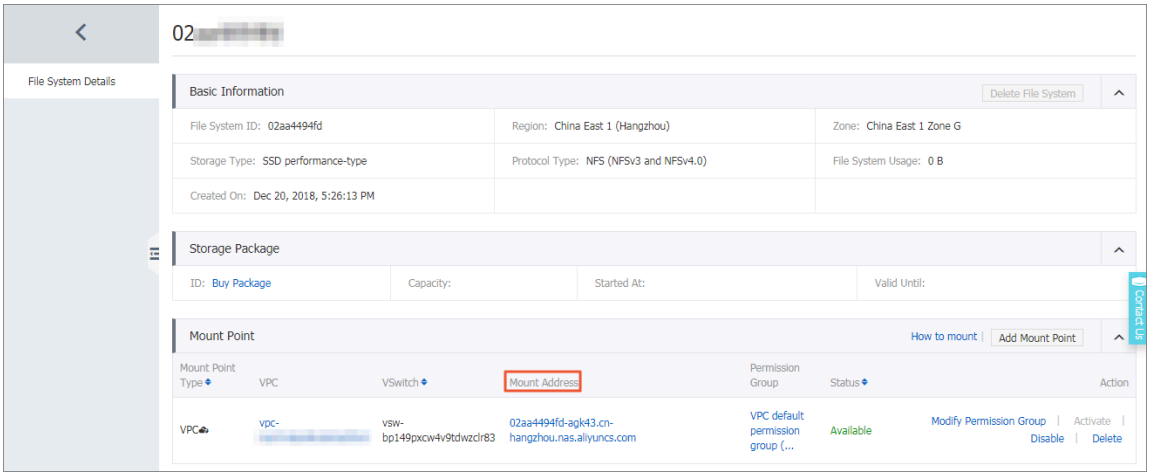
1. Create a `pv-nas.yaml` file.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nas
  labels:
    alicloud-pvname: pv-nas
spec:
  capacity:
    storage: 5Gi
  accessModes:
   - ReadWriteMany
  flexVolume:
    driver: "alicloud/nas"
    options:
      server: "***-**.cn-hangzhou.nas.aliyuncs.com"
 //// Replace this value with your mount point.
      path: "/k8s1"
```

```
        vers : " 4 . 0 "
```

Parameter description

- `alicloud - pvname` : indicates a PV name.
- `server` : indicates a NAS mount point. To view your mount point, log on to the NAS console, click File System List in the left-side navigation pane, select the target file system, and click Manage in the Action column to view the Mount Address in the Mount Point area. The mount address is the mount point of your NAS file system.



- `path` : indicates the NAS mount directory. You can mount a NAS sub-directory to your cluster. If the NAS sub-directory specified by you does not exist, the system automatically creates the NAS sub-directory and mounts it to your cluster.
- `vers` : (optional) indicates the version number of the NFS mount protocol. NFS file system V3.0 and V4.0 are available. The default is V4.0.
- `mode` : (optional) indicates the access permission to the mount directory. By default, this parameter is not set.

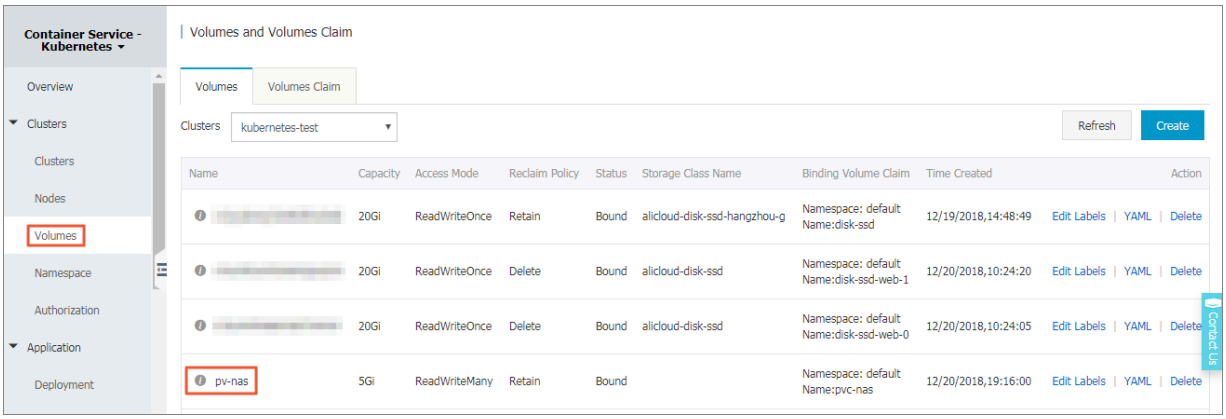
> **Note:**
>
> - Access permission to the root directory of the NAS file system cannot be set.
> - If you set the `mode` parameter for a NAS file system that stores a large amount of data, the process of mounting the NAS file system to a cluster may take an excessive amount of time or even fail. We recommend that you do not set this parameter.

2. Run the following command to create a PV:

```
$ kubectl create -f pv-nas.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Clusters > Volumes, and select the target cluster to view the created PV.



Create a PVC

Create a PVC for the NAS file system. Specifically, you need to set the `selector` field to filter for the created PV so that you can associate the PVC with the correct PV.

1. Create a *pvc-nas.yaml* file.

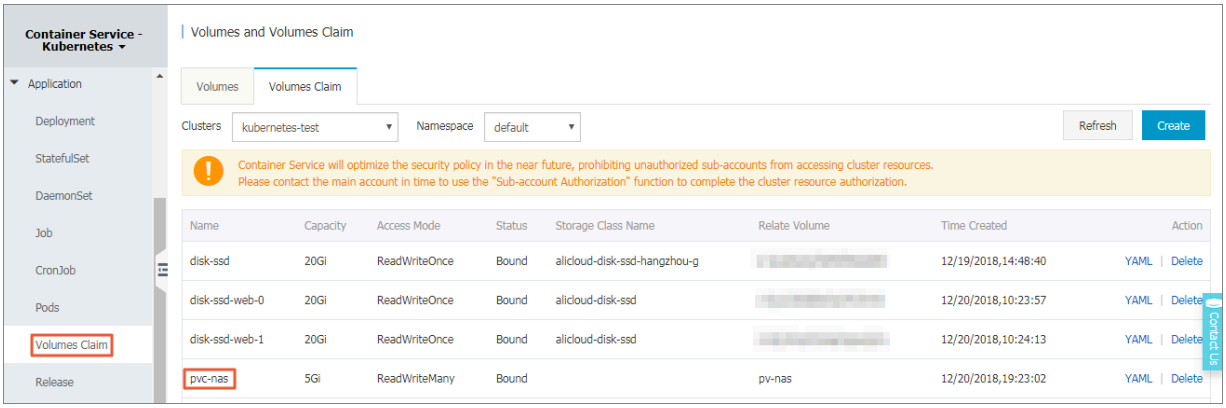
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-nas
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-nas
```

2. Run the following command to create a PVC:

```
$ kubectl create -f pvc-nas.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Volumes Claim, and select the target cluster and namespace to view the created PVC.



Create an application

1. Create a *nas.yaml*.

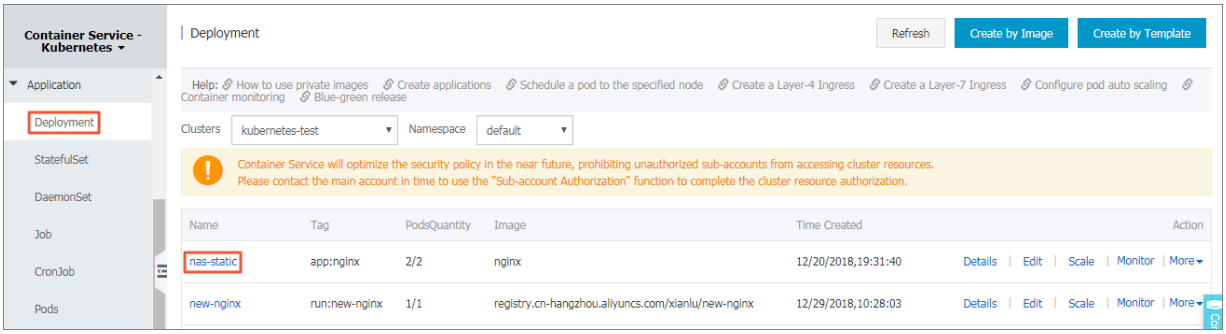
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nas-static
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-nas
            mountPath: "/data"
      volumes:
        - name: pvc-nas
          persistentVolumeClaim:
            claimName: pvc-nas
```

2. Run the following command to create a deployment:

```
$ kubectl create -f nas.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Deployment, and select the target cluster and namespace to view the created deployment.



Verify that the NAS file system is shared by pods

1. Run the following command to view the pods in which the created deployment resides:

```
$ kubectl get pod
NAME                                READY     STATUS    RESTARTS
  AGE
nas-static-f96b6b5d7-rcb2f          1/1       Running   0
          9m
nas-static-f96b6b5d7-wthmb          1/1       Running   0
          9m
```

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

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f ls /data
```

```
$ kubectl exec nas-static-f96b6b5d7-wthmb ls /data
```

> **Note:**
> The two */data* paths are empty.

3. Run the following command to create file *nas* in the */data* path of one pod:

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f touch /data
/nas
```

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

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f ls /data
nas
```

```
$ kubectl exec nas-static-f96b6b5d7-wthmb ls /data
nas
```

> **Note:**
> After you create the file in the */data* path of one pod, the file then exists in both the */data* paths of the two pods. This means that the two pods share the NSA file system.

Verify that data on the NAS file system are stored persistently

1. Run the following command to remove all the pods of the created application:

```
$ kubectl delete pod nas-static-f96b6b5d7-rcb2f nas-
static-f96b6b5d7-wthmb
pod "nas-static-f96b6b5d7-rcb2f" deleted
pod "nas-static-f96b6b5d7-wthmb" deleted
```

2. Open another kubectl interface and run the following command to view the process in which the original pods are removed and new pods are created by Kubernetes:

```
$ kubectl get pod -w -l app=nginx
NAME                              READY     STATUS      RESTARTS
  AGE
nas-static-f96b6b5d7-rcb2f        1/1       Running     0
          27m
nas-static-f96b6b5d7-wthmb        1/1       Running     0
          27m
nas-static-f96b6b5d7-rcb2f    1/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-wnqdj    0/1     Pending     0
0s
nas-static-f96b6b5d7-wnqdj    0/1     Pending     0
0s
nas-static-f96b6b5d7-wnqdj    0/1     ContainerC  reating
   0       0s
nas-static-f96b6b5d7-wthmb    1/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-nwkds    0/1     Pending     0
0s
nas-static-f96b6b5d7-nwkds    0/1     Pending     0
0s
nas-static-f96b6b5d7-nwkds    0/1     ContainerC  reating
   0       0s
nas-static-f96b6b5d7-rcb2f    0/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-wthmb    0/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-rcb2f    0/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-rcb2f    0/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-wnqdj    1/1     Running     0
10s
nas-static-f96b6b5d7-wthmb    0/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-wthmb    0/1     Terminating  g     0
     28m
nas-static-f96b6b5d7-nwkds    1/1     Running     0
17s
```

3. Run the following command to view the new pods created by Kubernetes:

```
$ kubectl get pod
NAME                              READY     STATUS      RESTARTS
  AGE
nas-static-f96b6b5d7-nwkds        1/1       Running     0
          21s
```

```
nas - static - f96b6b5d7 - wnqdj          1 / 1        Running      0
          21s
```

4. Run the following commands to view the files in the `/ data` path of each pod:

```
$ kubectl exec nas - static - f96b6b5d7 - nwkds ls / data
nas
```

```
$ kubectl exec nas - static - f96b6b5d7 - wnqdj ls / data
nas
```

Note:

The created file, namely, file `nas` has not been removed. This means that data in the NAS file system can be stored persistently.

3.5 Use an OSS bucket when creating a stateful service

This topic describes typical scenarios in which an Object Storage Service (OSS) bucket is needed for creating a stateful service, and the procedure for how to use the bucket.

Scenarios and method

Alibaba Cloud OSS provides massive, secure, low-cost, and highly reliable cloud storage services. An OSS bucket can be mounted to multiple pods.

Scenarios:

- Disk I/O performance requirements are low.
- Shared services such as files, figures, and short videos are to be configured.

Method:

1. Manually create a bucket.
2. Obtain the AccessKey ID and AccessKey Secret pair.
3. Manually create a Persistent Volume (PV) and a Persistent Volume Claim (PVC).

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have connected to the Kubernetes cluster by using kubectl, see *Connect to a Kubernetes cluster by using kubectl*.
- You have created a bucket in the OSS console, see *Create a bucket*.

Precautions

- Upgrading a Kubernetes cluster of Alibaba Cloud Container Service causes kubelet and the OSSFS driver to restart. As a result, the OSS directory becomes unavailable. In this case, the pods that use the OSS bucket must be recreated. We recommend that you add health check settings in the YAML file of your application. If you add health check settings for your application, the pods will be automatically restarted to remount the OSS bucket when the OSS directory within your container becomes unavailable.
- If you use a Kubernetes cluster of the latest version, the preceding issue does not affect you.

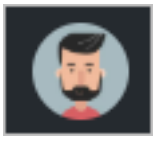
Create a PV

1. Create a *pv-oss.yaml* file.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-oss
  labels:
    alicloud-pvname: pv-oss
spec:
  capacity:
    storage: 5Gi
  accessModes:
   - ReadWriteMany
  storageClassName: oss
  flexVolume:
    driver: "alicloud/oss"
    options:
      bucket: "docker"                          //// Replace this value with your bucket name.
      url: "oss-cn-hangzhou.aliyuncs.com"     //// Replace this value with your URL.
      akId: "***"                               //// Replace this value with your AccessKey ID.
      akSecret: "***"                           //// Replace this value with your AccessKey Secret.
```

```
      otherOpts : "- o   max_stat_c  ache_size = 0  - o
allow_othe  r "   //// Replace   this   value   with   your
specified   otherOpts   value .
```

Parameter description

- `alicloud - pvname` : indicates a PV name. This parameter value must be used in the `selector` field of the PVC associated with the PV.
- `bucket` : indicates a bucket name. Only buckets can be mounted to a Kubernetes cluster. The sub-directories or files in a bucket cannot be mounted to any Kubernetes cluster.
- `url` : indicates a domain name used to access the OSS bucket, namely, an endpoint. For more information, see *Regions and endpoints*. You can also view the endpoint of the created OSS bucket in the OSS console. That is, log on to the OSS console, select the target bucket, and view Endpoint in the Domain Names area.
- `akId` : indicates your AccessKey ID. In the Container Service console, click  in the upper-right corner. For a primary account, select accesskeys.

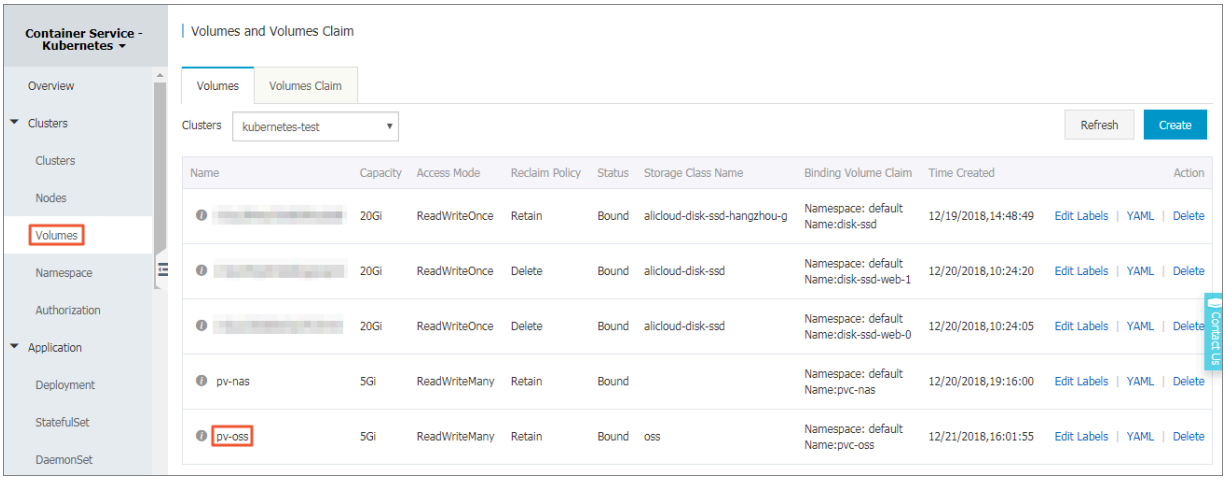
  For a RAM user, select AccessKey. Then, you can create your AccessKey ID and AccessKey Secret.
- `akSecret` : indicates your AccessKey Secret. Use the same method to obtain this parameter value as that to obtain the value of the `akId` parameter.
- `otherOpts` : indicates custom parameters for mounting the OSS bucket. Set this parameter in the format of `- o *** - o ***`. For more information, see *FAQ*.

2. Run the following command to create a PV:

```
$ kubectl  create  - f  pv - oss . yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Clusters > Volumes, and select the target cluster to view the created PV.



Create a PVC

Create a PVC for the OSS bucket. Specifically, you need to set the `selector` field to filter for the created PV so that you can associate the PVC with the correct PV. Set the `storageClassName` parameter to associate the PVC with only the PV of the OSS type.

1. Create a *pvc-oss.yaml* file.

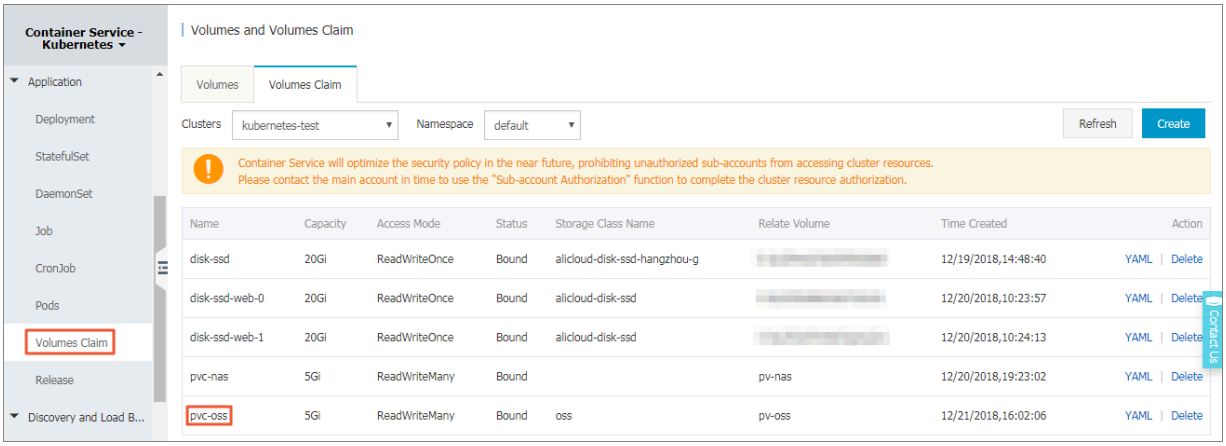
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-oss
spec:
  accessModes:
   - ReadWriteMany
  storageClassName: oss
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-oss
```

2. Run the following command to create a PVC:

```
$ kubectl create -f pvc-oss.yaml
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Volumes Claim, and select the target cluster and namespace to view the created PVC.



Create an application

1. Create an `oss-static.yaml` file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oss-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: pvc-oss
          mountPath: "/data"
        - name: pvc-oss
          mountPath: "/data1"
        livenessProbe:
          exec:
            command:
            - sh
            - -c
            - cd /data
          initialDelaySeconds: 30
          periodSeconds: 30
      volumes:
      - name: pvc-oss
        persistentVolumeClaim:
          claimName: pvc-oss
```

Note:

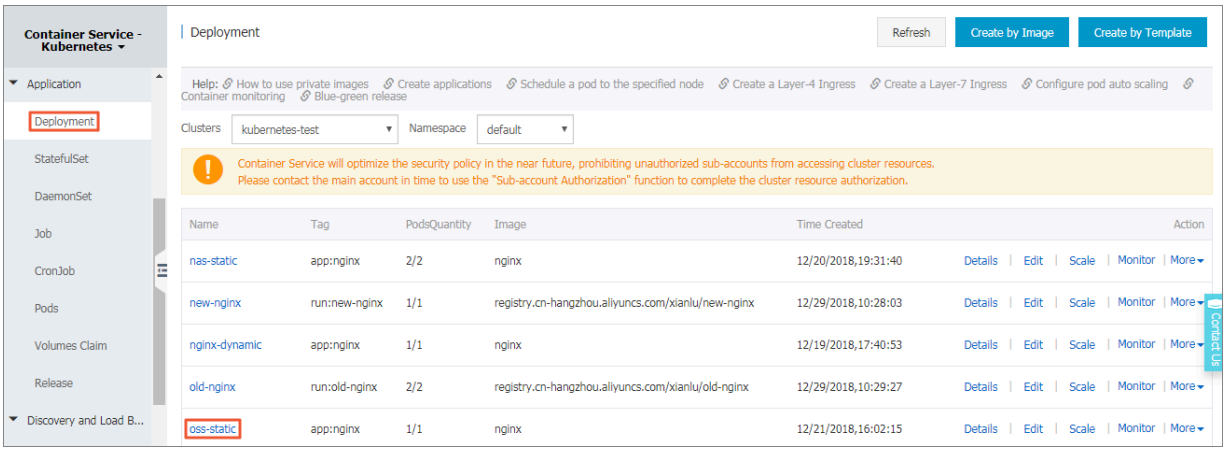
For more information about `livenessProbe`, see *Use Alibaba Cloud OSS volumes*.

2. Run the following command to create a deployment:

```
$ kubectl create -f oss-static.yaml -d
```

Result

In the left-side navigation pane under Kubernetes, choose Application > Deployment, and select the target cluster and namespace to view the created deployment.



Verify that data in the OSS bucket are stored persistently

1. Run the following command to view the pod in which the created deployment resides:

```
$ kubectl get pod
NAME                                READY     STATUS    RESTARTS
   AGE
oss-static-66fbb85b67-dqbl2         1/1       Running   0
          1h
```

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

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 ls /data
 | grep tmpfile
```

Note:

The */data* path is empty.

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

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 touch /
 data/tmpfile
```

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

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 ls /data
 | grep tmpfile
```

```
tmpfile
```

5. Run the following command to remove the pod named *oss-static-66fbb85b67-dqbl2*:

```
$ kubectl delete pod oss-static-66fbb85b67-dqbl2
pod "oss-static-66fbb85b67-dqbl2" deleted
```

6. Open another kubectl interface and run the following command to view the process in which the preceding pod is removed and a new pod is created by Kubernetes:

```
$ kubectl get pod -w -l app=nginx
NAME                                READY     STATUS      RESTARTS
  AGE
oss-static-66fbb85b67-dqbl2         1/1       Running     0
          78m
oss-static-66fbb85b67-dqbl2     1/1     Terminating     0
      78m
oss-static-66fbb85b67-zlvmw     0/1     Pending     0
<invalid>
oss-static-66fbb85b67-zlvmw     0/1     Pending     0
<invalid>
oss-static-66fbb85b67-zlvmw     0/1     ContainerCreating
    0      <invalid>
oss-static-66fbb85b67-dqbl2     0/1     Terminating     0
      78m
oss-static-66fbb85b67-dqbl2     0/1     Terminating     0
      78m
oss-static-66fbb85b67-dqbl2     0/1     Terminating     0
      78m
oss-static-66fbb85b67-zlvmw     1/1     Running     0
<invalid>
```

7. Run the following command to view the pod created by Kubernetes:

```
$ kubectl get pod
NAME                                READY     STATUS      RESTARTS
  AGE
oss-static-66fbb85b67-zlvmw         1/1       Running     0
          40s
```

8. Run the following command to verify that the created file named *tmpfile* in the */data* path has not been removed, indicating that data in the OSS bucket can be stored persistently:

```
$ kubectl exec oss-static-66fbb85b67-zlvmw ls /data
| grep tmpfile
tmpfile
```

4 Release

4.1 Implement Layer-4 canary release by using Alibaba Cloud Server Load Balancer in a Kubernetes cluster

In a Kubernetes cluster, Layer-7 Ingress cannot properly implement gray release for services accessed by using TCP/UDP. This document introduces how to implement Layer-4 canary release by using Server Load Balancer.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *#unique_34*.
- You have connected to the master node by using SSH. For more information, see *#unique_35*.

Step 1 Deploy the old version of the service

1. Log on to the *Container Service console*.
2. Click Application > Deployment in the left-side navigation pane.
3. Click Create by template in the upper-right corner.

4. Select the cluster and namespace from the Clusters and Namespace drop-down lists. Select a sample template or Custom from the Resource Type drop-down list. Click DEPLOY.

   In this example, an nginx orchestration that exposes the service by using SLB.

```
apiVersion :  extensions / v1beta1
 kind :  Deployment
 metadata :
   labels :
     run :  old - nginx
   name :  old - nginx
 spec :
   replicas :  1
   selector :
     matchLabel  s :
       run :  old - nginx
   template :
     metadata :
       labels :
         run :  old - nginx
         app :  nginx
     spec :
```

```
        containers :
       -  image :  registry . cn - hangzhou . aliyuncs . com /
xianlu / old - nginx
          imagePullP  olicy :  Always
          name :  old - nginx
          ports :
         -  containerP  ort :  80
            protocol :  TCP
       restartPol  icy :  Always
---
 apiVersion :  v1
 kind :  Service
 metadata :
   labels :
     run :  nginx
   name :  nginx
 spec :
   ports :
  -  port :  80
     protocol :  TCP
     targetPort :  80
   selector :
     app :  nginx
   sessionAff  inity :  None
   type :  LoadBalanc  er  ## Expose   the   service   by   using
Alibaba   Cloud   SLB .
```

5. **Click Application > Deployment and Application > Service in the left-side navigation pane to check the deployment and service.**

6. **Click the external endpoint at the right of the service to go to the Nginx default welcome page. In this example, old is displayed on the Nginx welcome page, which indicates that the currently accessed service corresponds to the backend old-nginx container.**

   To easily display the results of multiple releases , we recommend that you log on to the master node and execute the `curl` command to view the deployment results.

```
#  bash
#  for   x   in  { 1 ..  10 } ;  do   curl   EXTERNAL - IP ;  done
  ## EXTERNAL - IP   is   the   external   endpoint   of   the
 service .
 old
 old
 old
 old
 old
 old
 old
 old
 old
```

```
old
```

Step 2 Bring new deployment version online

1. Log on to the *Container Service console*.
2. Click Application > Deployment in the left-side navigation pane.
3. Click Create by template in the upper-right corner.
4. Select the cluster and namespace from the Clusters and Namespace drop-down lists. Select a sample template or Custom from the Resource Type drop-down list. Click DEPLOY.

   In this example, create a new version of nginx deployment that contains the `app : nginx` label. The label is used to use the same nginx service as that of the old version of deployment to bring the corresponding traffic.

   The orchestration template in this example is as follows:

```
apiVersion :  extensions / v1beta1
 kind :  Deployment
 metadata :
   labels :
     run :  new - nginx
   name :  new - nginx
 spec :
   replicas :  1
   selector :
     matchLabel  s :
       run :  new - nginx
   template :
     metadata :
       labels :
         run :  new - nginx
         app :  nginx
     spec :
       containers :
       -  image :  registry . cn - hangzhou . aliyuncs . com /
xianlu / new - nginx
         imagePullP  olicy :  Always
         name :  new - nginx
         ports :
         -  containerP  ort :  80
            protocol :  TCP
       restartPol  icy :  Always
```

5. Click Deployment in the left-side navigation pane. The deployment of new-nginx is displayed on the Deployment page.

6. Log on to the master node and execute the curl command to view the service access.

```
#  bash
#  for   x   in  { 1 ..  10 } ;  do   curl   EXTERNAL - IP ;  done
  ## EXTERNAL - IP   is   the   external   endpoint   of   the
 service .
 new
 new
 new
 old
 new
 old
 new
 new
 old
 old
```

You can see that the old service and new service are accessed for five times respectively. This is mainly because the service follows the Server Load Balancer policy of average traffic to process traffic requests, and the old deployment and new deployment are the same pod, which makes their traffic ratio as 1:1.

Step 3 Adjust traffic weight

You must adjust the number of pods in the backend to adjust the corresponding weight for the canary release based on Server Load Balancer. For example, to make the new service to have higher weight, you can adjust the number of new pods to four .

Note:

If the old application version and new application version coexist, the results returned after executing the curl command of a sample do not conform to the configured weight strictly. In this example, to obtain the approximate effect, execute the curl command for 10 times to observe more samples.

1. Log on to the *Container Service console*.
2. Under Kubernetes, click Application > Deployment in the left-side navigation pane.
3. Select the cluster and namespace from the Clusters and Namespace drop-down lists. Click Update at the right of the deployment.

4. In the displayed dialog box, set the number of pods to four.

Note:
The default update method of Kubernetes deployment resources is rollingUpdate. Therefore, during the update process, the minimum number of containers that provide the service is guaranteed and this number can be adjusted in the template.

5. After the deployment, log on to the master node and execute the curl command to view the effect.

```
#  bash
 #  for   x   in  { 1 ..  10 } ;  do   curl   EXTERNAL - IP ;  done
  ## EXTERNAL - IP   is   the   external   endpoint   of   the
 service .
  new
  new
  new
  new
  new
  old
  new
  new
  new
  old
```

You can see the new service is requested for eight times and the old service is requested twice among the 10 requests.

You can dynamically adjust the number of pods to adjust the weights of the new service and old service and implement the canary release.

4.2 Implement a gray release and a blue/green deployment through Ingress in a Kubernetes cluster

4.2.1 Gray releases and blue/green deployment

This topic describes how to implement a gray release and a blue/green deployment by using the Ingress function provided by Alibaba Cloud Container Service for Kubernetes.

Background information

With a gray release or a blue/green deployment, you can create two identical production environments for the latest version of the target software and an earlier

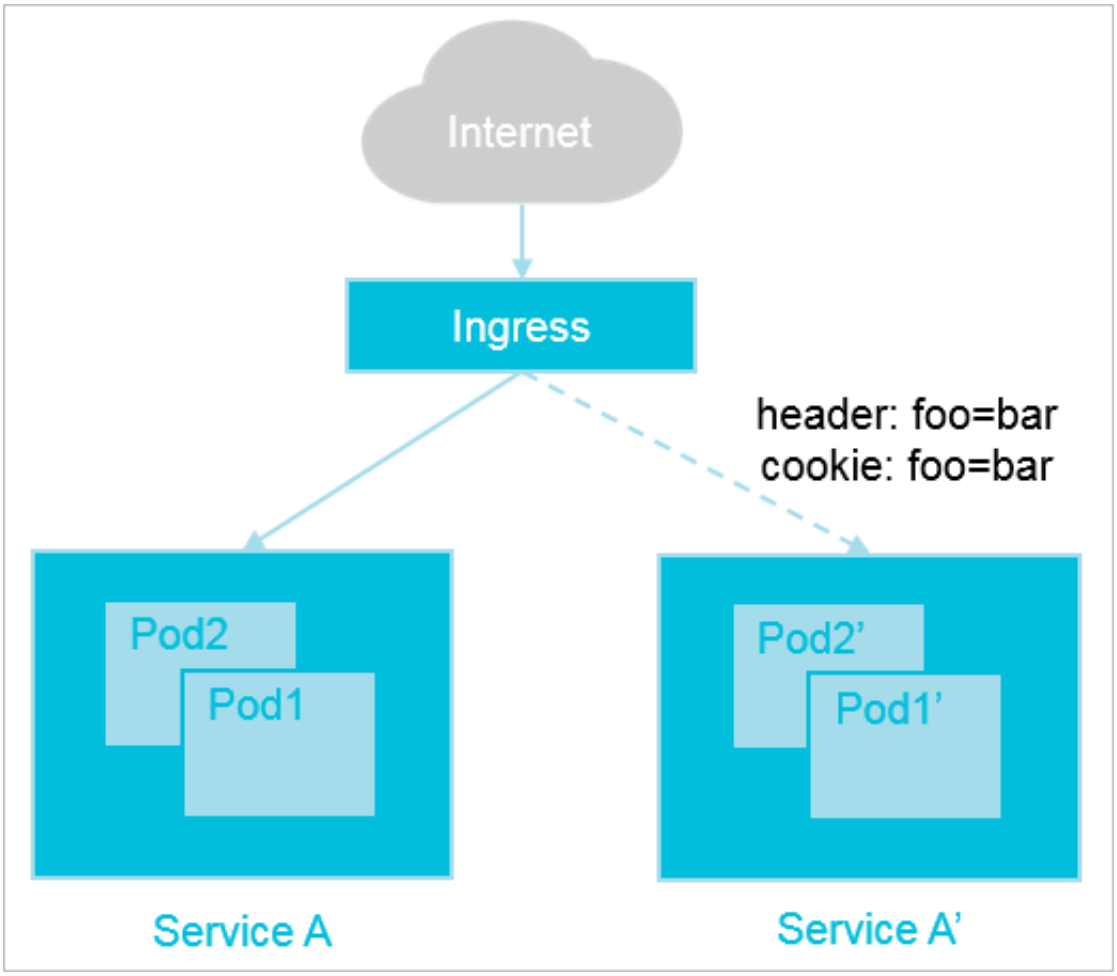
version. Then you can apply specific rules to reroute traffic from the earlier version to the latest version without affecting the software of the earlier version. After the software of the latest version has run without exceptions for a specified period, you can reroute all traffic from the earlier version to the latest version.

A/B testing is a type of comparative and incremental gray release. Specifically, with A/B testing, you can keep some users using the service of an earlier version, and reroute traffic of other users to the service of the latest version. If the service of the latest version runs without exceptions for the specified period of time, then you can gradually reroute all user traffic to the service of the latest version.

Scenarios

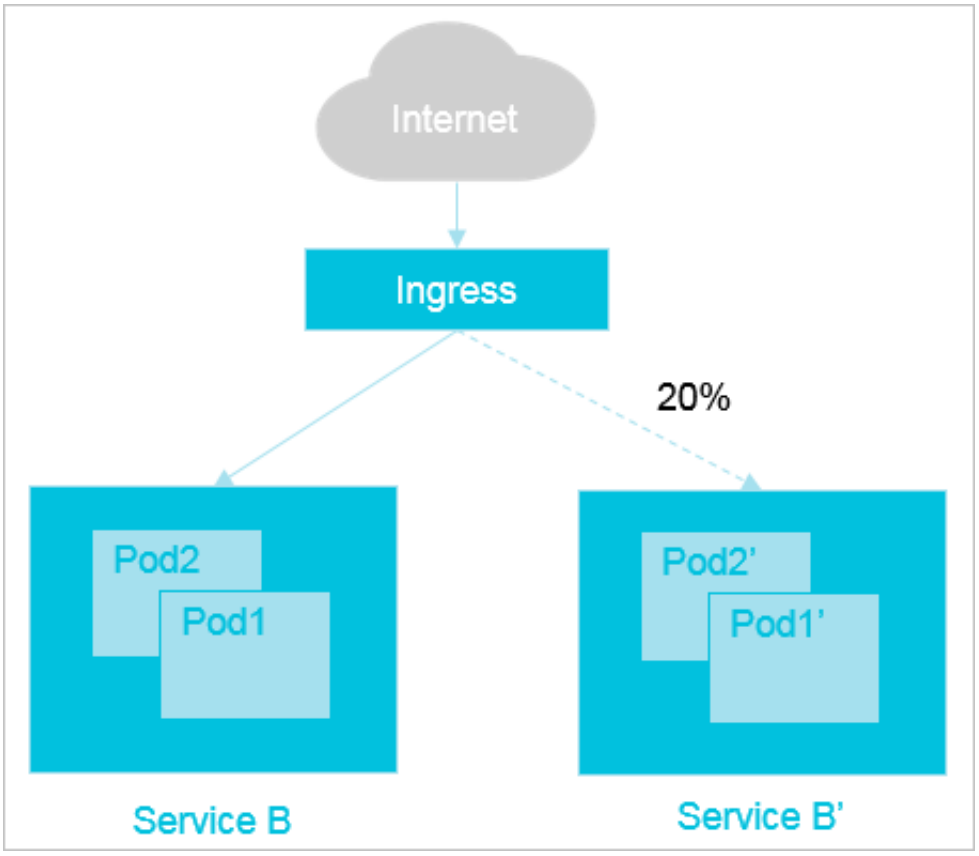
Scenario 1

For example, assume that Service A already runs online to provide an externally accessible Layer-7 service, and a new version of this service with new features, namely, Service A', is developed. You want to release Service A', but you do not want it to directly replace Service A at once. Additionally, you want the client requests of which the request headers contain `foo = bar` or the cookies contain `foo = bar` to be forwarded to Service A'. Then, after Service A' has run without exceptions for a specified period, you want to reroute all traffic from Service A to Service A', and then smoothly bring Service A offline.



Scenario 2

For example, assume that an earlier version of a service, named Service B, is running online to provide an externally accessible Layer-7 service. However, it has known problems. A new version, namely Service B' is developed with the problems fixed and you want to release this latest version. However, you initially want to reroute only 20% of all client traffic to Service B’. Then, after Service B’ has run without exceptions for a period, you want to reroute all traffic from Service B to Service B’, and then smoothly bring Service B offline.



To meet the preceding application release requirements, Alibaba Cloud Container Service for Kubernetes uses the Ingress function to provide the following four methods of traffic distribution:

In A/B testing

- Distribute traffic according to the request header
- Distribute traffic according to the cookie
- Distribute traffic according to the Query Param

In a blue/green deployment

- Distribute traffic according to the service weight

4.2.2 Gray release limits

This topic describes the limits for a gray release that is implemented by the Ingress function provided by Alibaba Cloud Container Service for Kubernetes.

The Ingress controller of Alibaba Cloud Container Service for Kubernetes must be V `0.12.0-5` or later.

To view the version number of the Ingress controller, run either of the following commands as required.

- For a cluster in which applications are deployed by using the Deployment method, run:

```
kubectl - n kube - system get deploy nginx - ingress - controller - o yaml | grep - v ' apiVersion ' | grep ' aliyun - ingress - controller '
```

- For a cluster in which applications are deployed by using the DaemonSet method, run:

```
kubectl - n kube - system get ds nginx - ingress - controller - o yaml | grep - v ' apiVersion ' | grep ' aliyun - ingress - controller '
```

If your Ingress Controller is earlier than `0 . 12 . 0 - 5` , you can upgrade it by running either of the following commands as required.

- For a cluster in which applications are deployed by using the Deployment method, run:

```
kubectl - n kube - system set image deploy / nginx - ingress - controller nginx - ingress - controller = registry . cn - hangzhou . aliyuncs . com / acs / aliyun - ingress - controller : 0 . 12 . 0 - 5
```

- For a cluster in which applications are deployed by using the DaemonSet method, run:

```
kubectl - n kube - system set image ds / nginx - ingress - controller nginx - ingress - controller = registry . cn - hangzhou . aliyuncs . com / acs / aliyun - ingress - controller : 0 . 12 . 0 - 5
```

4.2.3 Annotation

This topic describes the annotation used when you implement a gray release by using the Ingress function provided by Alibaba Cloud Container Service for Kubernetes.

To support a gray release, the Ingress function of Alibaba Cloud Container Service for Kubernetes provides the following annotation: routing rules set by using `nginx . ingress . kubernetes . io / service - match` and service weight set by using `nginx . ingress . kubernetes . io / service - weight` .

Note:

If you set routing rules by using `nginx . ingress . kubernetes . io / service - match` and service weight by using `nginx . ingress . kubernetes . io /`

`service-weight`, the system first determines whether the routing rules set by using `nginx.ingress.kubernetes.io/service-match` are matched when receiving a request:

- If no routing rules are matched, the system forwards the request to the application of the earlier version.
- If the routing rules are matched, the system forwards the request according to the service weight that you set by using `nginx.ingress.kubernetes.io/service-weight`.

Routing rules set by using `nginx.ingress.kubernetes.io/service-match`

This annotation is used to set the routing rules for the service of the latest version. The annotation format is as follows:

```
nginx.ingress.kubernetes.io/service-match: |
    <service-name>: <match-rule>
```

Parameter description

`service-name`: service name. The requests that meet the requirements of the route matching rules are routed to this service.

`match-rule`: matching rules of routes.

- Matching types:
  - `header`: based on the request header. This matching type supports regular expression matches and full expression matches.
  - `cookie`: based on the cookie. This matching type supports regular expression matches and full matches.
  - `query`: based on the queried parameter. This matching type supports regular expression matches and full matches.
- Matching methods:
  - The format of a regular expression match is `/{Regular Expression}/`.
  - The format of a full match is `"{exact expression}"`

Configuration examples

```
# If the request header of a request meets the
 requiremen ts of the regular expression of foo and
 ^bar$, the request is forwarded to the new-nginx
 service.
 new-nginx: header("foo", /^bar$/)
```

```
# In the request header of a request , if foo fully
  matches bar , the request will be forwarded to the
 new - nginx service .
 new - nginx : header (" foo ", " bar ")

# In the cookie of a request , if foo matches the
  regular expression ^ sticky -.+$, the request will be
 forwarded to the new - nginx service .
 new - nginx : cookie (" foo ", /^ sticky -.+$/)

# In the query param of a request , if foo fully
 matches bar , the request will be forwarded to the
 new - nginx service .
 new - nginx : query (" foo ", " bar ")
```

Service weight set by using `nginx . ingress . kubernetes . io / service - weight`

This annotation is used to set the traffic weights for the service of the latest version and the service of the earlier version. The annotation format is as follows:

```
nginx . ingress . kubernetes . io / service - weight : |
   < new - svc - name >:< new - svc - weight >, < old - svc - name
>:< old - svc - weight >
```

Parameter description

`new - svc - name` : name of the service of the latest version.

`new - svc - weight` : weight of the service of the latest version.

`old - svc - name` : name of the service of the earlier version.

`old - svc - weight` : weight of the service of the earlier version.

Configuration examples

```
nginx . ingress . kubernetes . io / service - weight : |
    new - nginx : 20 , old - nginx : 60
```

Note:

- Service weights are calculated by using relative values. In the preceding example, the service of the latest version is set to `20` weight and the service of the earlier version is set to `60` weight. Therefore, the weight percentage of the latest version service is 25% and the weight percentage of the earlier version service is 75%.
- In a service group that is composed of services that have the same host and path in an Ingress YAML, the default service weight is 100.

4.2.4 Step 1: Deploy a service

This topic describes how to deploy a service.

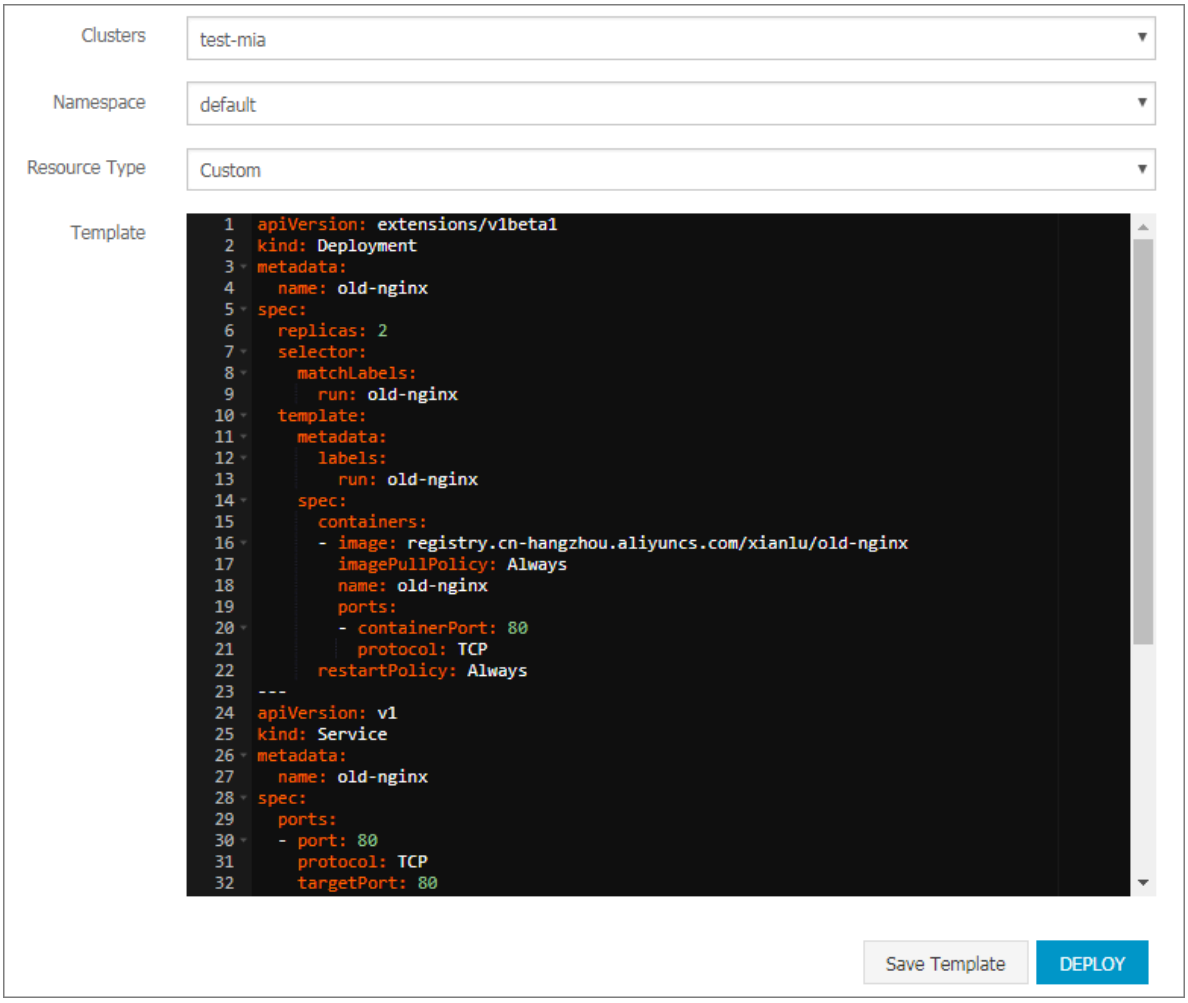
Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have connected to the Kubernetes cluster by using kubectl. For more information, see *Connect to a Kubernetes cluster by using kubectl*.

Procedure

1. Log on to the *Container Service console*.
2. In the left-side navigation pane under Kubernetes, choose Application > Deployment.
3. In the upper-right corner, click Create by Template.

4. Select the target cluster and namespace, select a sample template or customize a template, and then click DEPLOY.



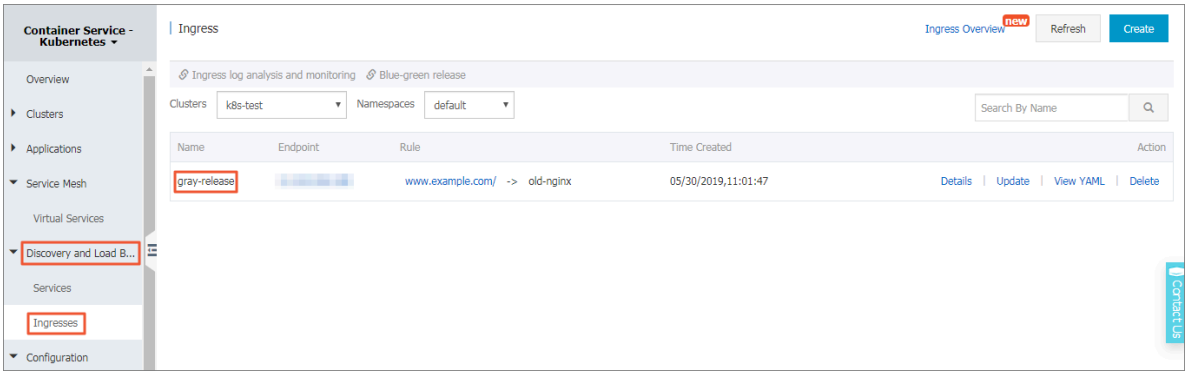
In this example, a template is orchestrated to deploy an Nginx application that contains the required deployment, the target service, and an Ingress. The deployment exposes its port through NodePort. The Ingress provides externally accessible services. The orchestration template is as follows:

```
apiVersion :  extensions / v1beta1
kind :  Deployment
metadata :
  name :  old - nginx
spec :
  replicas :  2
  selector :
    matchLabel  s :
      run :  old - nginx
  template :
    metadata :
      labels :
        run :  old - nginx
    spec :
      containers :
     -  image :  registry . cn - hangzhou . aliyuncs . com / xianlu
/ old - nginx
```

```
          imagePullP  olicy :  Always
          name :  old - nginx
          ports :
         -  containerP  ort :  80
            protocol :  TCP
       restartPol  icy :  Always
---
 apiVersion :  v1
 kind :  Service
 metadata :
   name :  old - nginx
 spec :
   ports :
  -  port :  80
     protocol :  TCP
     targetPort :  80
   selector :
     run :  old - nginx
   sessionAff  ity :  None
   type :  NodePort
---
 apiVersion :  extensions / v1beta1
 kind :  Ingress
 metadata :
     name :  gray - release
 spec :
   rules :
  -  host :  www . example . com
     http :
       paths :
      #  earlier   version   of   a   service
      -  path : /
         backend :
           serviceNam  e :  old - nginx
           servicePor  t :  80
```

5. In the left-side navigation pane, choose Application > Ingress.

   You can see that the virtual host name points to old-nginx.



6. Log on to the Master node and run the curl command to view the Ingress.

```
#  curl  - H  " Host :  www . example . com "  http ://< EXTERNAL_I
 P >
```

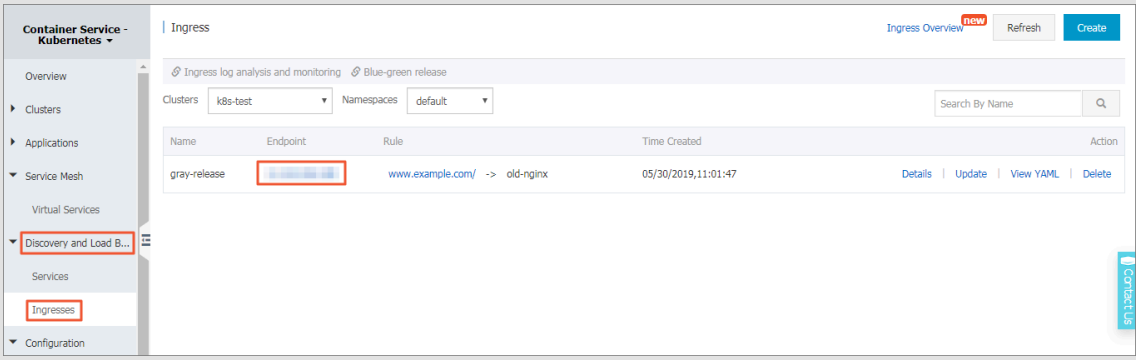
Note:

You can obtain the value of <EXTERNAL_IP> by using either of the following two methods:

- Run the following command:

```
kubectl get ingress
```

- Under the Kubernetes menu, choose Application > Ingress, and view the endpoint information of the target Ingress.




```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
```

4.2.5 Step 2: Release the latest version of a service

This topic describes how to release the latest version of a service by using the Ingress function provided by Alibaba Cloud Container Service for Kubernetes.

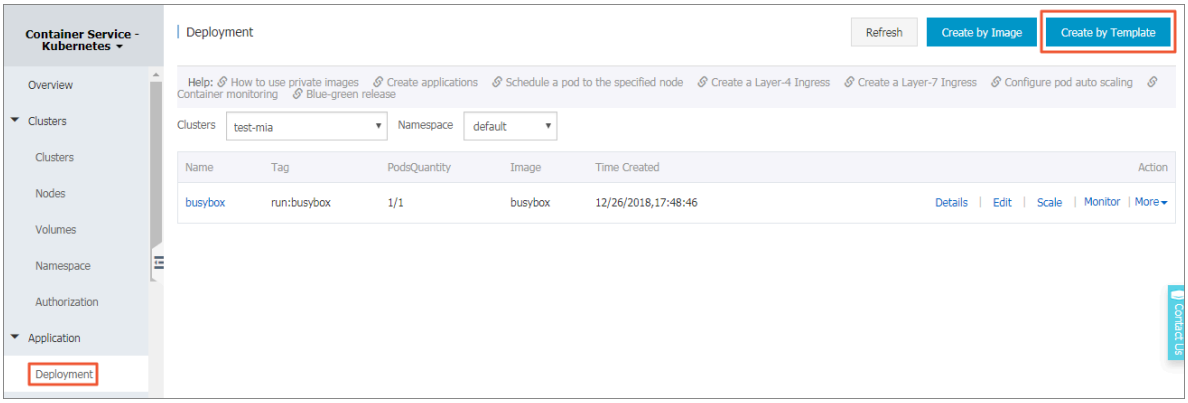
Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have connected to the Kubernetes cluster by using kubectl. For more information, see *Connect to a Kubernetes cluster by using kubectl*.

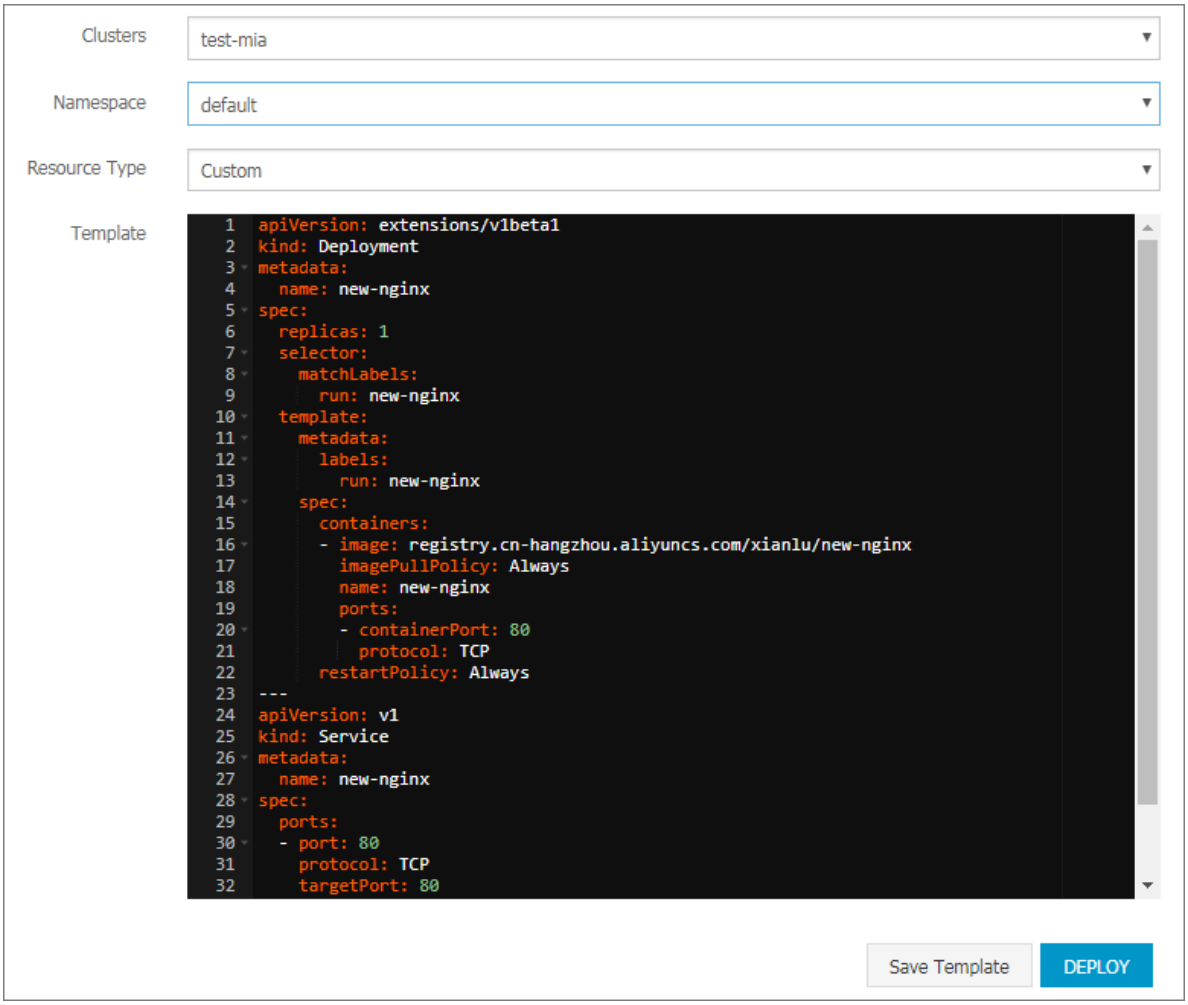
Procedure

1. Log on to the *Container Service console*.
2. In the left-side navigation pane under Kubernetes, choose Application > Deployment.

3. In the upper-right corner, click Create by Template.



4. Select the target cluster and namespace, select a sample template or customize a template, and then click DEPLOY.



Deploy an Nginx application of the latest version that contains the required deployment, the target service, and an Ingress. The orchestration template that contains the deployment and service is as follows:

```
apiVersion :  extensions / v1beta1
kind :  Deployment
```

```
 metadata :
   name :  new - nginx
 spec :
   replicas :  1
   selector :
     matchLabel  s :
       run :  new - nginx
   template :
     metadata :
       labels :
         run :  new - nginx
     spec :
       containers :
        -  image :  registry . cn - hangzhou . aliyuncs . com / xianlu
 / new - nginx
         imagePullP  olicy :  Always
         name :  new - nginx
         ports :
          -  containerP  ort :  80
             protocol :  TCP
       restartPol  icy :  Always
---
 apiVersion :  v1
 kind :  Service
 metadata :
   name :  new - nginx
 spec :
   ports :
  -  port :  80
     protocol :  TCP
     targetPort :  80
   selector :
     run :  new - nginx
   sessionAff  inity :  None
   type :  NodePort
```

The following are Ingress orchestration templates of different annotation settings:

Note:

If you do not set `service - match` or `service - weight` in the annotations field of an Ingress template, the Ingress controller forwards client requests evenly to the latest and earlier services in a random manner.

- Ingress template used to specify only the client requests that meet the requirement of the regular expression foo=bar to be routed to the latest version of the service

```
  apiVersion :  extensions / v1beta1
  kind :  Ingress
  metadata :
    name :  gray - release
    annotation  s :
        nginx . ingress . kubernetes . io / service - match : |
   #  Only  if  the  request  header  of  a  request  meets
    the  requiremen  ts  of  the  regular  expression  foo =
  bar ,  can  the  request  be  routed  to  the  new - nginx
    service .
```

```
          new - nginx :  header (" foo ", /^ bar $/)
spec :
  rules :
 -  host :  www . example . com
    http :
      paths :
     #  Earlier   version   of   the   service
     -  path : /
        backend :
          serviceNam  e :  old - nginx
          servicePor  t :  80
     #  Latest   version   of   the   service
     -  path : /
        backend :
          serviceNam  e :  new - nginx
          servicePor  t :  80
```

- Ingress template used to specify the proportion of requests that can be routed to the latest version of the service

Note:

In this example, the latest version of the service and the earlier version version of the service are weighted at 50% each.

```
apiVersion :  extensions / v1beta1
kind :  Ingress
metadata :
  name :  gray - release
  annotation  s :
      nginx . ingress . kubernetes . io / service - weight : |
#  Set   50 %  of   traffic   to   be   routed   to   the   new -
nginx   service .
          new - nginx :  50 ,  old - nginx :  50
spec :
  rules :
 -  host :  www . example . com
    http :
      paths :
     #  Earlier   version   of   the   service
     -  path : /
        backend :
          serviceNam  e :  old - nginx
          servicePor  t :  80
     #  Latest   version   of   the   service
     -  path : /
        backend :
          serviceNam  e :  new - nginx
          servicePor  t :  80
```

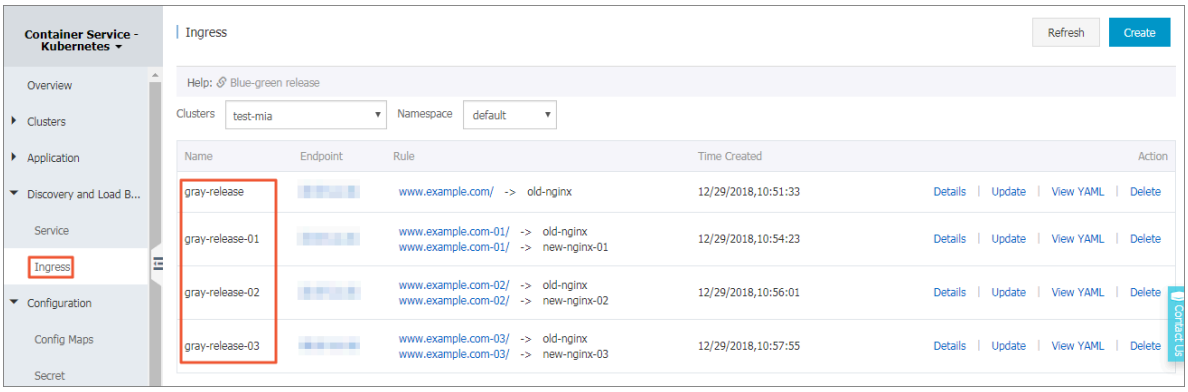
- Ingress template used to specify that only 50% of the client request traffic that meets the requirements of foo=bar will be routed the latest version of the service

```
apiVersion :  extensions / v1beta1
kind :  Ingress
metadata :
  name :  gray - release
```

```
  annotation s :
    nginx . ingress . kubernetes . io / service - match : |   #
Only if the request header of a request meets
the requiremen ts of the regular expression foo =
bar , can the request be routed to the new - nginx
  service .
        new - nginx : header (" foo ", /^ bar $/)
    nginx . ingress . kubernetes . io / service - weight : |  #
Only 50 % of the client request traffic that meets
  the requiremen ts of the preceding matching rule
can be routed to the new - nginx service .
        new - nginx : 50 , old - nginx : 50
spec :
  rules :
 - host : www . example . com
    http :
      paths :
     # Earlier version of the service
     - path : /
        backend :
          serviceNam e : old - nginx
          servicePor t : 80
     # Latest version of the service
     - path : /
        backend :
          serviceNam e : new - nginx
          servicePor t : 80
```

5. In the left-side navigation pane, choose Application > Ingress.

   You can see that the virtual host name points to old-nginx.



6. Log on to the Master node and run the following curl commands to view the Ingress access of the following settings:

   - Only the client requests that meet the requirements of the regular expression foo=bar can be routed to the latest version of the service.

```
# curl -H "Host: www.example.com" -H "foo: bar"
 http://<EXTERNAL_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
```

   - Requests of a specified proportion can be routed to the latest version of the service.

```
# curl -H "Host: www.example.com" http://<
 EXTERNAL_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
old
```

   - Only 50% of the client request traffic that meets the requirements of the regular express foo=bar can be routed to the latest version of the service.

```
# curl -H "Host: www.example.com" -H "foo: bar"
 http://<EXTERNAL_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
new
```

4.2.6 Step 3: Remove the earlier version of a service

This topic describes how to remove the earlier version of a service when the latest version of the service (which has been released through a gray release) has run without exceptions for a specified period of time.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.

- You have connected to the Kubernetes cluster by using kubectl, see *Connect to a Kubernetes cluster by using kubectl*.
- You have deployed an earlier version of the service. For more information, see *Step 1: Deploy a service*. You have also released a later version of the service through a gray release. For more information, see *Step 2: Release the latest version of a service*.

Run a command

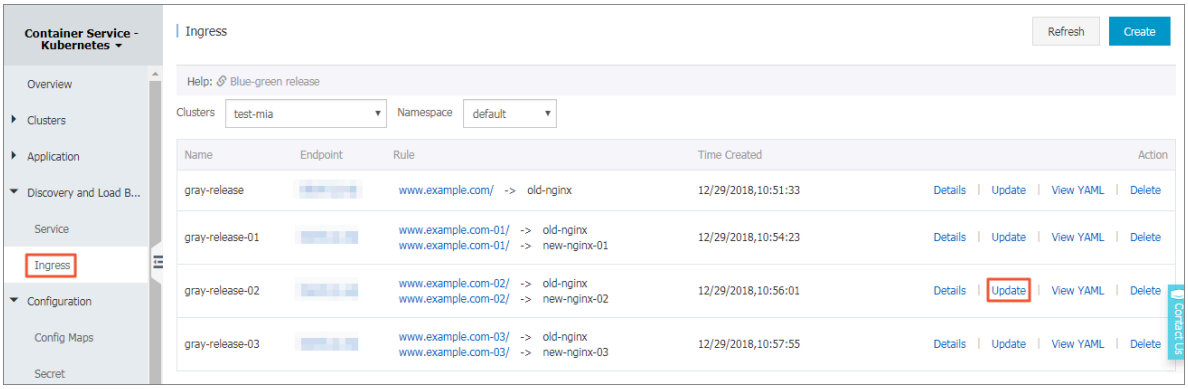
1. Run the following command to edit the YAML file deployed by *Step 2: Release the latest version of a service* to remove the earlier version of the service:

> **Note:**
> You need to remove the `annotation s` field.

```
$ kubectl get ingress gray-release-02
```

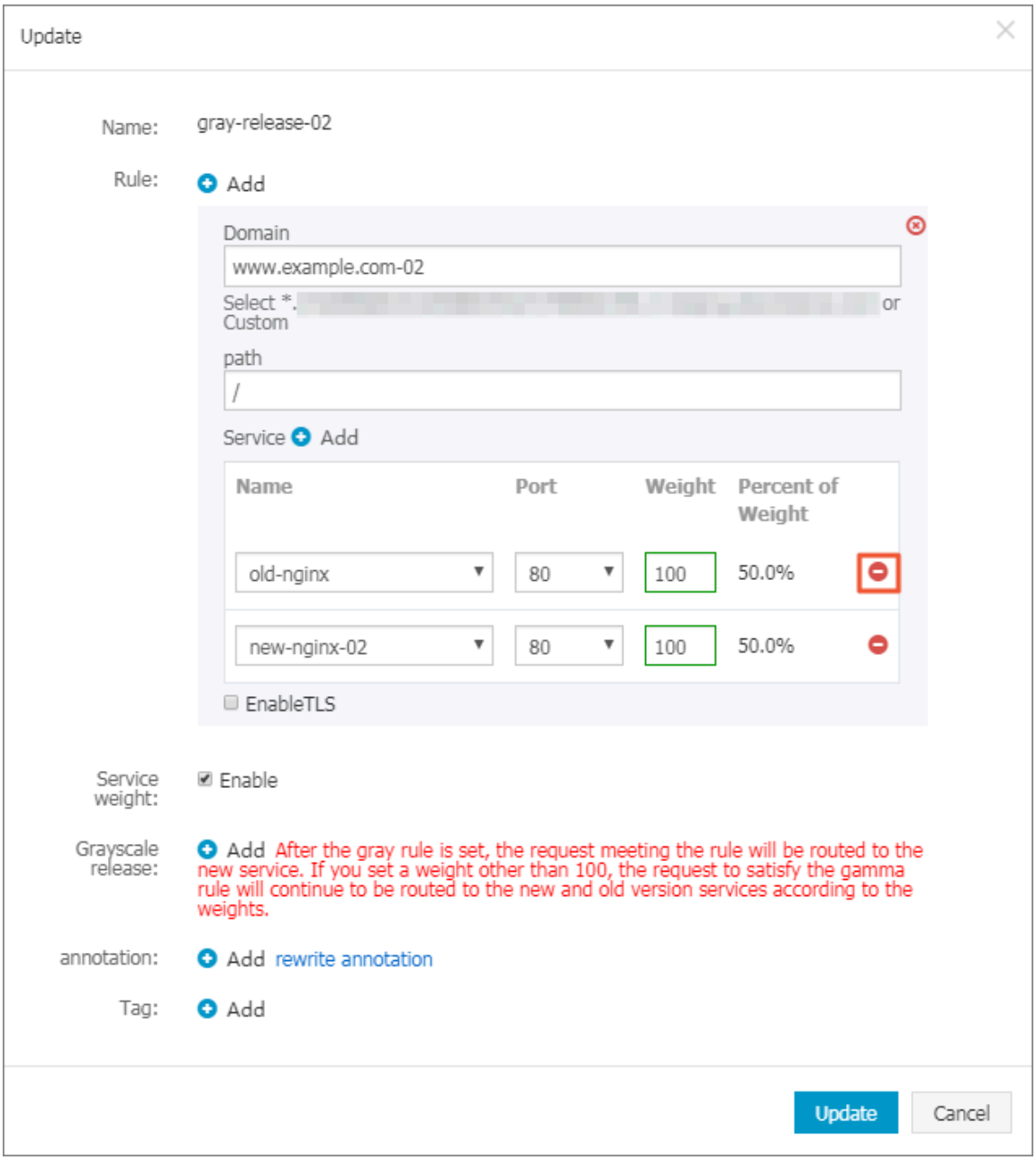
Use the Container Service console

1. Log on to the *Container Service console*.
2. In the left-side navigation pane under Kubernetes, choose Application > Ingress.
3. Select the target cluster and namespace, select the target Ingress, and click Update in the action column.



4. In the displayed dialog box, modify the Ingress as follows:

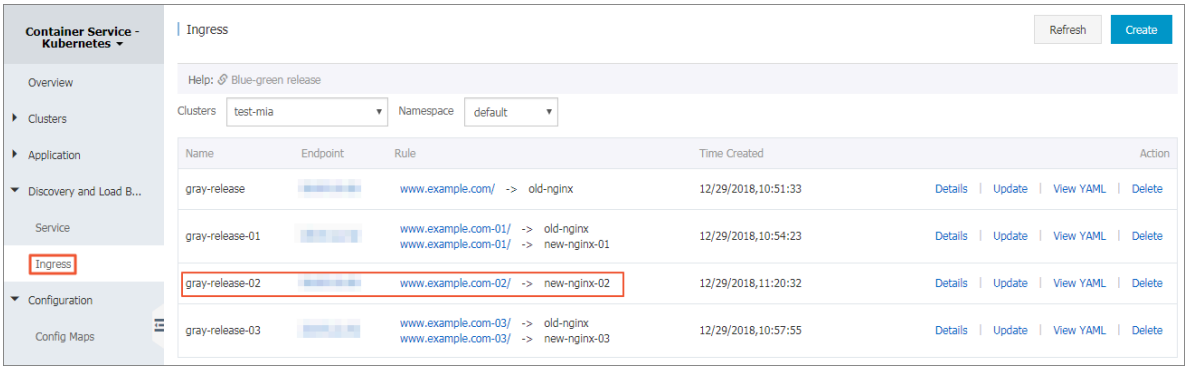
   a. In the Rule > Service area, remove the earlier version of the service rule.



   b. Click Update.

Result

1. Return to the Ingress page. Here, you can see that only one Ingress rule points to the new-nginx service.



2. Log on to the Master node and run the curl command to view the Ingress access.

```
$ curl -H "Host: www.example2.com" http://<EXTERNAL_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
```

Now, all requests are routed to the latest version of the service, which means you have completed the gray release deployment cycle. You can also remove the deployment and service of the earlier version.

4.3 Application

5 Istio

5.1 Use Istio to implement intelligent routing in Kubernetes

Alibaba Cloud Container Service for Kubernetes supports one-click deployment of Istio and multiple functions expanded on Istio. This topic describes how to implement intelligent routing through Istio. For information about Istio official documents, see *Intelligent Routing*.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have deployed Istio. For more information, see *Deploy Istio*.

   Note:
  Istio used in this topic is V 1.0.2.
- You have a local Linux environment in which you have configured the kubectl tool and used the tool to connect to the cluster. For more information, see *Connect to a Kubernetes cluster by using kubectl*.
- You have downloaded the project code of an Istio version and run the relevant commands in the Istio file directory. See *https://github.com/istio/istio/releases*.

Install the Istio official sample application

Install the Istio official sample application, Bookinfo. For more information, see *https://istio.io/docs/guides/bookinfo*.

Quickly deploy the Bookinfo sample application

1. Label the `default` namespace with the `istio-injection=enabled` tag.

    Note:

Kubernetes clusters running on Alibaba Cloud Container Service support one-click deployment of Istio and automatic sidecar injection.

```
$ kubectl label namespace default istio - injection =
 enabled
```

2. Run the following kubectl command to deploy the Bookinfo sample application:

```
$ kubectl apply - f samples / bookinfo / platform / kube /
 bookinfo . yaml
```

The preceding command starts all four microservices. All three `Reviews` service versions (v1, v2, and v3) are also started.

3. Run the following command to vefiry that all services and pods are properly defined and started:

```
$ kubectl get svc , pods
```

4. You need to access the application from the outside of your Kubernetes cluster, for example, a browser. You need to create an *Istio Gateway*. Define the ingress gateway for the application.

```
$ kubectl apply - f samples / bookinfo / networking / bookinfo
 - gateway . yaml
```

Run the following command to verify that the gateway has been created:

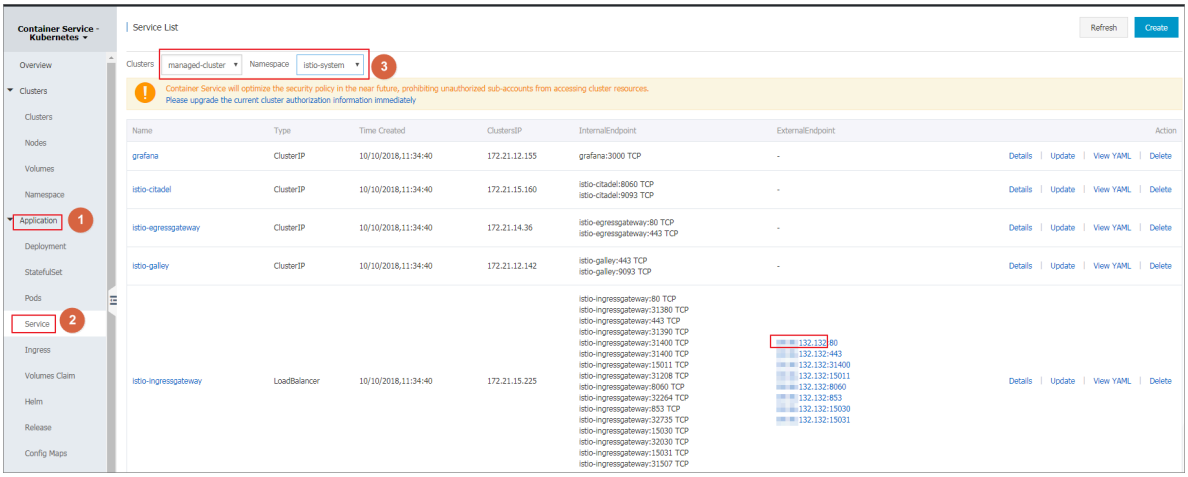
```
$ kubectl get gateway
 NAME AGE
```

```
bookinfo - gateway      32s
```

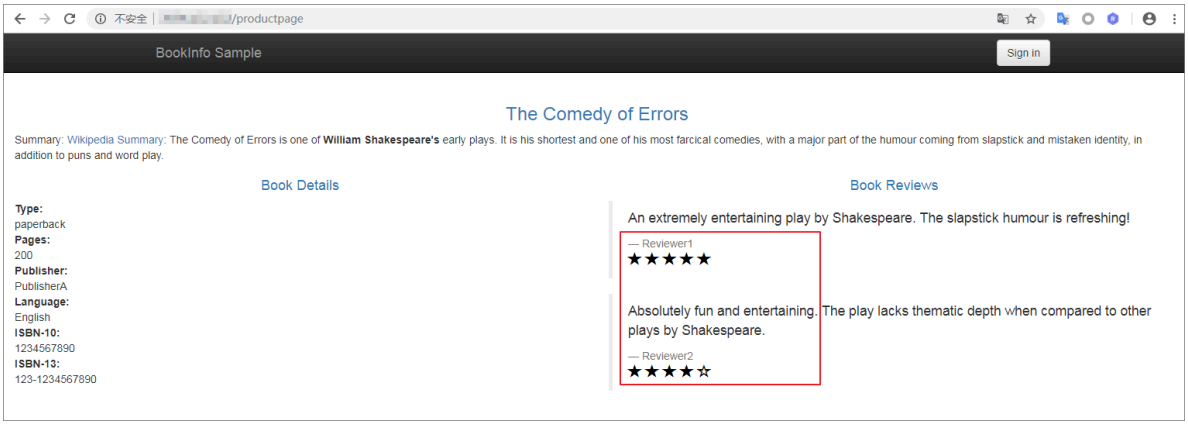
5. Run the following command to check the IP address of `istio - ingressgat eway` .

```
$ kubectl get svc istio - ingressgat eway - n istio - system
```

You can also log on to the Container Service console to view the IP address of `istio - ingressgat eway` . Specifically, choose Application > Service in the left-side navigation pane, select the target cluster and the Istio-system namespace.



6. Access the BookInfo home page. The access address is `http ://{ EXTERNAL - IP }/ productpag e` .



If you refresh the browser, different versions of the reviews are displayed on the productpage in a round-robin manner (starting with a red star, to a black star, to no star). This indicates that Istio is currently not being used to control the version routing.

Set a route for requests

You need to set a default route because three Reviews service versions are deployed for the BookInfo sample application. Otherwise, if you access the application multiple times, you will notice that sometimes the book review output contains star ratings and other times it does not. This is because you have not set a default route for the rating service versions, and Istio then randomly routes requests to all available versions in a round robin fashion.

You need to define available versions in the destination routing rule before using Istio to control the route to the service versions of the BookInfo application.

Create the default destination routing rule for the BookInfo service.

- If you do not want to enable bidirectional TLS, run the following command:

```
$ kubectl apply -f samples/bookinfo/networking/
destinationrule-all.yaml
```

- If you want to enable bidirectional TLS, run the following command:

```
$ kubectl apply -f samples/bookinfo/networking/
destinationrule-all-mtls.yaml
```

Wait for a few seconds until the destination routing rule takes effect. Run the following command to view the destination routing rule:

```
$ kubectl get destinationrules -o yaml
```

Set the default version of all microservices to v1

Run the following command to set the default version of all microservices to v1:

```
$ kubectl apply -f samples/bookinfo/networking/virtual-
service-all-v1.yaml
```

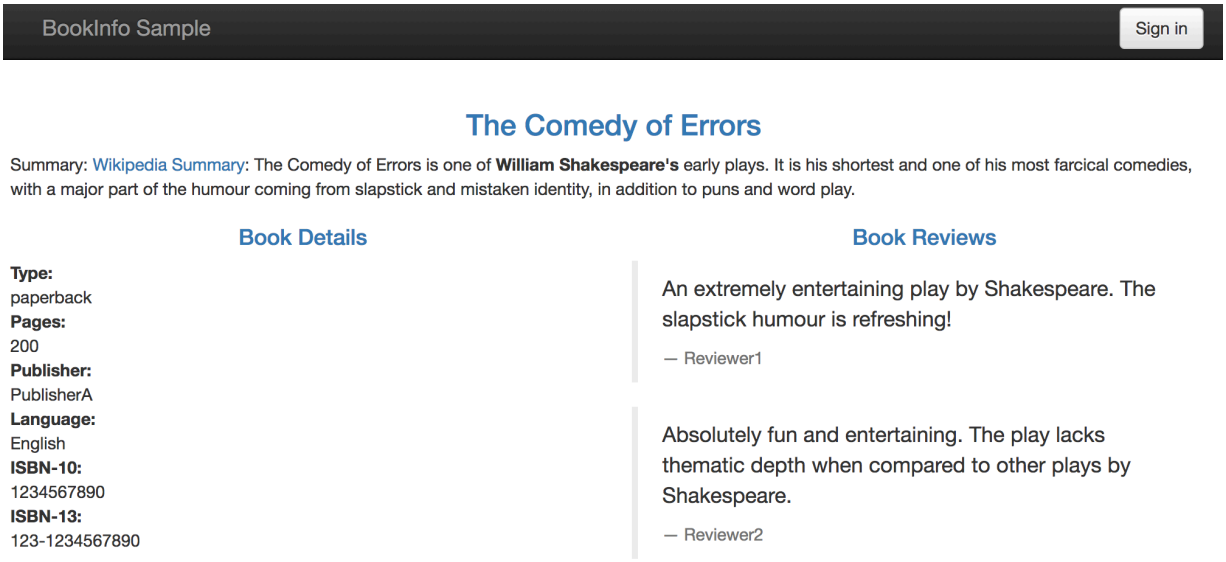
Run the following command to display all the created routing rules:

```
kubectl get virtualservices -o yaml
```

It takes a period of time for the routing rule to be synchronized to all pods because the routing rule is distributed to the proxy in an asynchronized manner. Therefore, we recommend that you wait for a few seconds before accessing the application.

Open the URL of the Bookinfo application in your browser: `http://{EXTERNAL-IP}/productpage`.

On the product page of the BookInfo application, the displayed content does not contain the reviews with starts. This is because the reviews:v1 service does not access the ratings service.



Route the requests from a specific user to reviews:v2

Run the following command to route requests from the test user named jason to reviews:v2 to enable the ratings service:

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-reviews-test-v2.yaml
```

Run the following command to check whether routing rules are created:

```
$ kubectl get virtualservice reviews -o yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
```

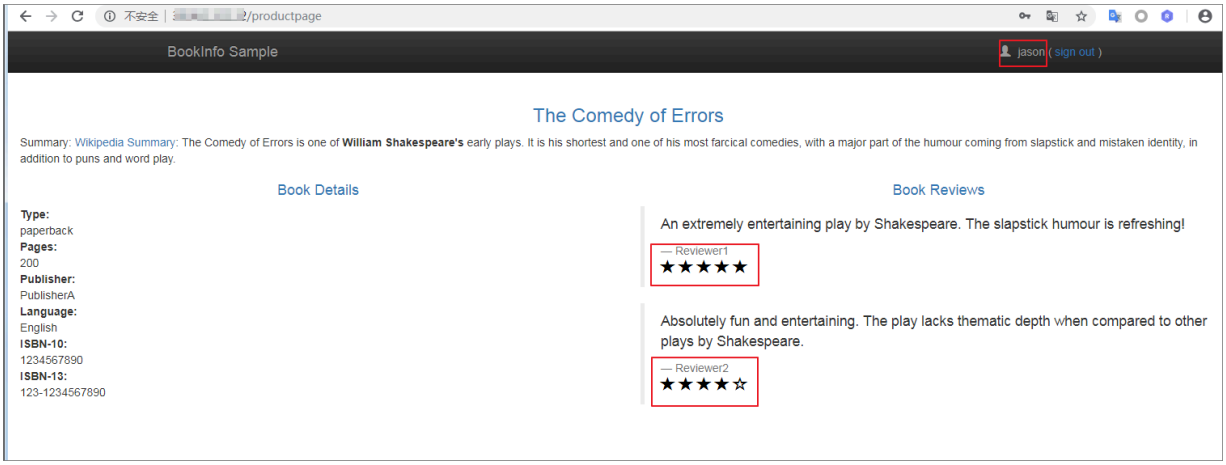
```
        subset :  v1
```

After you confirm that the routing rule is created, open the URL of the BookInfo application in your browser: `http ://{ EXTERNAL - IP }/ productpag e`.

Log on to the product page as the jason user to verify that the rating information is displayed under each review record.

Note:

Both the logon account name and password for are `jason ..`



Note:

In this example, two request routing rules have been changed. Firstly, all requests are routed to the v1 version of the Reviews service provided by the BookInfo application. Then, a new routing rule is set to route specific requests to the v2 version of the Reviews service according to the header of a request (for example, the user cookie).

Inject faults

To test the resiliency of the microservices application, namely, BookInfo, inject a 7-second delay between the `reviews : v2` microservices and the `ratings` microservices for the jason user. Note that the reviews:v2 service has a 10-second hard-coded connection timeout for calls to the ratings service. Therefore, you can still expect the end-to-end flow to continue without any errors even you have set the 7-second delay .

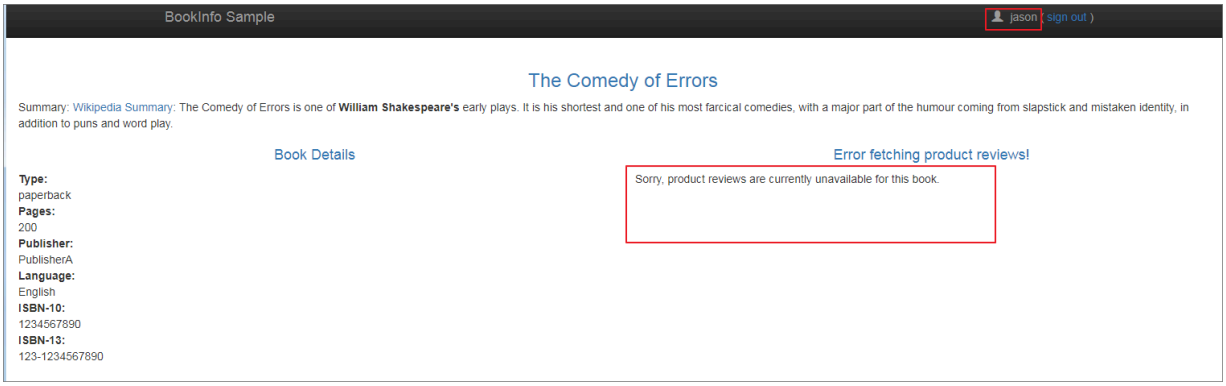
Inject an HTTP delay fault

Create a fault injection rule to delay traffic coming from the jason user.

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-delay.yaml
```

After you confirm that the rule is created, open the URL of the BookInfo application in your browser: `http://{EXTERNAL-IP}/productpage`.

Log on to the productpage as the jason user to view the following.



Note:

The reviews service fails because the timeout between the productpage and reviews services is shorter than the timeout between the reviews and ratings services, that is, (3 seconds + 1 retry = 6 seconds) is shorter than 10 seconds. Bugs like this can occur in typical enterprise applications where different teams develop different microservices independently. Istio's fault injection rules help you identify such anomalies without impacting end users.

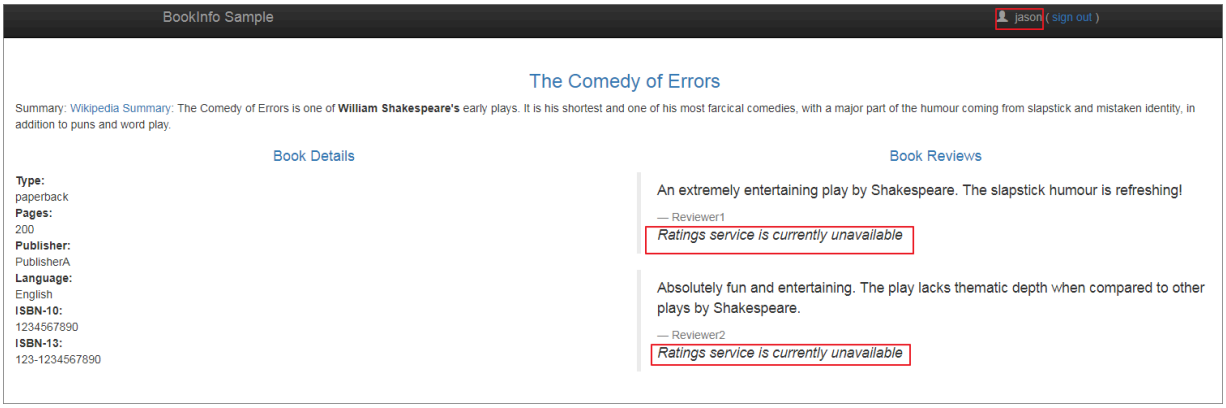
Inject an HTTP abort fault

Create a fault injection rule to send an HTTP abort

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-abort.yaml
```

After you confirm that the rule is created, open the URL of the BookInfo application in your browser: `http://{EXTERNAL-IP}/productpage`.

Log on to the productpage as the jason user to view the following.



Migrate traffic

In addition to the content-based routing rule, Istio also supports the weight-based routing rule.

Run the following command to route all traffic to the v1 version of all microservices.

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-all-v1.yaml
```

Run the following command to route 50% of traffic from the reviews v1 service to the reviews v3 service:

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-reviews-50-v3.yaml
```

Refresh the productpage for multiple times in the browser. You have a 50% probability to see the review content marked with red stars on the page.

Note:

Note that this method is completely different from using the deployment feature of the container orchestration platform for version migration. The container orchestration platform uses the instance scaling method to manage the traffic. With istio, two versions of the reviews service can expand and shrink capacity independently, without affecting the distribution of traffic between the two versions of services.

Assuming you decide that the `reviews : v3` microservice is stable, you can route 100% of the traffic to `reviews : v3` to implement a gray release by running the following command:

```
$ kubectl replace -f samples / bookinfo / networking / virtual
 - service - reviews - v3 . yaml
```

Conclusion

You can use Alibaba Cloud Container Service for Kubernetes to quickly build the open platform, that is, Istio, to connect, manage, and secure microservices, and to introduce and configure multiple relevant services for applications. This topic uses a sample application from Istio to detail how to use Istio functions such as traffic rouging, fault injection, and traffic migrating. We recommend that you use Alibaba Cloud Container Service for Kubernetes to quickly build Istio, an open management platform for microservices, and integrate Istio with the microservice development of your project.

5.2 Implement Istio distributed tracking in Kubernetes

Background

Microservice is a focus in the current era. More and more IT enterprises begin to embrace the microservices. The microservice architecture splits a complex system into several small services and each service can be developed, deployed, and scaled independently. As a heaven-made match, the microservice architecture and containers (Docker and Kubernetes) further simplify the microservice delivery and strengthen the flexibility and robustness of the entire system.

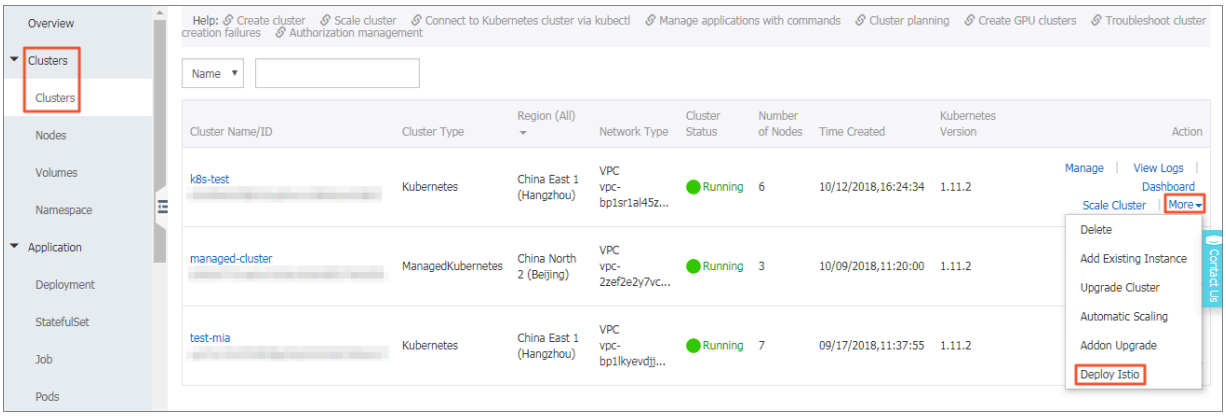
When monolithic applications are transformed to microservices, the distributed application architecture composed of a large number of microservices also increases the complexity of operation & maintenance, debugging, and security management . As microservices grow in scale and complexity, developers must be faced with complex challenges such as service discovery, Server Load Balancer, failure recovery , indicator collection, monitoring, A/B testing, throttling, access control, and end-to-end authentication, which are difficult to resolve.

In May 2017, Google, IBM, and Lyft published the open-source service network architecture Istio, which provides the connection, management, monitoring, and security protection of microservices. Istio provides an infrastructure layer for

services to communicate with each other, decouples the issues such as version management, security protection, failover, monitoring, and telemetry in applicatio n logics and service access. Being unrelated to codes, Istio attracts enterprises to transform to microservices, which will make the microservice ecology develop fast.

Architecture principle of Istio

In Kubernetes, a pod is a collection of close-coupled containers, and these containers share the same network namespace. With the extension mechanism of Initializer in Kubernetes, an Envoy container is automatically created and started for each business pod, without modifying the deployment description of the business pod. The Envoy takes over the inbound and outbound traffic of business containers in the same pod. Therefore, the microservice governance functions, including the traffic management, microservice tracking, security authentication, access control, and strategy implementation, are realized by operating on the Envoy.



An Istio service mesh is logically split into a data plane and a control plane.

- The data plane is composed of a collection of intelligent proxies (Envoys) deployed as sidecars that mediate and control all network communication between microservices.
- The control plane is used to manage and configure the proxies to route traffic, and enforce polices at the runtime.

An Istio is mainly composed of the following components:

- Envoy: The Envoy is used to mediate all the inbound and outbound traffic for all the services in the service mesh. Functions such as dynamic service discovery, Server Load Balancer, fault injection, and traffic management are supported. The Envoy is deployed as a sidecar to the pods of related services.

- Pilot: The Pilot is used to collect and verify the configurations and distribute the configurations to all kinds of Istio components.
- Mixer: The Mixer is used to enforce the access control and usage policies in the service mesh, and collect telemetry data from Envoy proxies and other services.
- Istio-Auth: Istio-Auth provides strong service-to-service and end user authentication.

For more information about Istio, see the *Istio official document*.

Install Istio

Use an Alibaba Cloud Container Service Kubernetes cluster as an example.

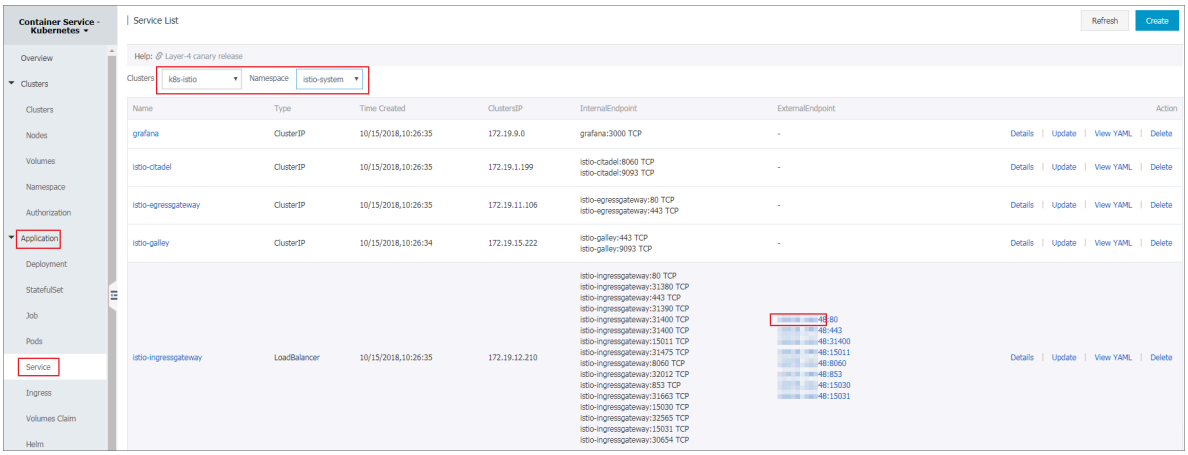
Alibaba Cloud Container Service has enabled the Initializers plug-in by default for Kubernetes clusters if the cluster version is later than 1.8. No other configurations are needed.

Note:
After you deploy the Istio, a sidecar is injected to each pod to take over the service communication. Therefore, we recommend that you verify this in the independent test environment.

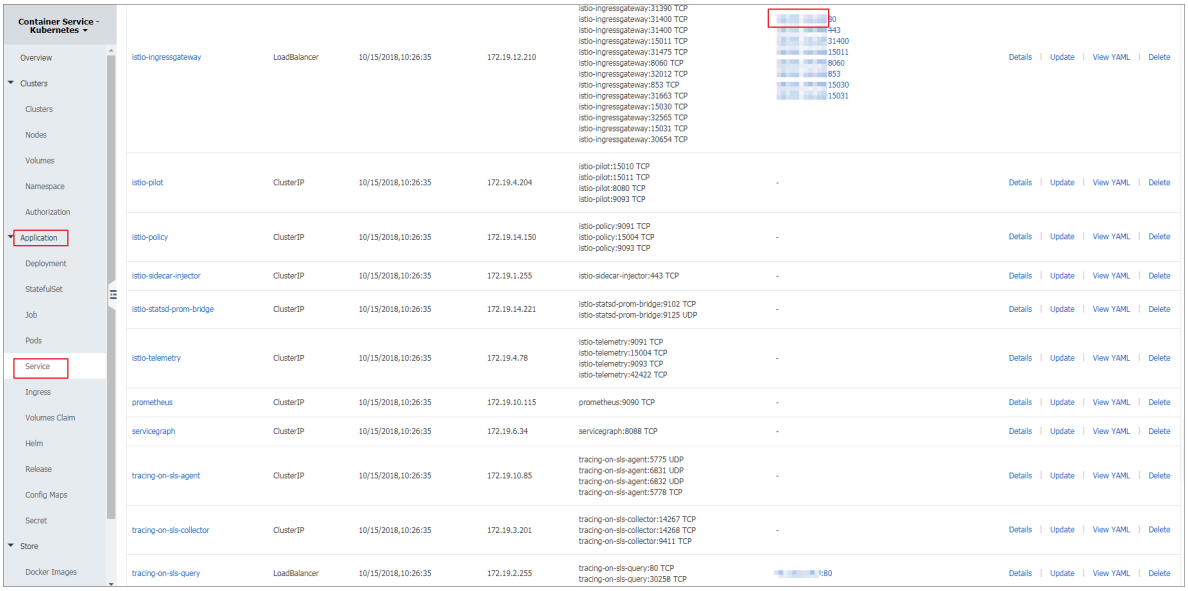
Create a Kubernetes cluster

1. Log on to the *Container Service console*.
2. Under Kubernetes, click Clusters in the left-side navigation pane, and click Create Kubernetes cluster in the upper-right corner.
3. Configure the parameters to create a cluster. For how to create a Kubernetes cluster, see *Create a Kubernetes cluster*.

4. After the cluster is created, click Manage at the right of the cluster when the cluster status is changed to Running.



5. On the cluster Basic Information page, you can configure the corresponding connection information based on the page information. You can connect to the cluster either by using *Connect to a Kubernetes cluster by using kubectl* or *Access Kubernetes clusters by using SSH*.



Deploy Istio release version

Log on to the master node and run the following command to get the latest Istio installation package.

```
curl  - L   https :// git . io / getLatestI  stio  |  sh  -
```

Run the following command:

```
cd   istio - 0 . 4 . 0                              ## Change   the
working   directory   to   Istio
```

```
export PATH =$ PWD / bin :$ PATH ## Add the
istioctl client to PATH environmen t variable
```

Run the following command to deploy Istio.

```
kubectl apply - f install / kubernetes / istio . yaml
  ## Deploy Istio system components
kubectl apply - f install / kubernetes / istio - initialize r .
yaml ## Deploy Istio initialize r plug - in
```

After the deployment, run the following command to verify if the Istio components are successfully deployed.

```
$ kubectl get svc , pod - n istio - systemNAME TYPE
CLUSTER - IP EXTERNAL - IP PORT ( S ) AGEsvc / istio - ingress
  LoadBalanc er 172 . 21 . 10 . 18 101 . 37 . 113 . 231 80 :
30511 / TCP , 443 : 31945 / TCP 1msvc / istio - mixer ClusterIP
  172 . 21 . 14 . 221 9091 / TCP , 15004 / TCP , 9093 / TCP , 9094
/ TCP , 9102 / TCP , 9125 / UDP , 42422 / TCP 1msvc / istio - pilot
  ClusterIP 172 . 21 . 4 . 20 15003 / TCP , 443 / TCP 1mNAME
READY STATUS RESTARTS AGEpo / istio - ca - 55b954ff7 - crsjq
1 / 1 Running 0 1mpo / istio - ingress - 948b746cb - 4t24c 1
/ 1 Running 0 1mpo / istio - initialize r - 6c84859cd - 8mvfj
  1 / 1 Running 0 1mpo / istio - mixer - 59cc756b48 - tkx6c 3
/ 3 Running 0 1mpo / istio - pilot - 55bb7f5d9d - wc5xh 2 / 2
  Running 0 1m
```

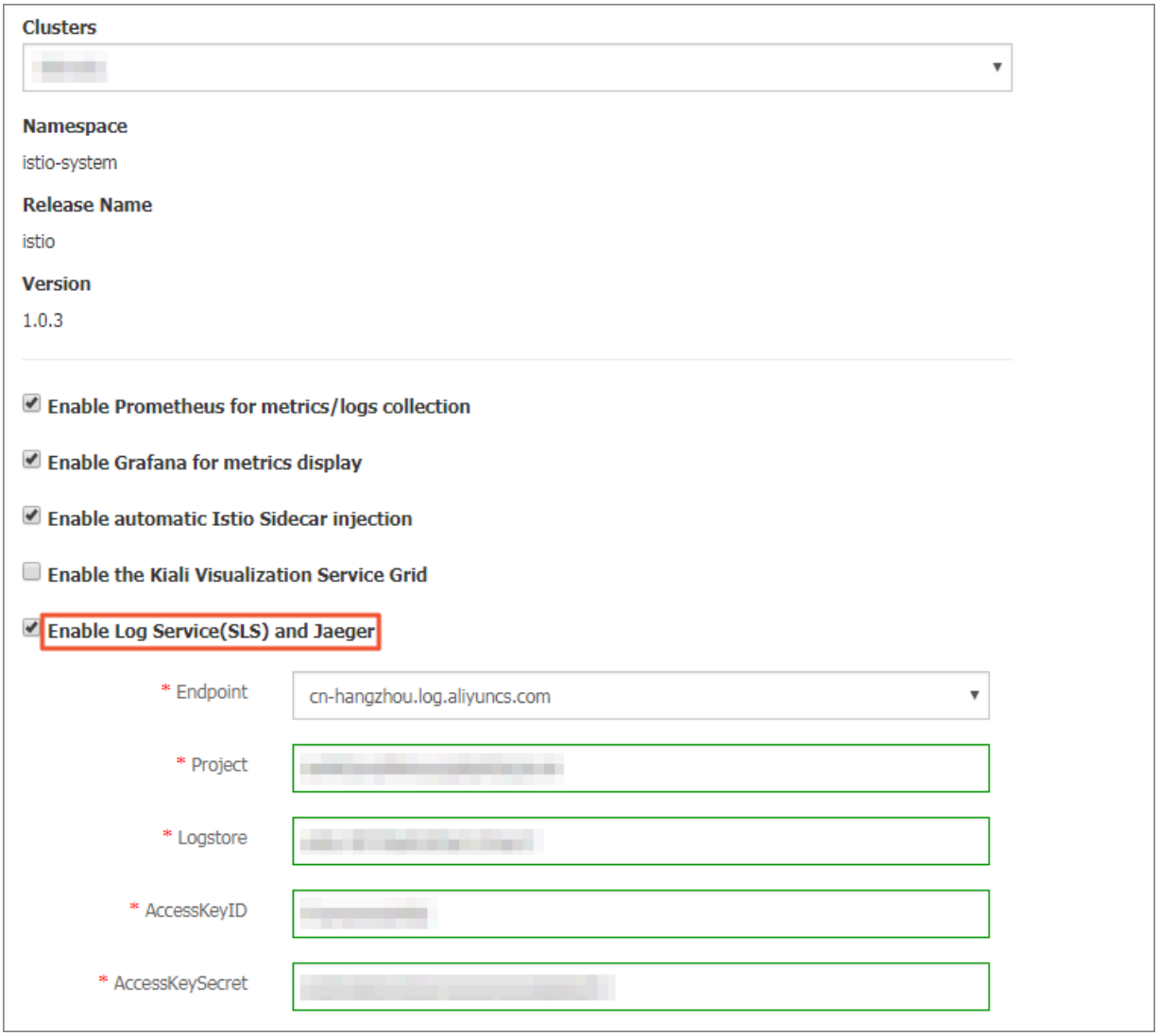
After all the pods are in the running status, the Istio deployment is finished.

Istio distributed service tracking case

Deploy and test the application BookInfo

BookInfo is an application similar to an online bookstore, which is composed of several independent microservices compiled by different languages. The application BookInfo is deployed in the container mode and does not have any dependencies on Istio. All the microservices are packaged together with an Envoy sidecar. The Envoy sidecar intercepts the inbound and outbound call requests of services to demonstrate the distributed tracking function of Istio service mesh.

For more information about BookInfo, see *Bookinfo guide*.



Run the following command to deploy and test the application Bookinfo.

```
kubectl apply -f samples/bookinfo/kube/bookinfo.yaml
```

In the Alibaba Cloud Kubernetes cluster environment, every cluster has been configured with the Server Load Balancer and Ingress. Run the following command to obtain the IP address of Ingress.

```
$ kubectl get ingress -o wide
NAME HOSTS ADDRESS PORTS AGE
```

```
gateway    *          101 . 37 . xxx . xxx     80          2m
```

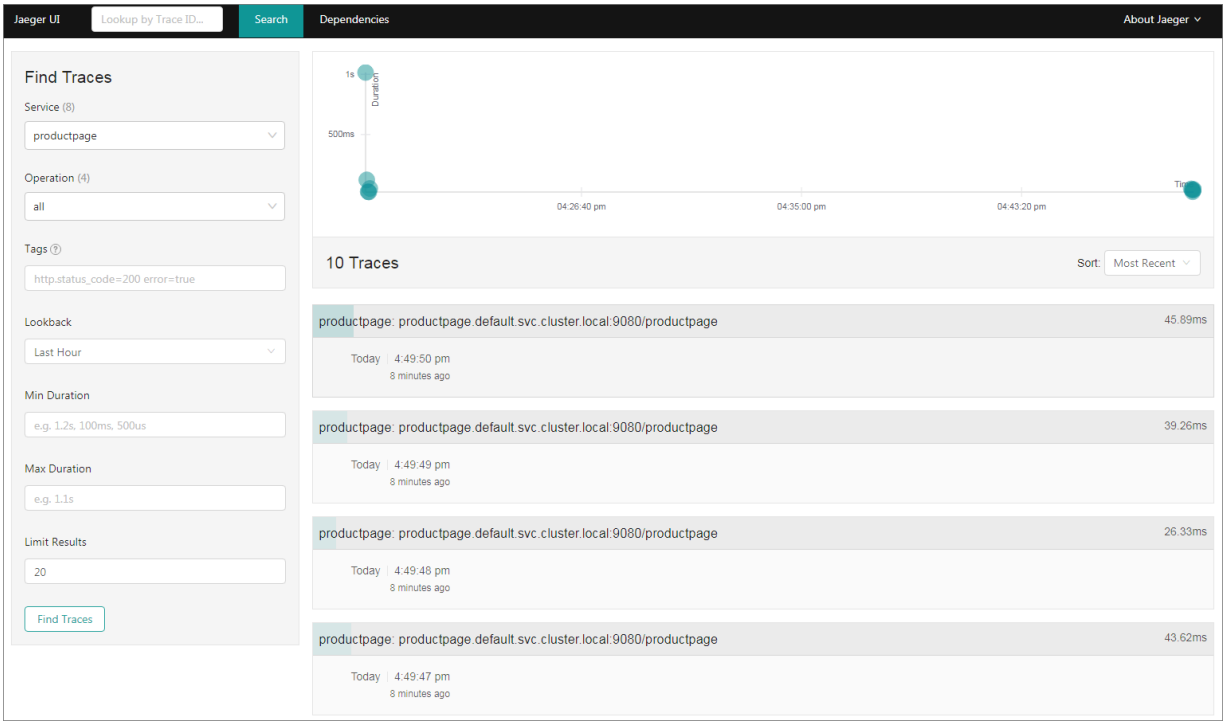
If the preceding command cannot obtain the external IP address, run the following command to obtain the corresponding address.

```
export   GATEWAY_UR  L =$( kubectl   get   ingress  - o   wide  - o
jsonpath ={. items [ 0 ]. status . loadBalanc  er . ingress [ 0 ]. ip
})
```

The application is successfully deployed if the following command returns 200.

```
curl  - o  / dev / null  - s  - w  "%{ http_code }\ n "  http ://${
GATEWAY_UR  L }/ productpag  e
```

You can open `http ://${ GATEWAY_UR  L }/ productpag  e` in the browser to access the application. GATEWAY_URL is the IP address of Ingress.



Deploy Jaeger tracking system

Distributed tracking system helps you observe the call chains between services and is useful when diagnosing performance issues and analyzing system failures.

Istio ecology supports different distributed tracking systems, including *Zipkin* and *Jaeger*. Use the Jaeger as an example.

Istio version 0.4 supports Jaeger. The test method is as follows.

```
kubectl apply -n istio-system -f https://raw.githubusercontent.com/jaegertracing/jaeger-kubernetes/master/all-in-one/jaeger-all-in-one-template.yml
```

After the deployment is finished, if you connect to the Kubernetes cluster by using kubectl, run the following command to access the Jaeger control panel by using port mapping and open `http://localhost:16686` in the browser.

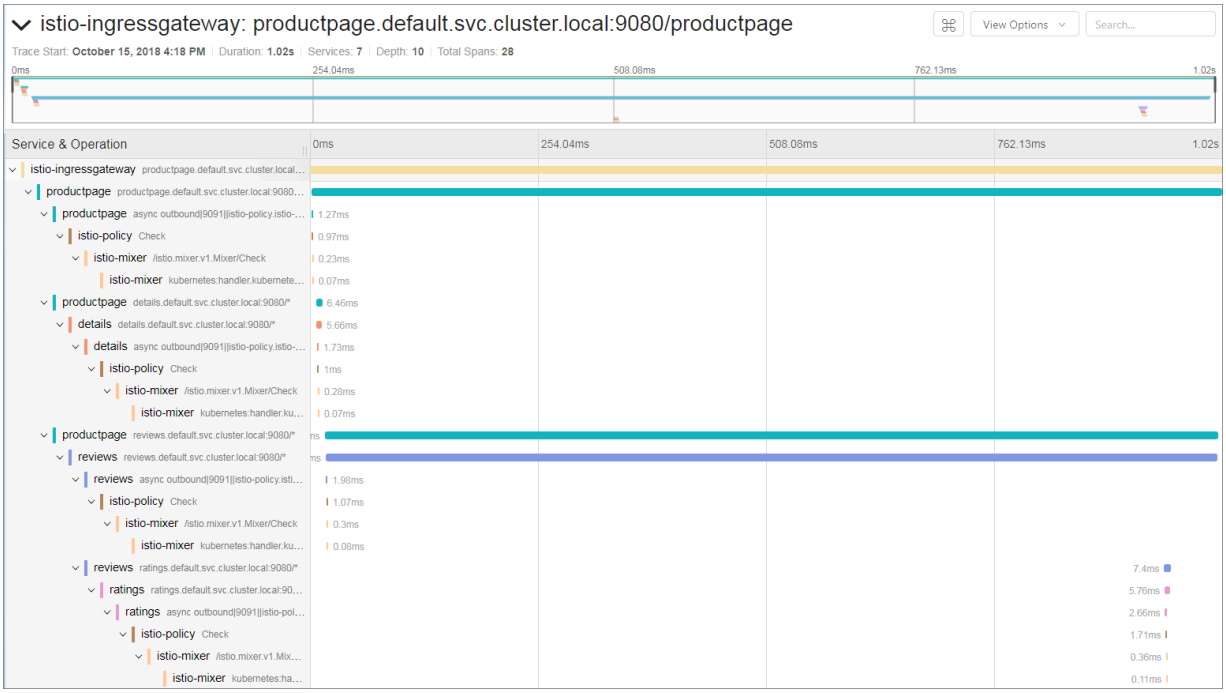
```
kubectl port-forward -n istio-system $(kubectl get pod -n istio-system -l app=jaeger -o jsonpath='{.items[0].metadata.name}') 16686:16686 &
```

If you connect to the Alibaba Cloud Kubernetes cluster by using SSH, run the following command to check the external access address of jaeger-query service.

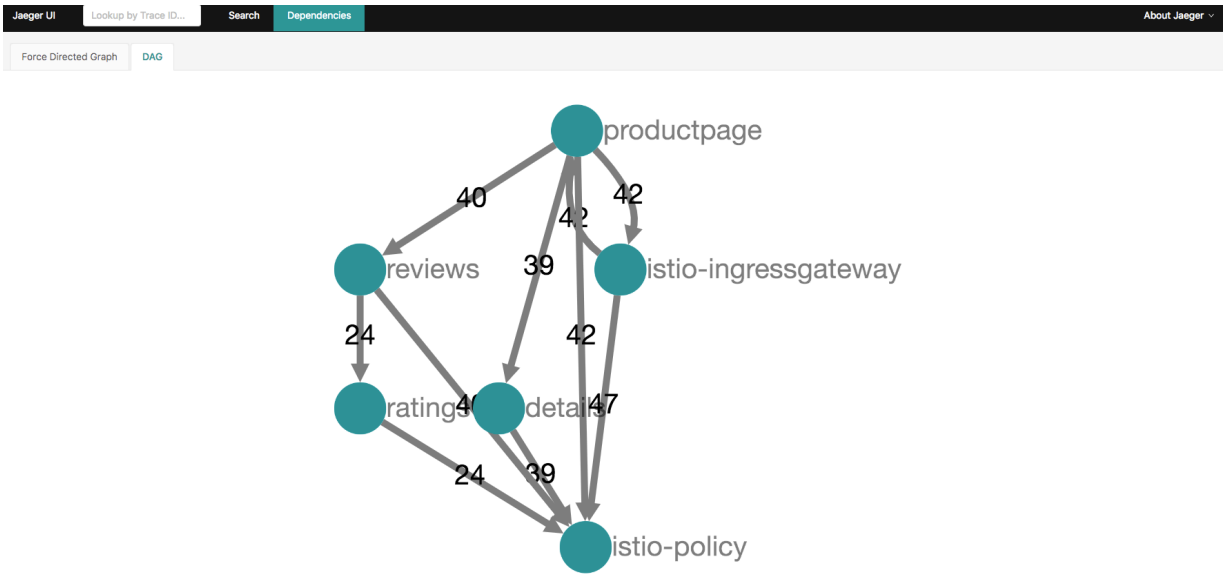
```
$ kubectl get svc -n istio-system
NAME                 TYPE           CLUSTER-IP       EXTERNAL-IP       PORT(S)
             AGE
jaeger-agent         ClusterIP      None             <none>
       5775/UDP,6831/UDP,6832/UDP
                 1h
jaeger-collector     ClusterIP      172.21.10.187    <none>
            14267/TCP,14268/TCP,9411/TCP
                  1h
jaeger-query         LoadBalancer   172.21.10.197    114.55.82.11
       80:31960/TCP      ## The external access address is 114.55.82.11:80.
zipkin               ClusterIP      None             <none>
     9411/TCP
```

Record the external access IP address and port of jaeger-query and then open the application in the browser.

By accessing the application BookInfo for multiple times and generating the call chain information, we can view the call chain information of services clearly.



Click a specific Trace to view the details.



You can also view DAG.

Implementation principle of Istio distributed tracking

The kernel of Istio service mesh is the Envoy, which is a high-performance and open-source Layer-7 proxy and communication bus. In Istio, each microservice is injected with an Envoy sidecar and this instance is responsible for processing all the inbound and outbound network traffic. Therefore, each Envoy sidecar can monitor all the API calls between services, record the time required by each service call, and record whether each service call is successful or not.

Whenever a microservice initiates an external call, the client Envoy will create a new span. A span represents the complete interaction process between a collection of microservices, starting from a caller (client) sending a request to receiving the response from the server.

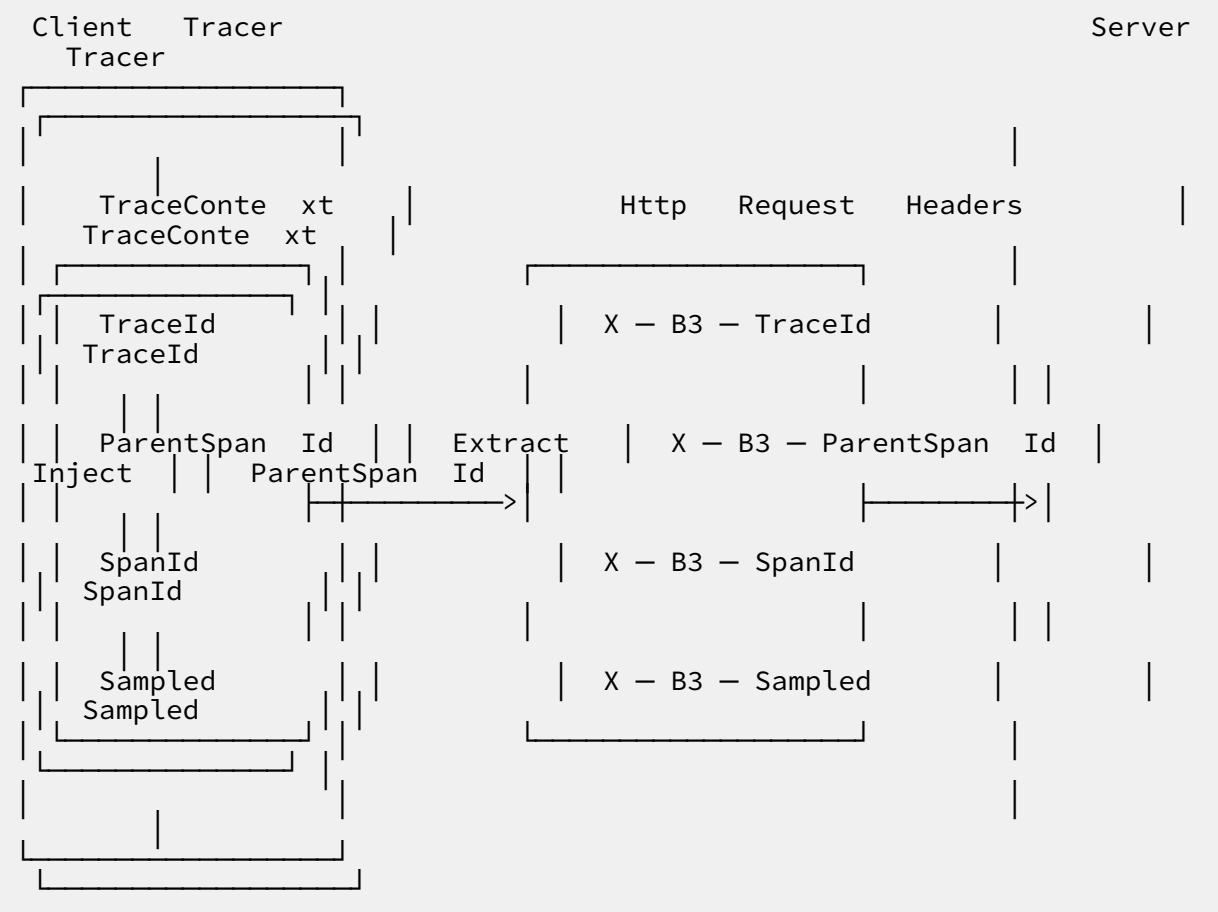
In the service interaction process, clients record the request start time and response receipt time, and the Envoy on the server records the request receipt time and response return time.

Each Envoy distributes their own span view information to the distributed tracking system. When a microservice processes requests, other microservices may need

to be called, which causes the creation of a causally related span and then forms the complete trace. Then, an application must be used to collect and forward the following Headers from the request message:

- `x-request-id`
- `x-b3-traceid`
- `x-b3-spanid`
- `x-b3-parentspan id`
- `x-b3-sampled`
- `x-b3-flags`
- `x-ot-span-context`

Envoys in the communication links can intercept, process, and forward the corresponding Headers.



For specific codes, see the Istio document *https://istio.io/docs/tasks/telemetry/distributed-tracing.html*.

Conclusion

Istio is accelerating the application and popularization of service mesh by using the good expansion mechanism and strong ecology. In addition to those mentioned in the preceding sections, Weave Scope, Istio Dashboard, and Istio-Analytics projects provide abundant call link visualization and analysis capabilities.

5.3 Integrate Istio with Alibaba Cloud Log Service on Kubernetes

With Alibaba Cloud Container Service for Kubernetes, you can integrate Istio with Log Service. This topic uses an official example to describe how to integrate Istio with the distributed tracing system based on Alibaba Cloud Log Service.

Prerequisites

- You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.
- You have a local Linux environment in which you have configured the kubectl tool and connected to the cluster with kubectl. For more information, see *Connect to a Kubernetes cluster by using kubectl*.
- You have downloaded the project codes of an Istio version and run the relevant commands in the Istio file directory. For more information, see *https://github.com/istio/istio/releases*.
- You have deployed the test application, BookInfo. For more information, see*Install the Istio official sample application* or *Bookinfo Application* .

Introduction to OpenTracing

The OpenTracing specification has been created to address API incompatibility between different distributed tracing systems. OpenTracing is a lightweight and standardized layer that is located between applications/class libraries and tracing or log analysis programs. OpenTracing has joined the Cloud Native Computing Foundation (CNCF) and provided unified concepts to provide data standards for global distributed tracing systems. It provides platform-independent and vendor-independent APIs. With OpenTracing, developers can easily add or replace tracing systems.

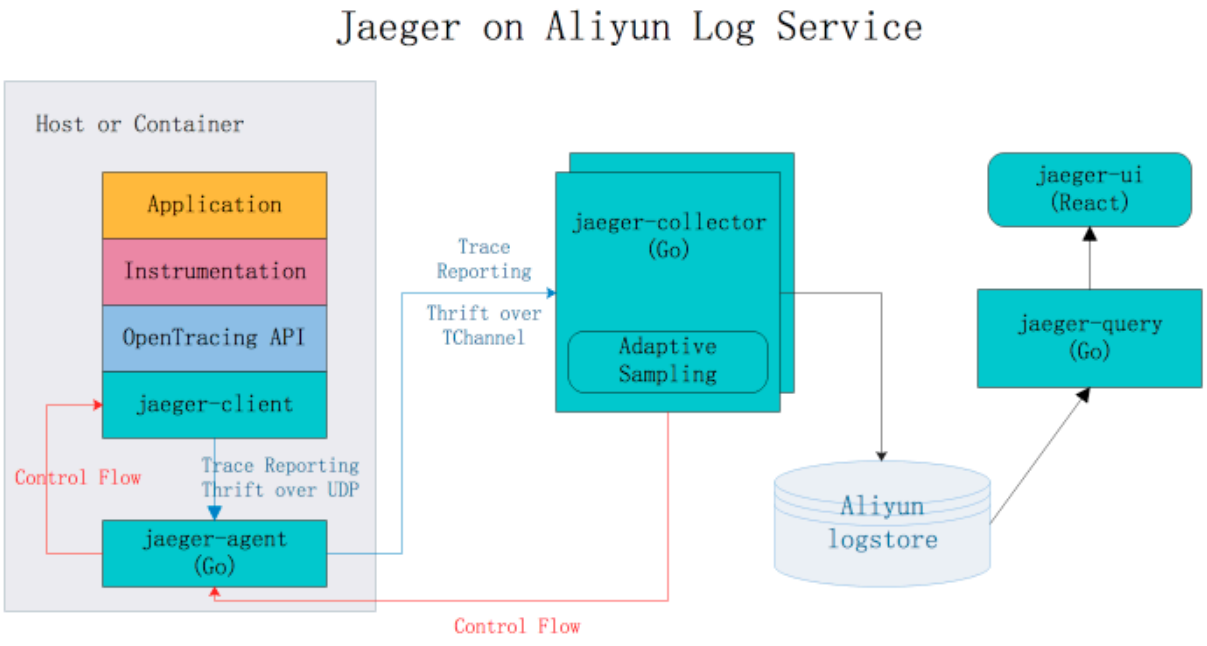
Jaeger is an open source distributed tracing system released by CNCF, and is compatible with OpenTracing APIs.

Alibaba Cloud Log Service and the distributed tracing system Jaeger

Log Service (LOG for short, formerly called SLS) is a one-stop service for real-time data. It has been used extensively by Alibaba Group in big data scenarios. Log service helps you collect, consume, ship, query, and analyze massive log data with improved processing capabilities.

Jaeger is an open source distributed tracing system released by Uber for microservice scenarios. It is used to analyze the calling process of multiple servers and display the service calling tracks with graphs. It is a powerful tool for diagnosing performance and analyzing system failures.

Jaeger on Aliyun Log Service is a distributed tracing system developed on Jaeger. With this system, you can store the collected tracing data to Alibaba Cloud Log Service permanently, and query and present the data through the native interfaces of Jaeger.



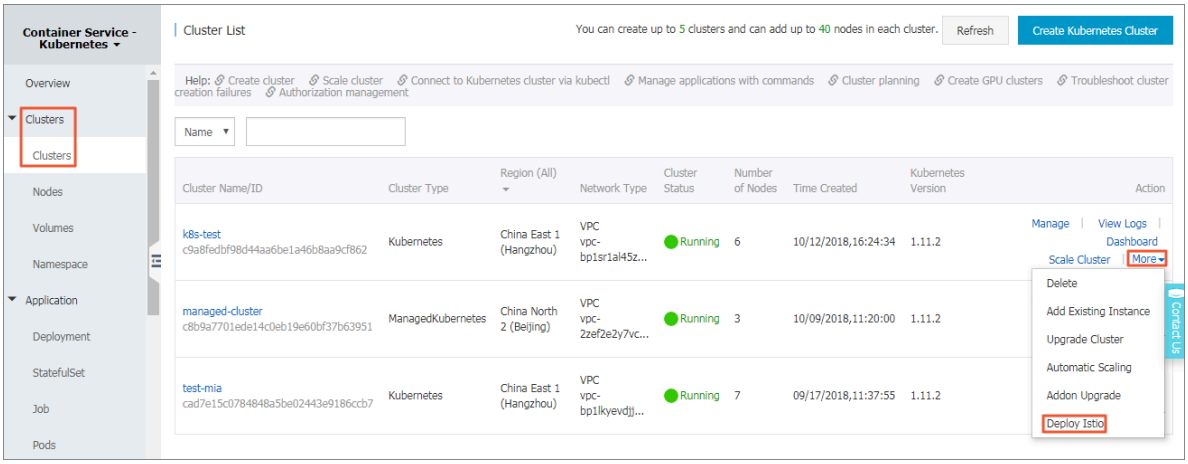
Component	Description
Jaeger client	The Jaeger client provides SDKs that conform to the OpenTracing standard for different languages. Applications write data to the client through APIs. The client library delivers the traces to the Jaeger agent according to the sampling policy specified by the applications. The tracing data is serialized by Thrift and transmitted through UDP.
Jaeger agent	The Jaeger agent is a network daemon that listens to spans sent over UDP and sends data in batches to the Jaeger collector. The Jaeger agent is designed as an infrastructure component that is deployed to all hosts. The Jaeger agent decouples the Jaeger client and Jaeger collector, and makes the collectors invisible to the client. Collectors are then no longer routed or discovered by the client.
Jaeger collector	The Jaeger collector receives traces from the Jaeger agent and writes the traces into the backend storage system . The backend storage is a pluggable component. Jaeger on Aliyun Log Service supports Alibaba Cloud Log Service.
Alibaba Cloud Log Service	The Jaeger collector stores the received spans to Log Service permanently. Query retrieves data from Log Service.
Query & UI	Query is a service that receives query requests, retrieves traces from the backend storage system, and hosts a UI to display traces.

Step 1: Deploy Istio with a single click through the Web interface

1. Log on to the *Container Service console*.

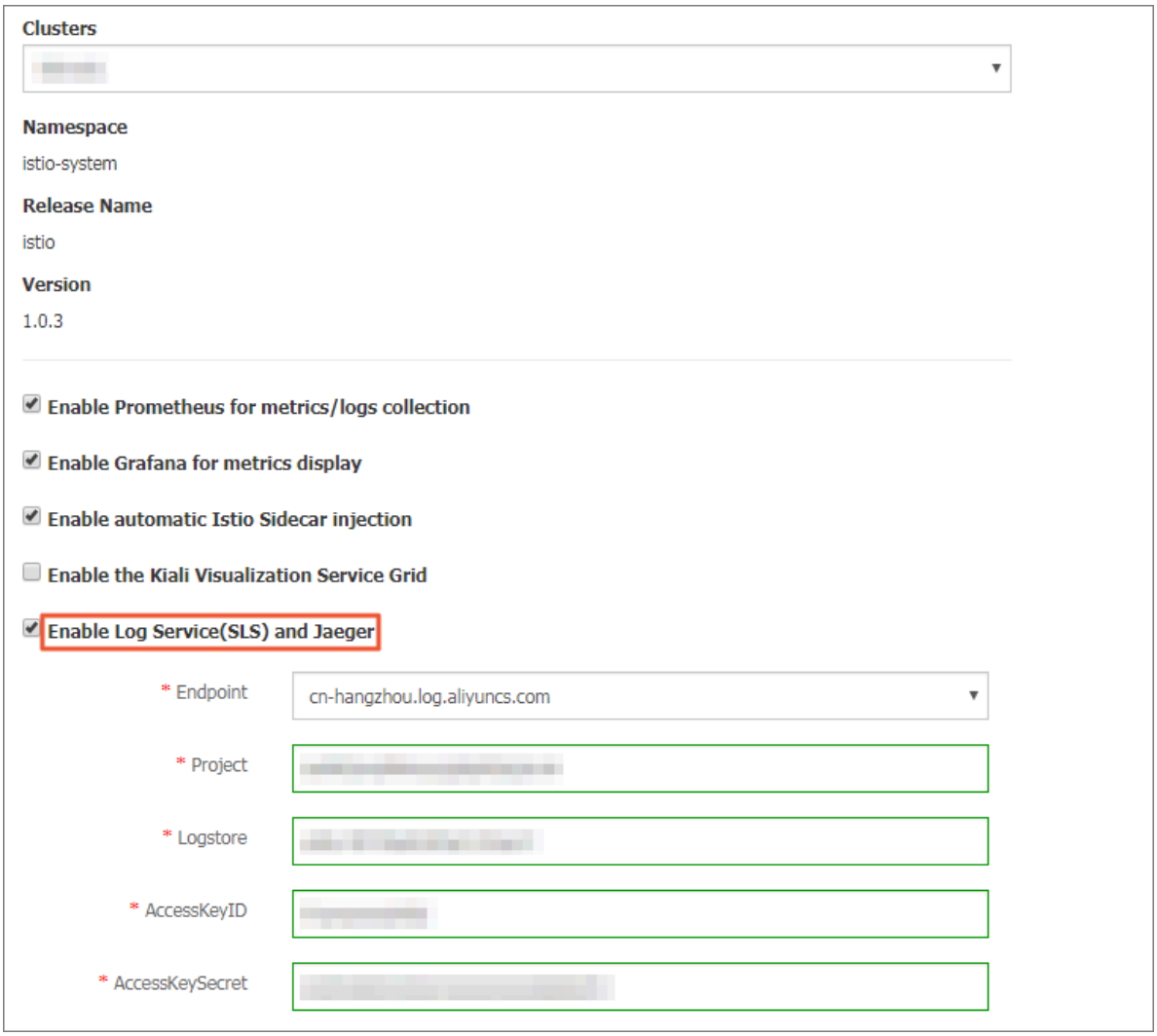
2. In the left-side navigation pane, choose Clusters, select a cluster, and click More > Deploy Istio.

> **Note:**
> In this example, you need to enable Log Service (SLS) and Jaeger. For more information, see *Deploy Istio*. This example uses Istio V1.0.



3. On the Istio deployment page, select Log Service(SLS) and Jaeger.

   Container Service for Kubernetes is integrated with Log Service. Its distributed tracing data is saved to the Logstore of Log Service.



   The descriptions of Log Service parameters are as follows.

Name	Parameter	Description
Endpoint	`storage . aliyun_sls . endpoint`	Specifies the endpoint in which the Project resides to store spans.

Name	Parameter	Description
Project	`storage . aliyun_sls . project`	Specifies the Project that stores spans. The Project name must be a string of 3 to 63 bytes and must contain only lowercase letters, numbers, and hyphens (-). In addition, the name must start and end with lowercase letters or numbers.
Logstore	`storage . aliyun_sls . logstore`	Specifies the Logstore that stores spans. The Logstore name must be a string of 3 to 63 bytes that must contain lowercase letters, numbers, hyphens (-), and underscores (_). The name must start and end with lowercase letters or numbers. A Logstore name must be unique in the Project to which it belongs.
AccessKeyID	`storage . aliyun_sls . accesskey . id`	Specify the user Access Key ID.
AccessKeySecret	`storage . aliyun_sls . accesskey . secret`	Specify the user Access Key Secret.

 Note:
If you specify a Project that does not exist, the system automatically creates and initializes a new Project.

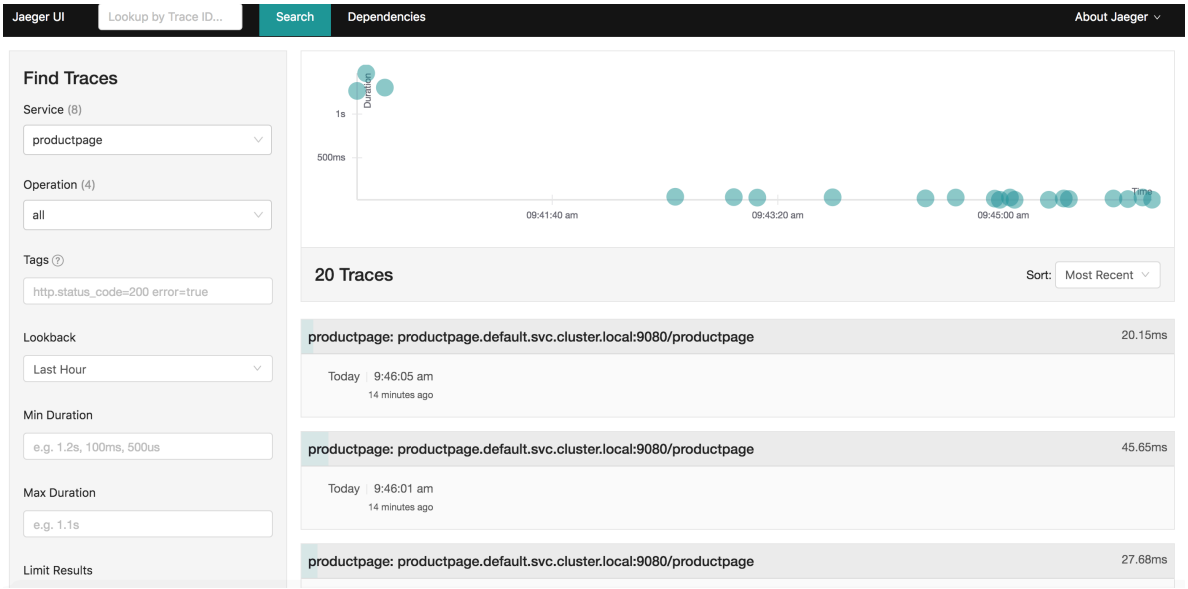
4. Click Deploy Istio. After a few minutes, the Istio integrated with Alibaba Cloud Log Service is created.
5. In the left-side navigation pane, when you choose Application > Service, the Istio services, including the integrated Log Service, are displayed on the right.

Step 2: View Jaeger on Aliyun Log Service

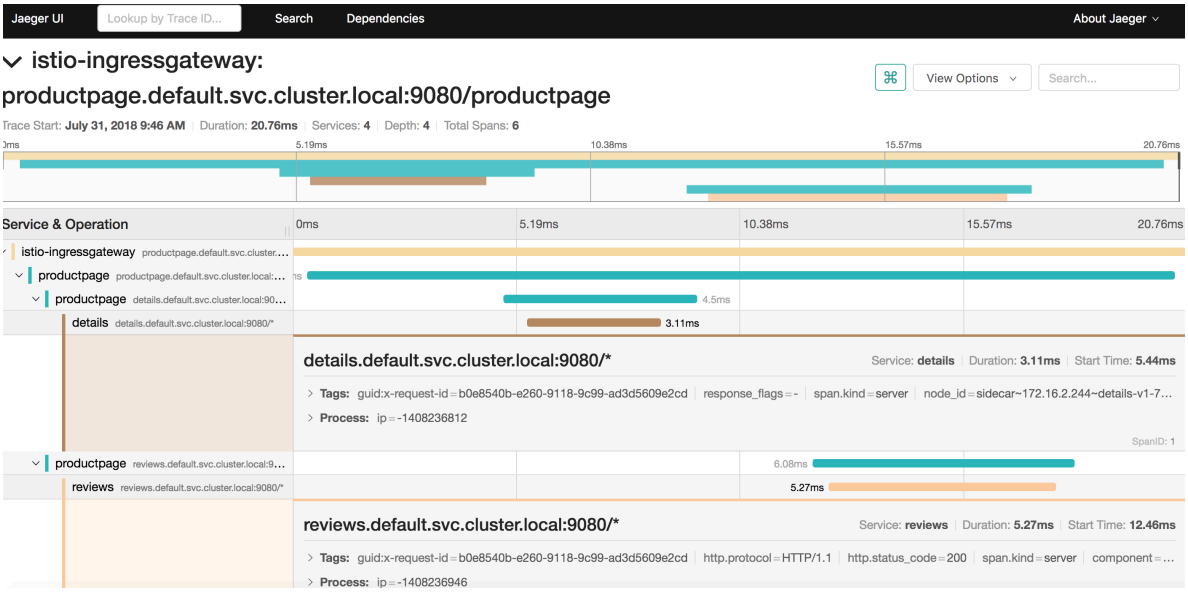
1. Log on to the *Container Service console*.
2. In the left-side navigation pane, click Application > Service, click a cluster and the istio-system namespace, and then click the external endpoint of the tracing-on-sls-query service.

3. On the left side, select a service from the Service drop-down list, set other parameters, and click Find Traces. The scatter diagram of time and duration displayed in the upper-right corner displays the results and provides drill-down capability.

> **Note:**
> In this example, the Bookinfo sample application is used to view the productpage service calling.



4. You can select from multiple views that display trace results in different ways. For example, you can select a histogram within a tracing period, or the service duration time of the tracing process.



Step 3: View Log Service

1. Log on to the *Log Service console*.
2. Select a Project and click the Project name.
3. Click Search on the right of a Logstore.

4. Select a log query time range, and then click Search & Analysis.

   With Log Service, you can use query and analysis statements and other extended functions to optimize your query. For example, you can use analysis charts.

   The following is a statistical chart.

Conclusion

Alibaba Cloud Container Service for Kubernetes can integrate Istio with Log Service . This topic uses an official example to describe how to integrate Istio with the distributed tracing system based on Alibaba Cloud Log Service.

We recommend that you use Alibaba Cloud Container Service for Kubernetes to quickly build Istio, an open management platform for microservices, and integrate Istio with the microservice development of your project.

5.4 Use Istio to deploy application services across Kubernetes and ECS instances

Starting from v0.2, Istio provides mesh expansion. With this feature, you can integrate non-Kubernetes services that typically run on VMs or bare metal hosts with the Istio service mesh that runs on your Kubernetes cluster.

Alibaba Cloud Container Service for Kubernetes supports the Istio mesh expansion capabilities. This topic uses an example from the Istio official website to details how to use Istio to deploy application services across Kubernetes and ECS instances.

Mesh expansion

Mesh expansion is a method based on the Istio service mesh deployed on Kubernetes. With this method, you can integrate VMs or bare metal hosts into the service mesh.

Mesh expansion is suitable for when you need to migrate your applications from your local system to cloud services. In a microservices system, not all workloads can run in Kubernetes. This means you may encounter scenarios in which you can only operate and maintain some services in Kubernetes, while other services run on VMs or bare metal hosts.

With the Istio control plane, you can manage services across Kubernetes and VMs or bare metal hosts, and ensure that all your services can continue to run normally.

Create a Kubernetes cluster and install Istio

Alibaba Cloud Container Service for Kubernetes 1.11.5 is now available. You can quickly create a Kubernetes cluster through the Container Service console. For more information, see *Create a Kubernetes cluster*.

Note:

You must make sure that you can connect to your Kubernetes cluster by using kubectl. For more information, see *Connect to a Kubernetes cluster by using kubectl*.

Deploy Istio through the app catalog. Create the `istio - system` namespace through a command or the console.

1. Log on to the *Container Service console*.
2. In the left-side navigation pane, choose Store > App Catalog, and click `ack - istio` on the right side.

3. On the displayed page, select `istio - system` from the namespace drop-down list, and click Values. You can edit parameters to customize your Istio.

Note:
The readme document on the page provides the installation and removal information, including common questions about Custom Resource Definition (CRD) versions.

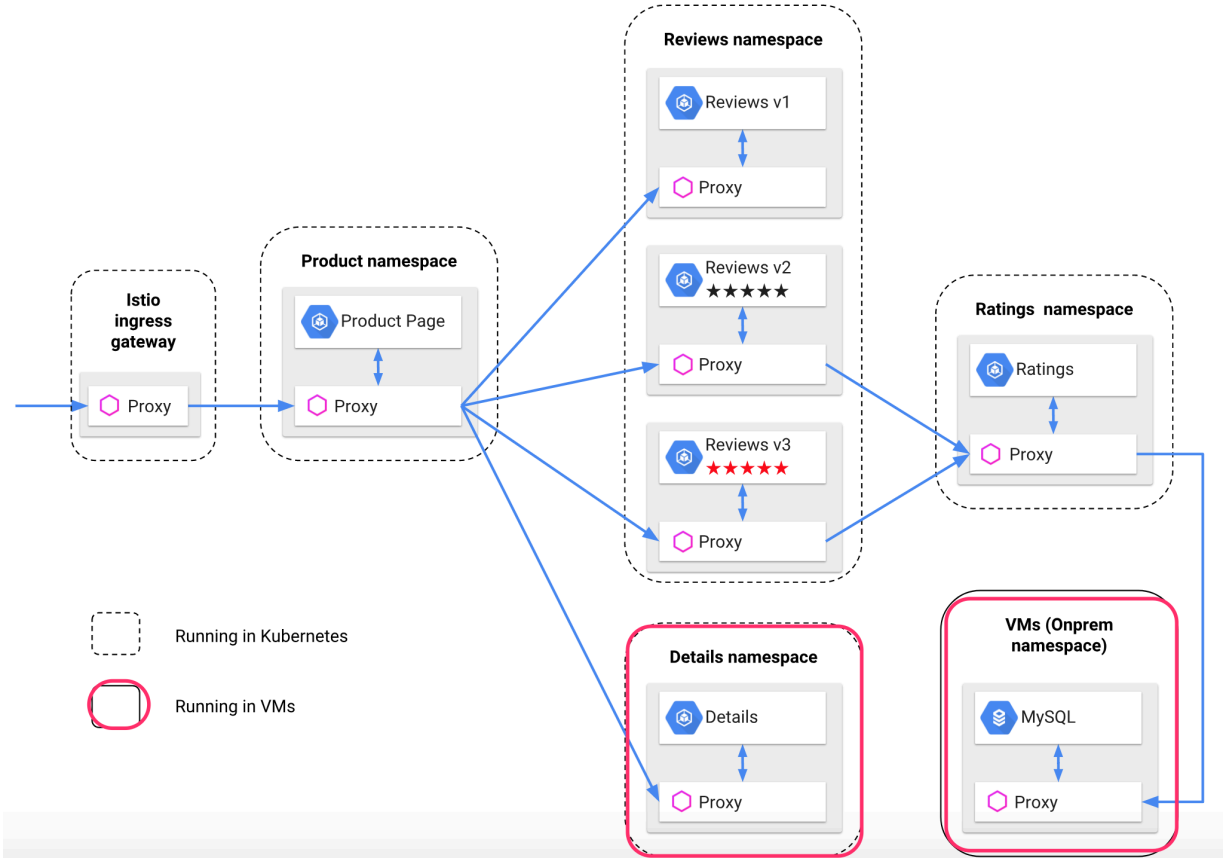
Install the sample application in your Kubernetes cluster

Run the following commands or use the console to create the `bookinfo` namespace, and then deploy the modified application. In the modified application, the `details` component is removed and `ingressgat eway` is defined.

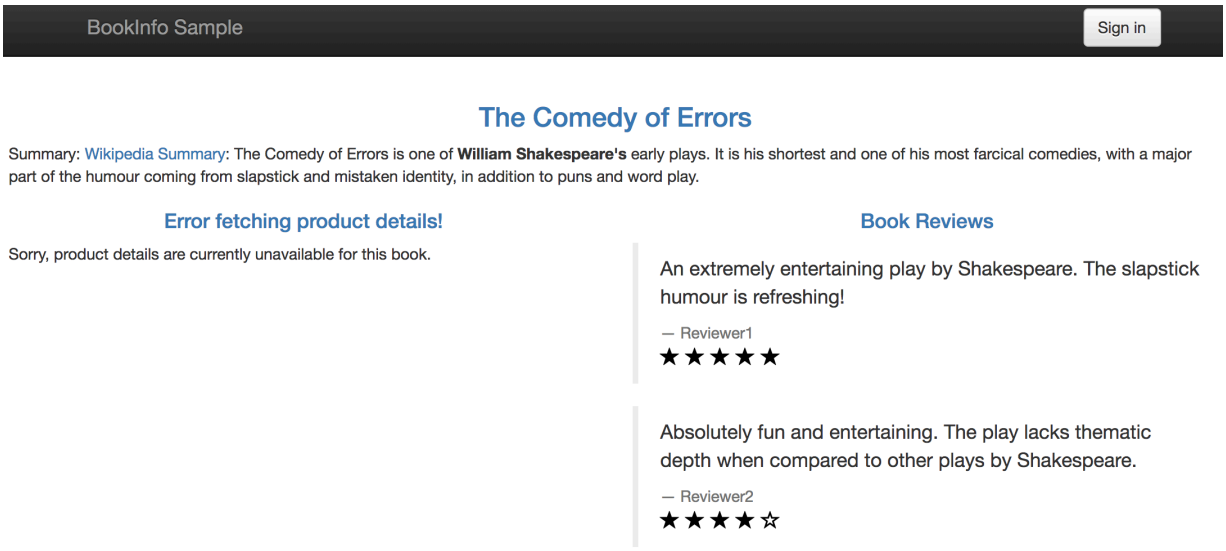
To obtain the files used in this example, see *Istio multi-cluster sample files*.

```
kubectl   create   ns   bookinfo
kubectl   label   namespace   bookinfo   istio - injection = enabled
kubectl   apply  - n   bookinfo  - f  ./ bookinfo / bookinfo -
without - details . yaml
kubectl   apply  - n   bookinfo  - f  ./ bookinfo / bookinfo -
gateway . yaml
```

Both the `details` and the database components of the application deployment run on the ECS instance that is outside the Kubernetes system.



Access the `/productpage` page through the address exposed by `ingressgateway` and verify that the `details` part cannot be displayed.



Configure your Kubernetes

1. If you have not set internal load balancers for Kube DNS, Pilot, Mixer, and Citadel when you install Istio, you need to run the following command:

   ```
   kubectl apply -f ./mesh-expansion.yaml
   ```

   As shown in the following figure, the four services are created.

   ```
   ali-1c36bbed0b91:meshexpansion wangxn$ kubectl apply -f ./mesh-expansion.yaml
   service "istio-pilot-ilb" created
   service "dns-ilb" created
   service "mixer-ilb" created
   service "citadel-ilb" created
   ```

2. Generate the *cluster.env* Istio configuration file and the *kubedns* DNS configuration file both of which are to be deployed in the VMs. The *cluster.env* file contains the range of the cluster IP addresses that will be intercepted. The *kubedns* file contains the cluster service names that can be resolved by the applications on the VMs and then will be intercepted and forwarded by the sidecar.

   To generate the configuration files, run the following command:

   ```
   ./setupMeshEx.sh generateClusterEnvAndDnsmasq
   ```

   Configuration file *cluster.env*

   ```
   ali-1c36bbed0b91:meshexpansion wangxn$ cat cluster.env
   ISTIO_SERVICE_CIDR=[illegible]
   ISTIO_SYSTEM_NAMESPACE=istio-system
   ISTIO_CP_AUTH=MUTUAL_TLS
   ```

   Configuration file *kubedns*

   ```
   ali-1c36bbed0b91:meshexpansion wangxn$ cat kubedns
   server=/svc.cluster.local/[illegible]
   address=/istio-policy/[illegible]
   address=/istio-telemetry/[illegible]
   address=/istio-pilot/[illegible]
   address=/istio-citadel/[illegible]
   address=/istio-ca/[illegible]
   address=/istio-policy.istio-system/[illegible]
   address=/istio-telemetry.istio-system/[illegible]
   address=/istio-pilot.istio-system/[illegible]
   address=/istio-citadel.istio-system/[illegible]
   address=/istio-ca.istio-system/[illegible]
   ```

Set the ECS instance

Configure your working environment to communicate with the ECS instance. Generate an SSH key and assign it to the ECS instance. You can run the `ssh root@< ECS_HOST_IP >` command to check if you can connect to the ECS instance.

To generate a public key, run the following command:

```
ssh-keygen -b 4096 -f ~/.ssh/id_rsa -N ""
```

Note:

To ensure that the ECS instance and Kubernetes are mutually accessible over the Internet, you need to add them to the same security group.

With Alibaba Cloud Container Service for Kubernetes, you can quickly configure an ECS instance by running the following script:

```
export SERVICE_NAMESPACE=default
./setupMeshEnv.sh machineSet up root@< ECS_HOST_IP >
```

Run the following command to check the running process:

```
ps aux | grep istio
```

```
root@remotevm:~# ps aux |grep istio
root     19460  0.0  0.0  52284  3404 ?        Ss   11:59   0:00 su -s /bin/bash -c INSTANCE_IP=192.168.3.70 POD_NAME=remotevm POD_NAMESPACE=default exec /u
sr/local/bin/pilot-agent proxy      --serviceCluster rawvm     --discoveryAddress istio-pilot.istio-system:8080     --controlPlaneAuthPolicy MUTUAL_TLS
2> /var/log/istio/istio.err.log > /var/log/istio/istio.log istio-proxy
istio-p+ 19508  0.0  0.0  45276  4592 ?        Ss   11:59   0:00 /lib/systemd/systemd --user
istio-p+ 19510  0.0  0.0  61324  2064 ?        S    11:59   0:00 (sd-pam)
istio-p+ 19516  0.0  0.2  31204 17252 ?        Ssl  11:59   0:00 /usr/local/bin/pilot-agent proxy --serviceCluster rawvm --discoveryAddress istio-pilot.isti
o-system:8080 --controlPlaneAuthPolicy MUTUAL_TLS
istio-p+ 19537  7.1  0.5 133208 41572 ?        Sl   11:59   0:02 /usr/local/bin/envoy -c /etc/istio/proxy/envoy-rev1.json --restart-epoch 1 --drain-time-s 2
 --parent-shutdown-time-s 3 --service-cluster rawvm --service-node sidecar~192.168.3.70~remotevm.default~default.svc.cluster.local --max-obj-name-len 189 -l
 warn --v2-config-only
```

Run the following command to check if the node agent authenticated by Istio is running in a healthy status:

```
sudo systemctl status istio-auth-node-agent
```

Run services on the ECS instance

As shown in the preceding deployment figure, two services run on the ECS instance: one is the Details service, the other one is the Database service.

Run the Details service on the ECS instance

Run the following commands to simulate (by using Docker only) the `Details` service, run the service on the ECS instance, and expose port 9080 for the service.

```
docker pull istio/examples-bookinfo-details-v1:1.8.0
docker run -d -p 9080:9080 --name details-on-vm istio/examples-bookinfo-details-v1:1.8.0
```

Configure the sidecar to intercept the port. You need to configure this in the */var/lib/istio/envoy/sidecar.env* path and use the `ISTIO_INBOUND_PORTS` environment variable.

Run the following command on the VM in which the service runs:

```
echo "ISTIO_INBOUND_PORTS=9080,8080" > /var/lib/istio/envoy/sidecar.env
systemctl restart istio
```

Register the Details service with Istio

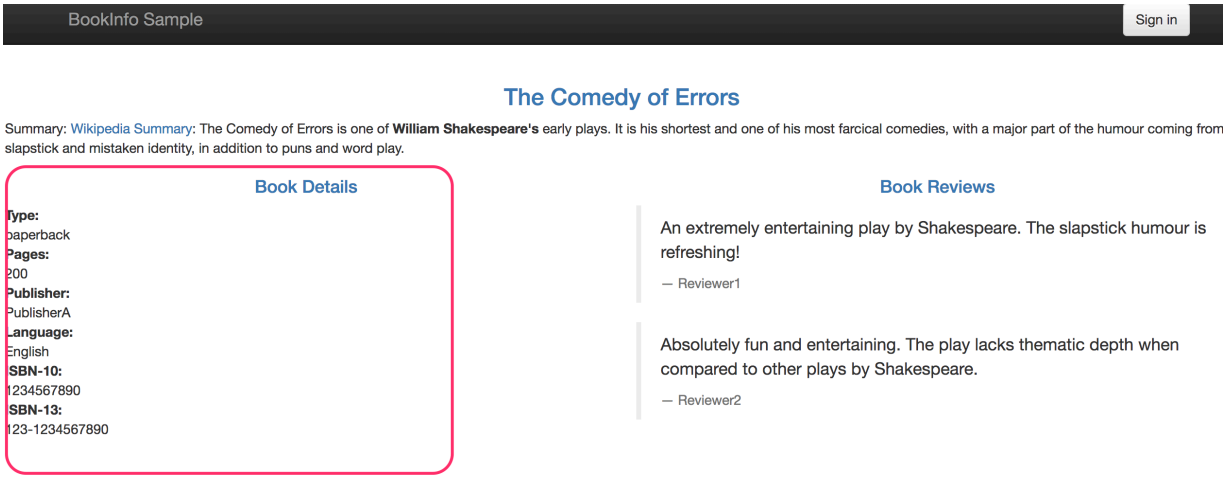
Run the following command to view the IP address of the VM so that you can add it to the service mesh:

```
hostname -I
```

Manually configure a selector-less service and endpoints. The selector-less service is used to host services that are not backed by Kubernetes pods. For example, run the following command to register the `Details` service on a server that has the permissions to modify Kubernetes services and supports istioctl commands:

```
istioctl -n bookinfo register details 192.168.3.202 http:9080
```

Access the `/productpage` page again to verify that the `details` part is displayed as shown in following figure.

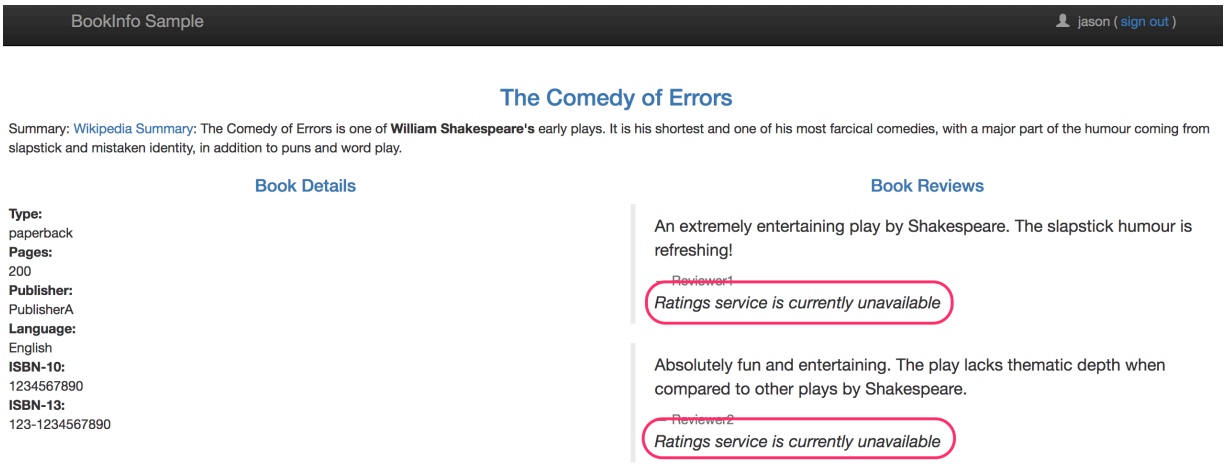


Update the Ratings service to the version that can access a database

By default, the Ratings service cannot access any database. Run the following command to update the service version so that the service can access the database:

```
kubectl apply -f ./bookinfo/bookinfo-ratings-v2-mysql-vm.yaml
kubectl apply -f ./bookinfo/virtual-service-ratings-mysql-vm.yaml
```

Access the `/productpage` page to verify that the `Ratings` part cannot be displayed as shown in the following figure. Then, you need to build a database service on the ECS instance and add the service to Istio.



Run a database service on the ECS instance

On the VM, run MariaDB as the backend for the Ratings service, and set MariaDB to be remotely accessible.

```
apt-get update && apt-getinstall -y mariadb-server
```

```
sed - i ' s / 127 \. 0 \. 0 \. 1 / 0 \. 0 \. 0 \. 0 / g ' /
etc / mysql / mariadb . conf . d / 50 - server . cnf
sudo mysql
# Grant the root permission .
GRANT ALL PRIVILEGES ON *. * TO ' root '@' localhost '
IDENTIFIED BY ' password ' WITHGRANTO PTION ;
quit ;
sudo systemctl restart mysql
```

Run the following command to initialize the Ratings database on the VM:

```
curl - q https :// raw . githubuser content . com / istio / istio
/ master / samples / bookinfo / src / mysql / mysqldb - init . sql
| mysql - u root - ppassword
```

To view different outputs of the Bookinfo application, run the following command to modify the rating records to generate different rating data that are displayed on the page:

```
mysql - u root - ppassword test - e " select * from
ratings ;"
mysql - u root - ppassword test - e " update ratings set
  rating = 2 ; select * from ratings ;"
```

Register the database service into Istio

Configure the sidecar to intercept the port. You need to configure this in the */ var / lib / istio / envoy / sidecar . env* path and use the `ISTIO_INBOUND_PORTS` environment variable.

Run the following command on the VM in which the service runs:

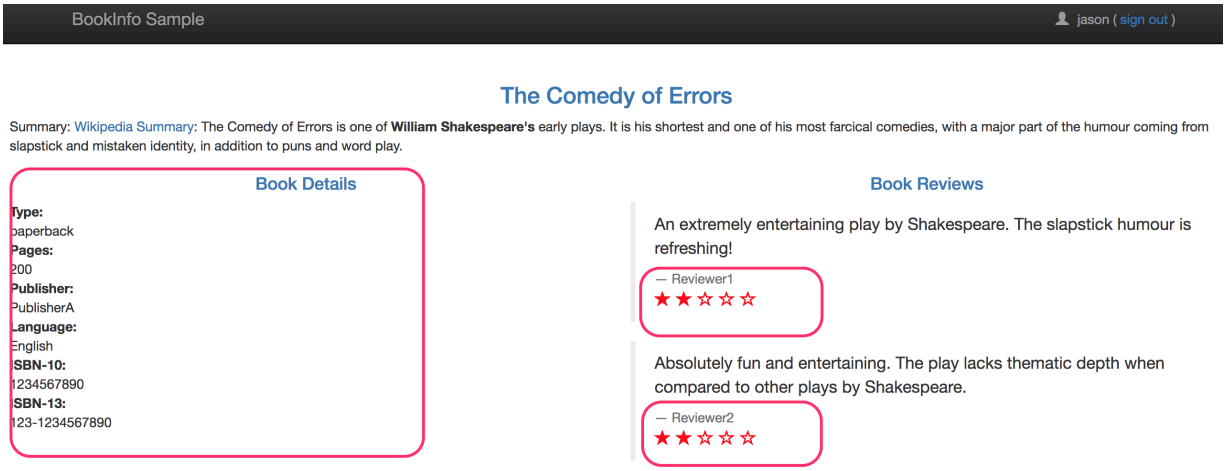
```
echo " ISTIO_INBOUND_PORTS = 3306 , 9080 , 8080 " > / var / lib /
istio / envoy / sidecar . env
systemctl restart istio
```

Run the following command to register the database service on a server that has the permissions to modify Kubernetes services and supports istioctl commands:

```
istioctl - nbookinforegistermysqldb 192 . 168 . 3 . 202 3306
```

Now Kubernetes pods and other servers included by mesh expansion can access the database service running on this server.

Access the `/ productpage` page to verify that both the Details and Ratings parts can be displayed and these two services are provided by the ECS instance.



Conclusion

Alibaba Cloud Container Service for Kubernetes provides the Istio mesh expansion capabilities. This topic uses a sample application from the Istio official website to details how to use Istio to deploy application services across Kubernetes and ECS instances.

We recommend that you use Alibaba Cloud Container Service for Kubernetes to quickly build Istio, an open management platform for microservices, and integrate Istio with the microservice development of your project.

5.5 Use the Canary method that uses Istio to deploy a service

This topic describes how to use the Canary method (that uses Istio) to deploy a service through Alibaba Cloud Container Service for Kubernetes (ACK).

Background information

Istio is a service mesh that can be used to meet the requirements of the distributed application architectures that involve microservices such as application O&M, debugging, and security management. You also can use Istio for microservice network scenarios such as load balancing, service-to-service authentication, and monitoring.

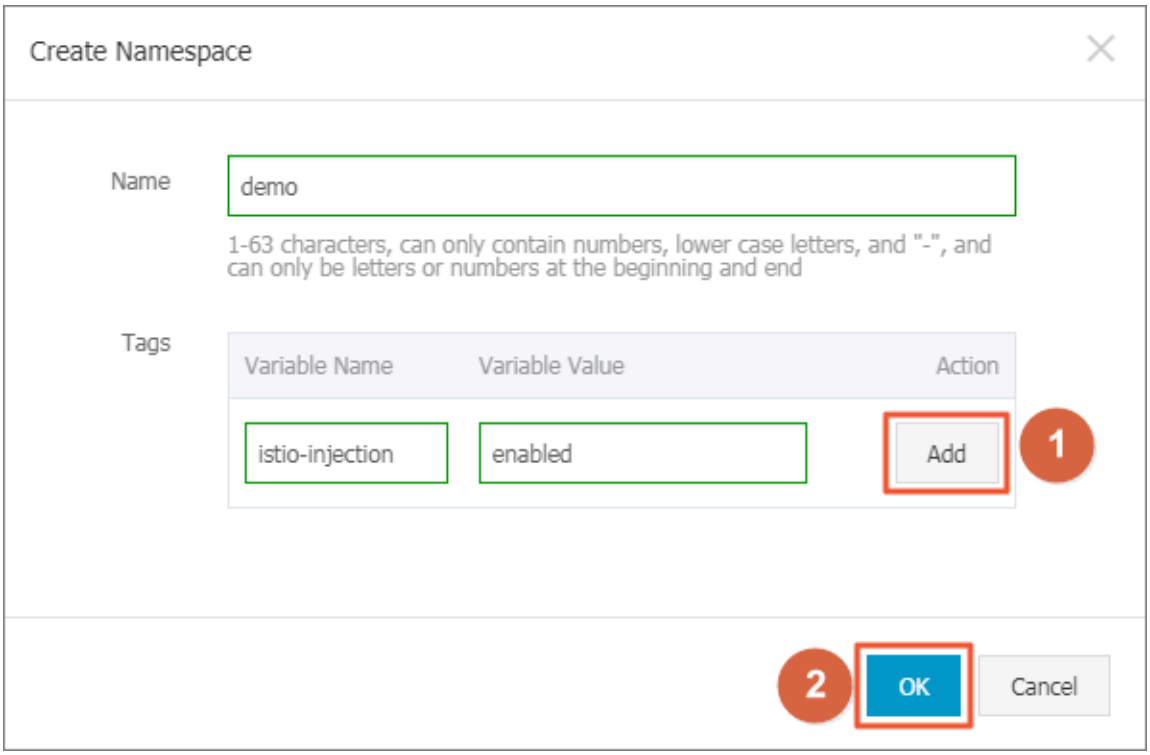
By using Istio, ACK allows you to manage services of an application. With ACK, you can create and manage multiple versions of one service for an application, manage the distribution of traffic destined for the service, and implement a Canary release for the service.

Prerequisites

- A Kubernetes cluster is created. For more information, see *Create a Kubernetes cluster*.
- Istio is deployed for this Kubernetes cluster. For more information, see *Deploy Istio*.

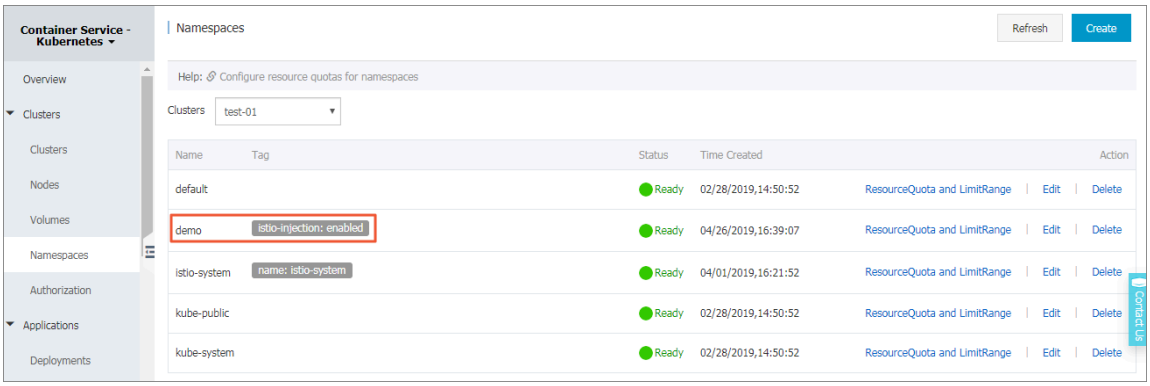
Procedure

1. Enable automatic sidecar injection.
   a. Log on to the *Container Service console*.
   b. In the left-side navigation pane under Container Service-Kubernetes, choose Clusters > Namespaces.
   c. Click Create.
   d. In the displayed dialog box, set Name to `demo` , and set Tag to follows:
      - The variable name of the tag is set to `istio - injection` .
      - The variable value of the tag is set to `enabled` .

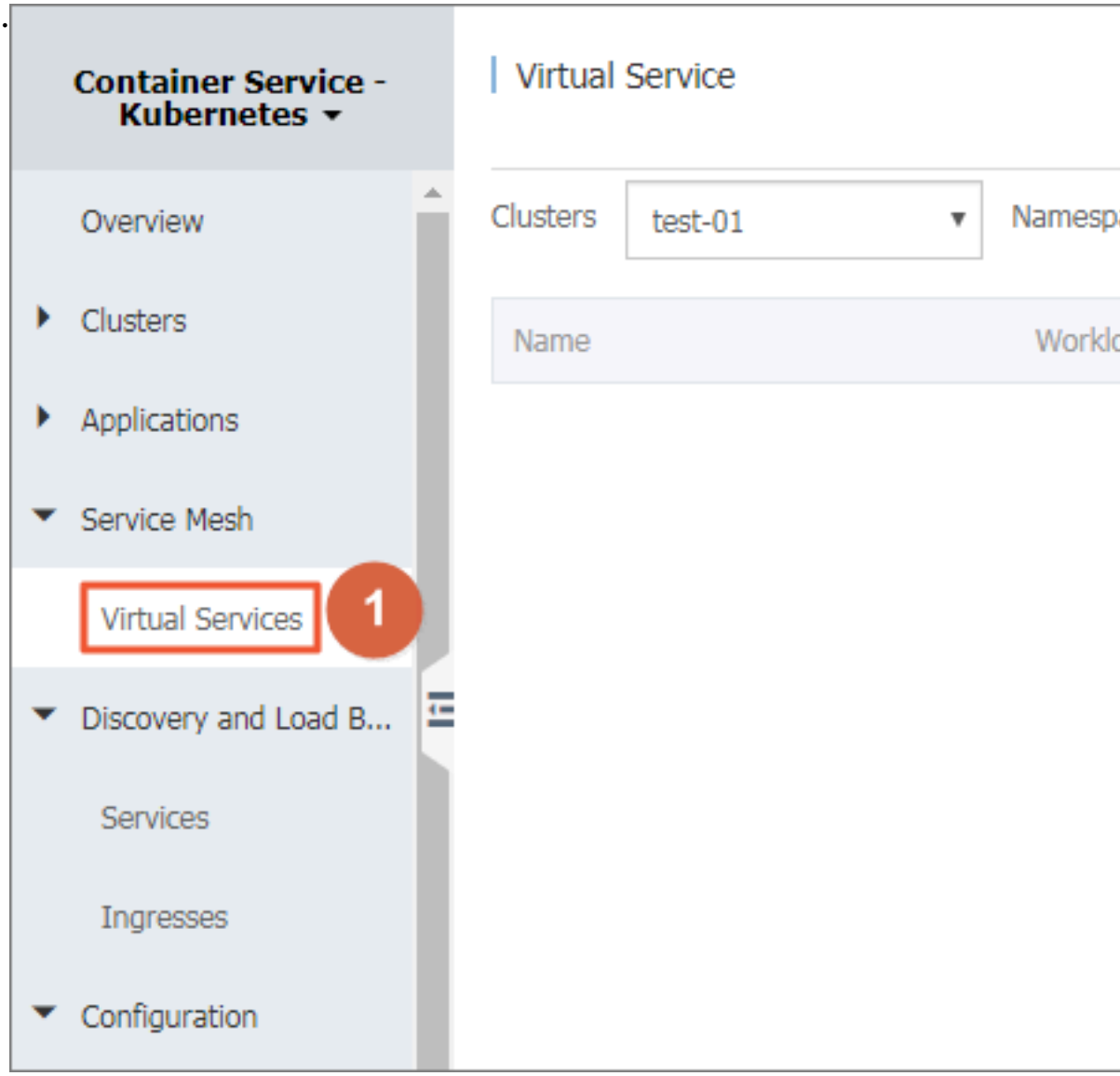


   e. Click Add, and then click OK.

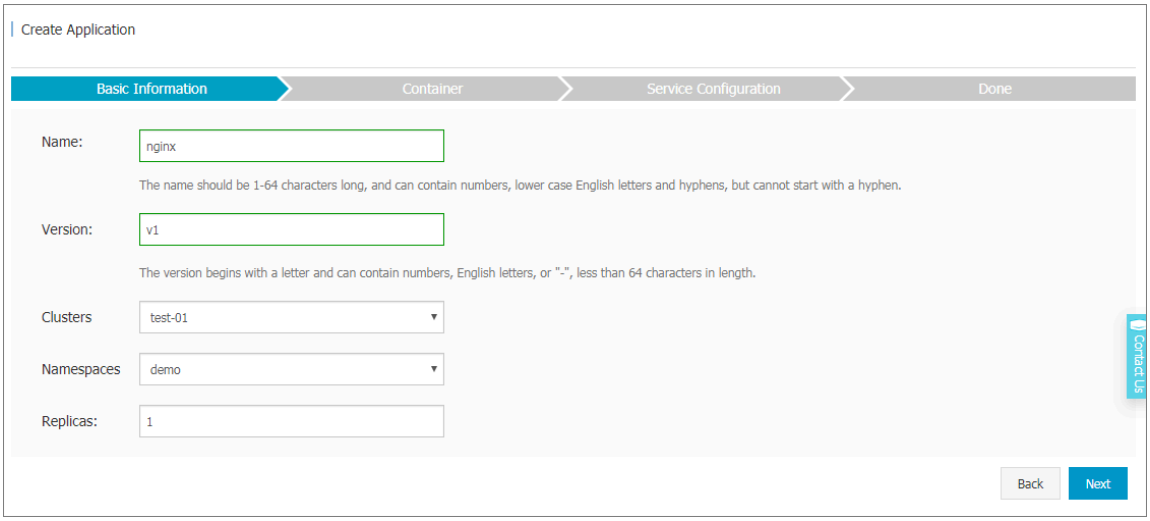
   Then, the newly created namespace is displayed in the namespace list.



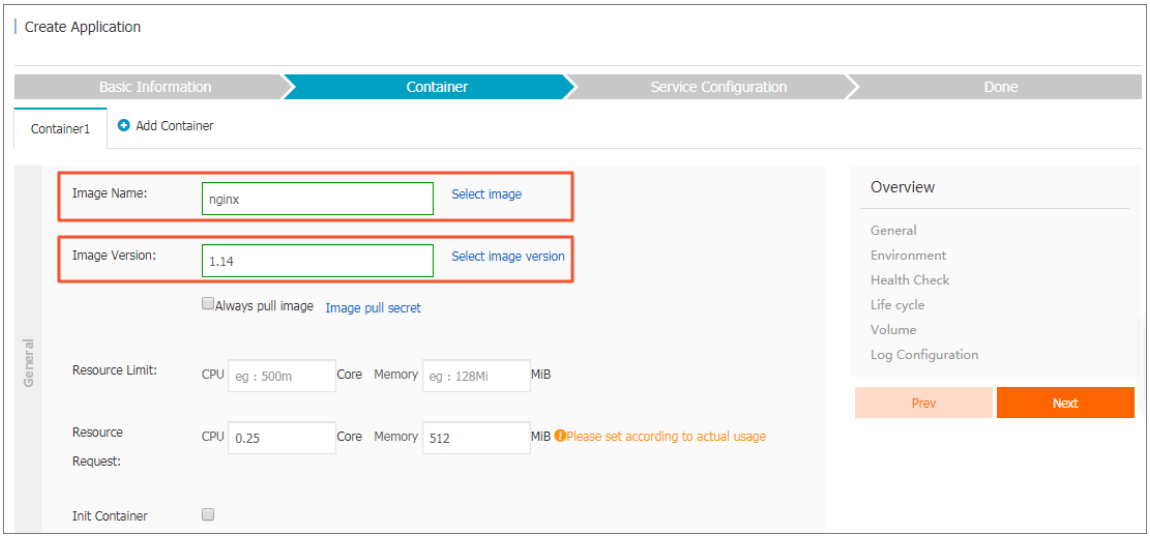
2. Create a server-end application and its virtual service.
   a. Log on to the *Container Service console*.
   b. In the left-side navigation pane, choose Service Mesh > Virtual Service.
   c. In the upper-right corner of the page, click Create.



   d. On the Basic Information tab page, set the required parameters.
      - Name: Set the application name.
      - Version: Set the application version.
      - Clusters: Select the cluster where you want to deploy the application.
      - Namespaces: Select the namespace where you want to deploy the application. You must select the namespace created in step 1.
      - Replicas: Set the number of replicas, that is, the number of pods.

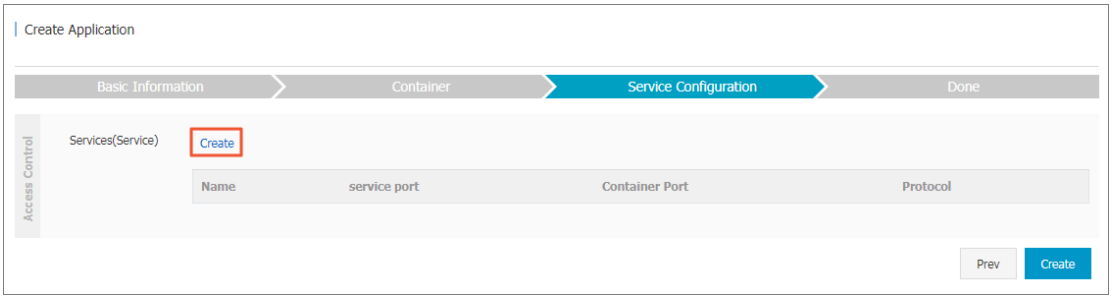


e. Configure the container. Set Image Name to nginx, and set Image Version to 1.14. For more information about other container settings, see *Create a deployment application by using an image*. Then, click Next.



f. Set the Service parameters.

A. In the upper-left corner, click Create. Then, in the displayed dialog box, set the required parameters.



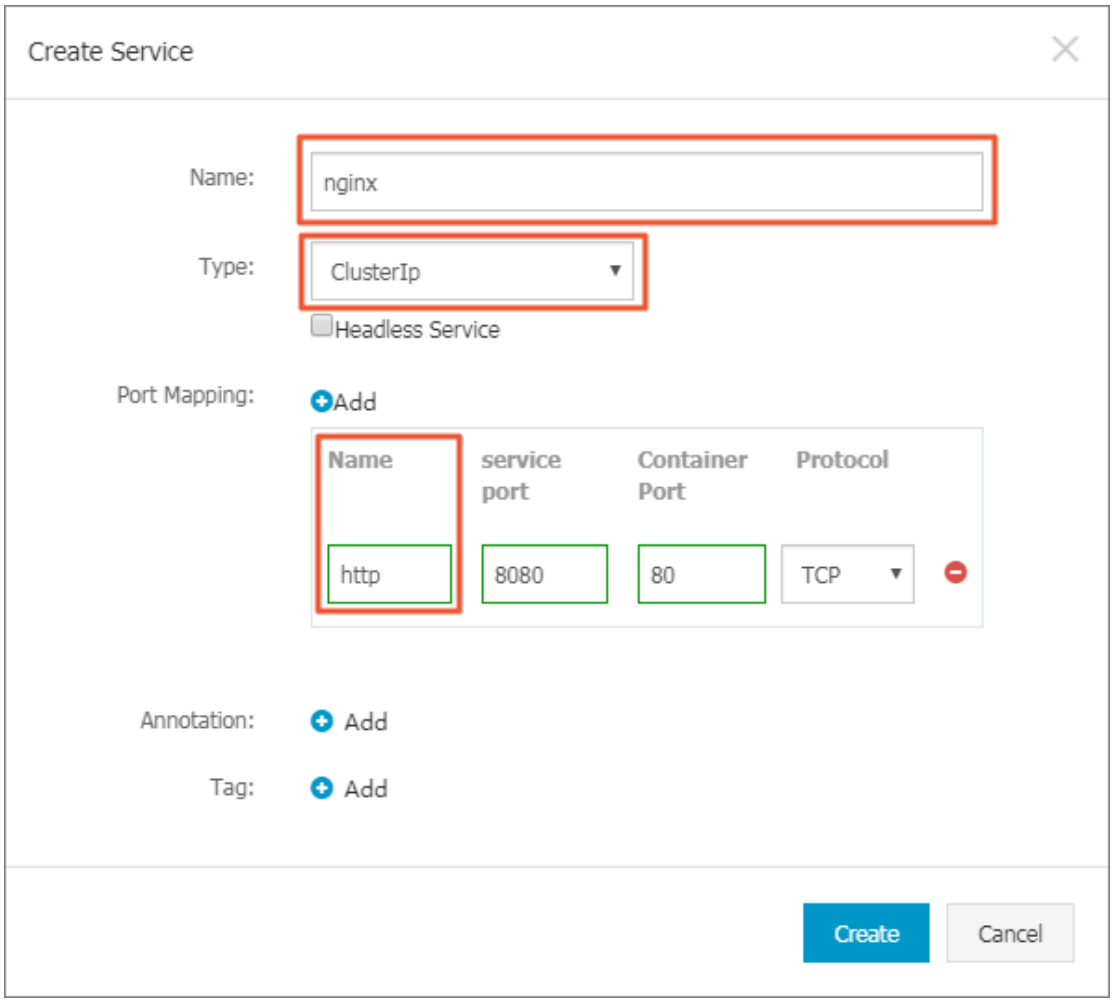
- Name: Enter the service name.

- Type: Select the ClusterIP service type.
- Port Mapping: Set the service port name, service port number, container port number, and TCP protocol. The format of a service port name must be <protocol>[-<suffix>-]. The protocol part of a service port name can be one of the following: gRPC, HTTP, HTTP/2, HTTPS, Mongo, Redis, TCP, TLS, or UDP. Istio supports a routing service through one of these protocols.

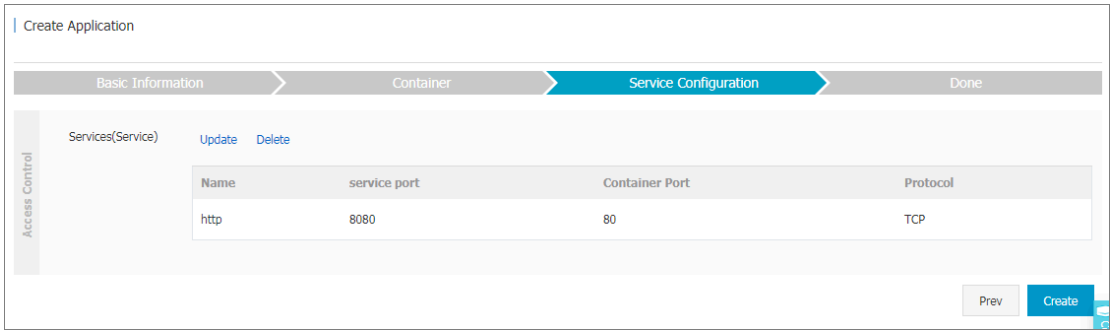
Note:

The protocol part of a service port name is case insensitive.

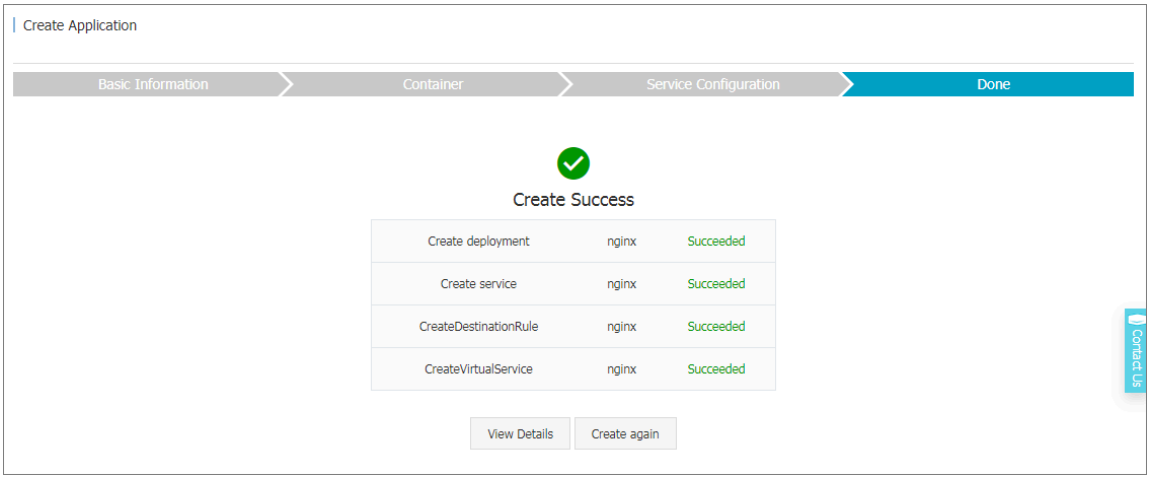
- Annotation: Add an annotation to the service.
- Tag: Add a tag to the service to identify the service.



B. Click Create.



g. In the lower-right corner, click Create. Then, the system displays all the created resources if no error occurs. You can click View Details to view the application details.



3. Create a client-end application named `sleep`. This application will be used to send access traffic to the created virtual service.

   a. Save the following as a file named *sleep.yaml*.

```
apiVersion: v1
kind: Service
metadata:
  name: sleep
  labels:
    app: sleep
spec:
  ports:
  - port: 80
    name: http
  selector:
    app: sleep
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: sleep
    spec:
      containers:
      - name: sleep
        image: pstauffer/curl
        command: ["/bin/sleep", "3650d"]
```

```
          imagePullP  olicy :  IfNotPrese  nt
```

b. In the demo namespace, run the `kubectl apply -f sleep.yaml -n demo` command.

c. Run the `kubectl exec -it -n demo <podName> bash` command to log on to the pod where the `sleep` application runs, and then run the following commands to call the `nginx` service:

```
for i in `seq 1000`
        do
        curl -I http://nginx.demo:8080;
        echo '';
        sleep 1;
        done;
```

If the service is called, the content shown in the following figure is displayed.

```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

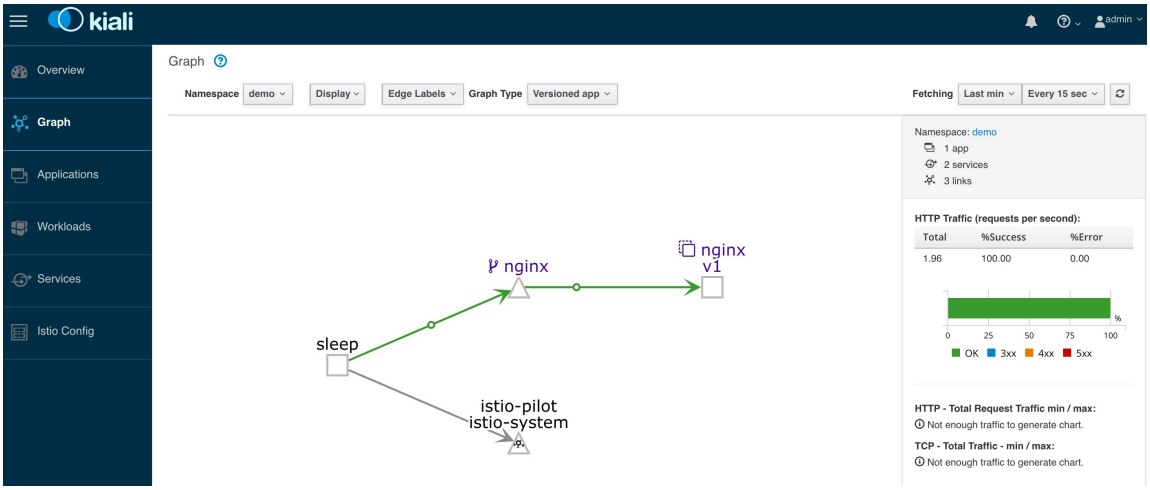
d. Access the Kiali console to verify that the virtual service is called.

Note:

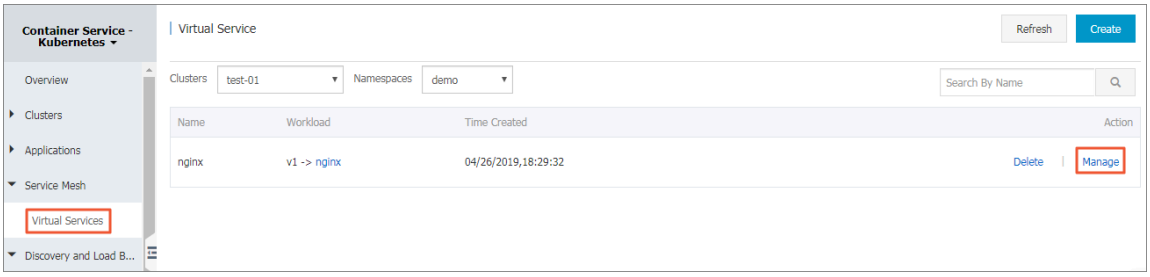
Before you perform this step, you must enable the Kiali visualized service mesh when deploying Istio for the target Kubernetes cluster.

A. Run the `kubectl port-forward svc/kiali -n istio-system 20001:20001` command to open port 20001 in your local host.

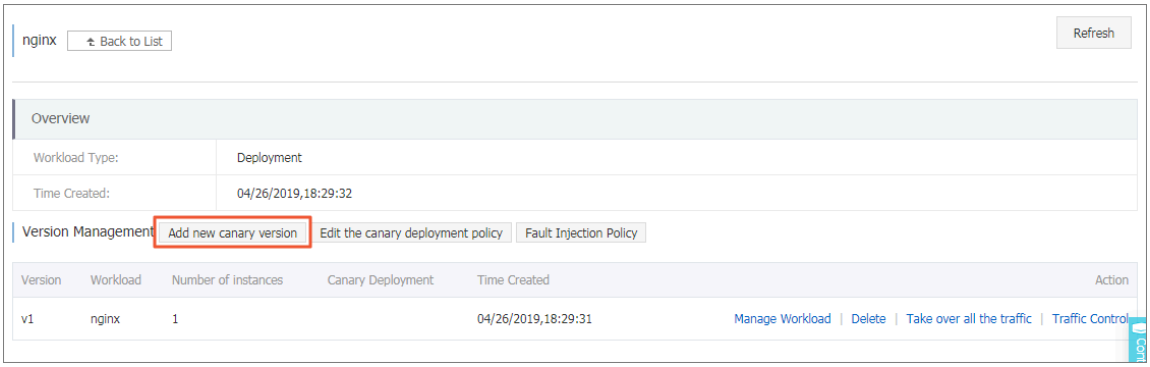
B. Enter *`http://localhost:20001`* in your browser.



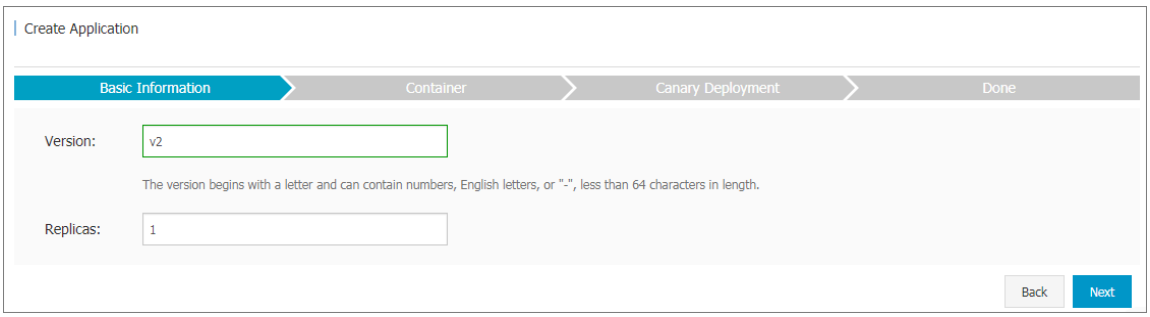
4. Create a version of the virtual service for a Canary release.
   a. Log on to the *Container Service console*.
   b. In the left-side navigation pane under Container Service-Kubernetes, choose Service Mesh > Virtual Service.
   c. On the right of the target virtual service, click Manage.



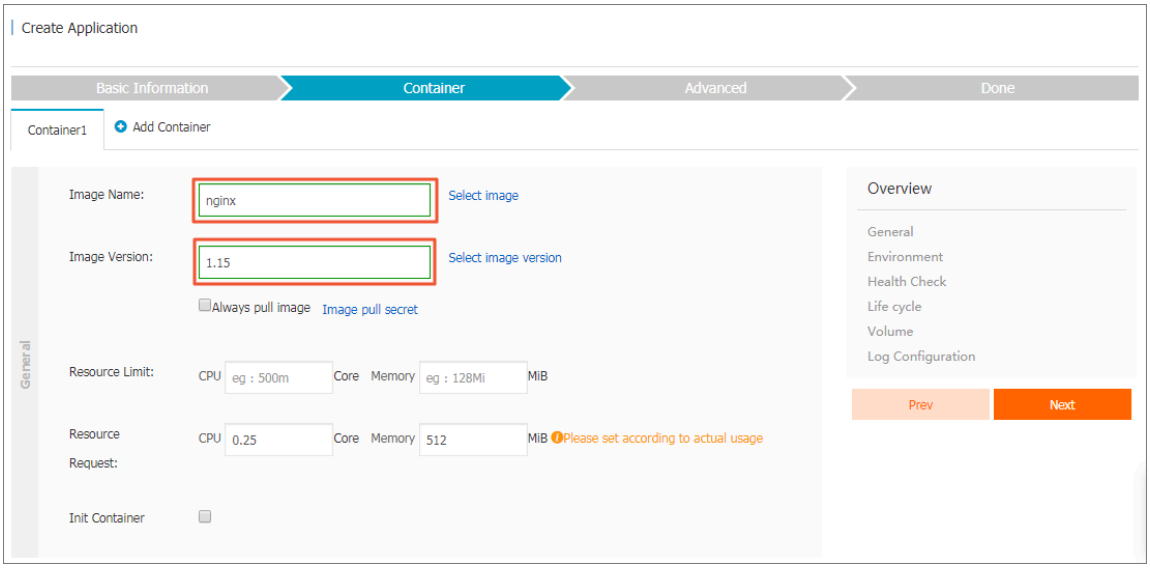
   d. Click Add new canary version.



   e. Set the Version and Replicas, then click Next.



   f. Configure a container. Set Image Name to nginx, and set Image Version to 1.15. For more information about other container configurations, see *Create a deployment application by using an image*. Then, click Next.

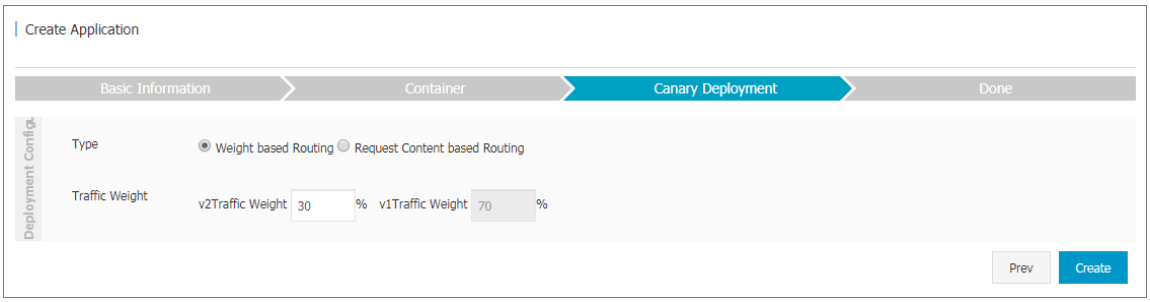


g. Set the Type parameter to select a Canary deployment policy. For example, you can select the Weight based Routing radio button, and then set v2Traffic Weight to `30 %`.

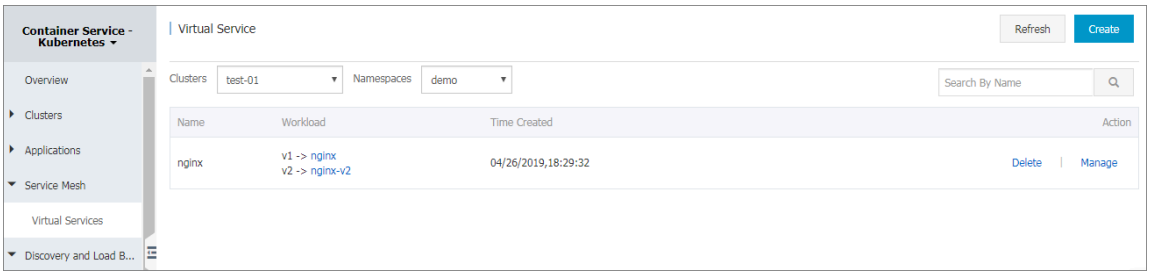
Note:

- If you select the Weight based Routing radio button, then you can set the traffic distribution ratio for versions of the service that is to be deployed.

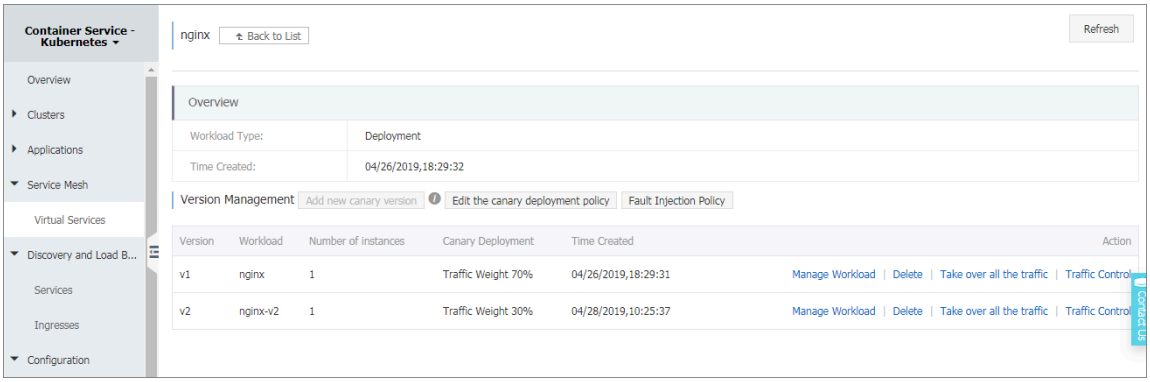
- You only need to set the v2Traffic Weight parameter. The v1Traffic Weight value is shown through automatic calculation.



h. Click Create. After a creation success message is displayed by the system, choose Service Mesh > Virtual Services.



i. On the right of the target service, click Manage.



j. Run the `kubectl exec -it -n demo <podName> bash` command to log on to the pod where the `sleep` application runs, and then run the following commands to call the `nginx` service:

```
for i in `seq 1000`
    do
    curl -I http://nginx.demo:8080;
    echo '';
    sleep 1;
```

```
done ;
```

If the service is called, the content shown in the following figure is displayed.

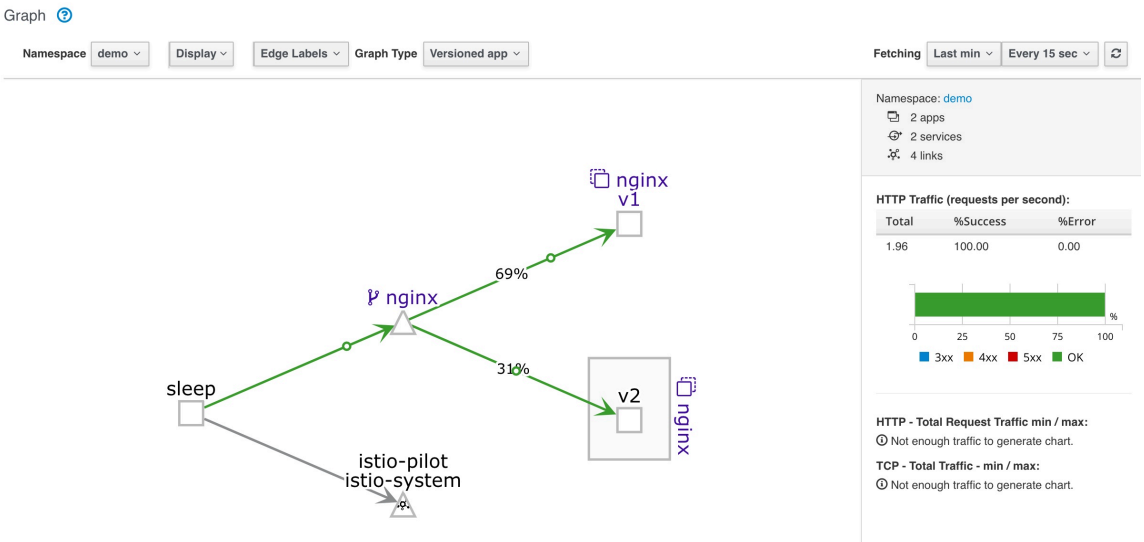
```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

k. Access the Kiali console to verify that the access traffic of the virtual service are distributed to the two versions according to the distribution ratio that you set.

> **Note:**
> Before you perform this step, you must enable the Kiali visualized service mesh when deploying Istio for the target Kubernetes cluster.

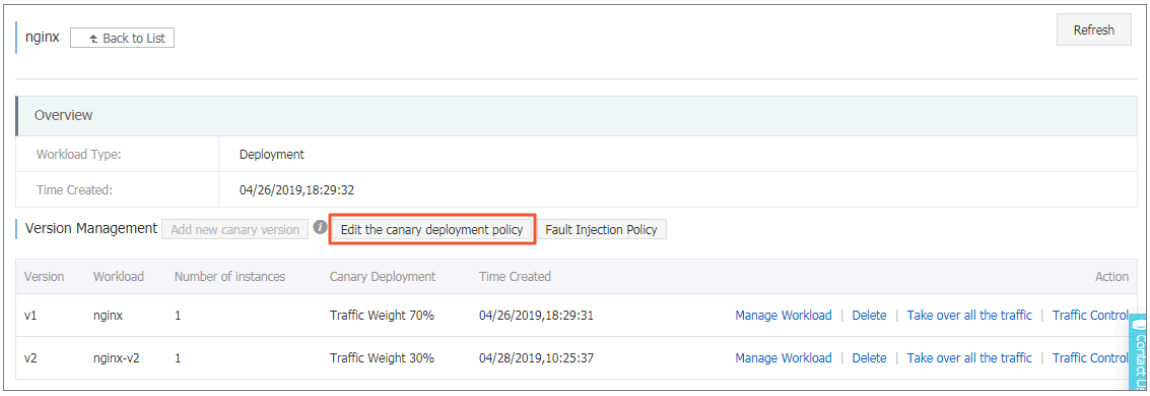
A. Run the `kubectl port-forward svc/kiali -n istio-system 20001:20001` command to open port 20001 in your local host.

B. Enter *`http://localhost:20001`* in your browser.

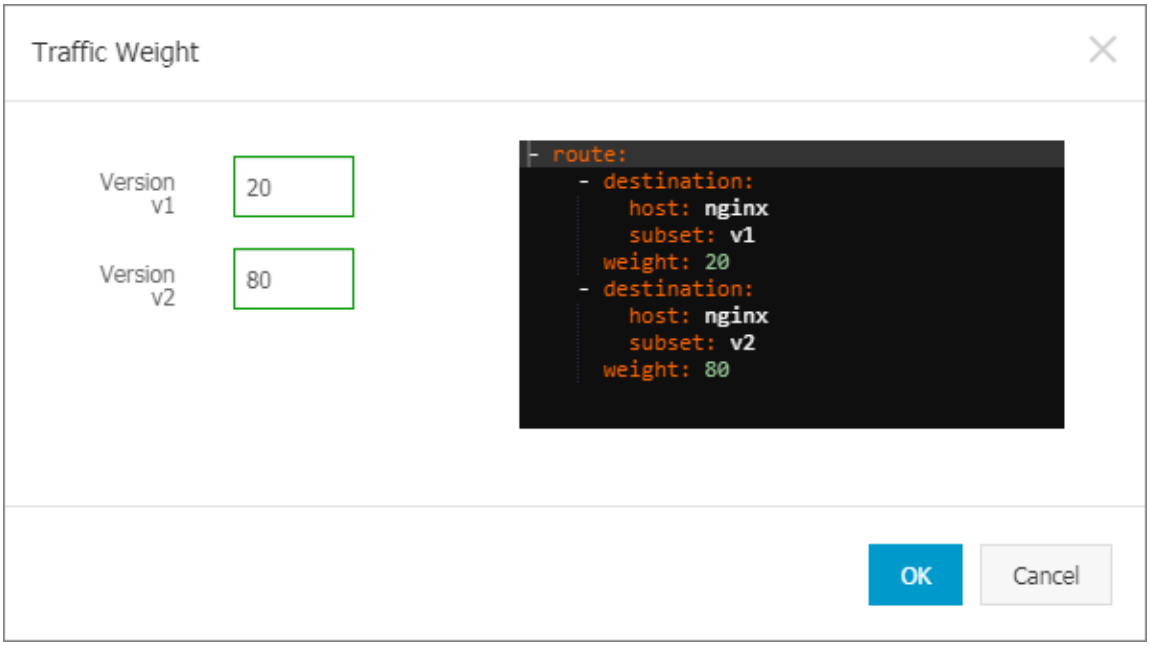


5. Edit the Canary release policy.

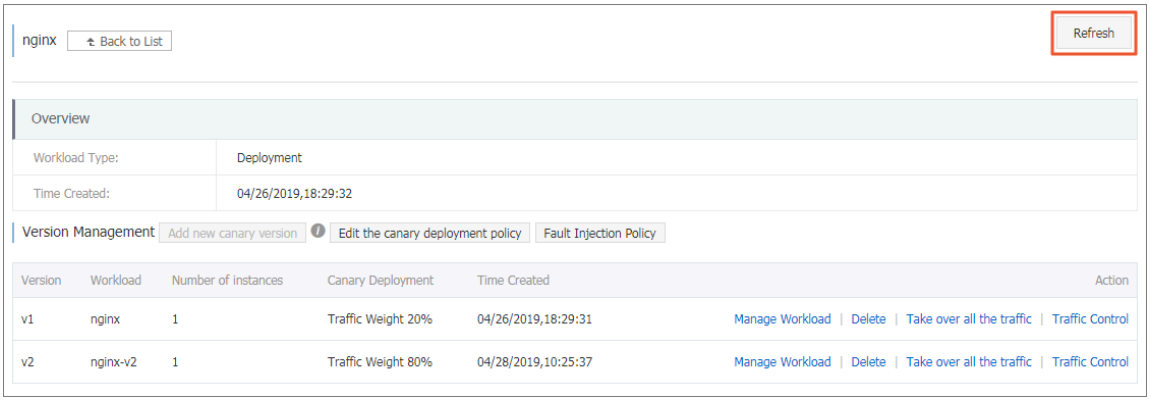
   a. On the page that displays the details of the target virtual service, click Edit the canary deployment policy.



   b. Change the traffic distribution ratio for the two versions of the target virtual services. For example, set Version v1 to `20 %`, and set Version v2 to `80 %`. Then, click OK.



   c. In the upper-right corner, click Refresh.



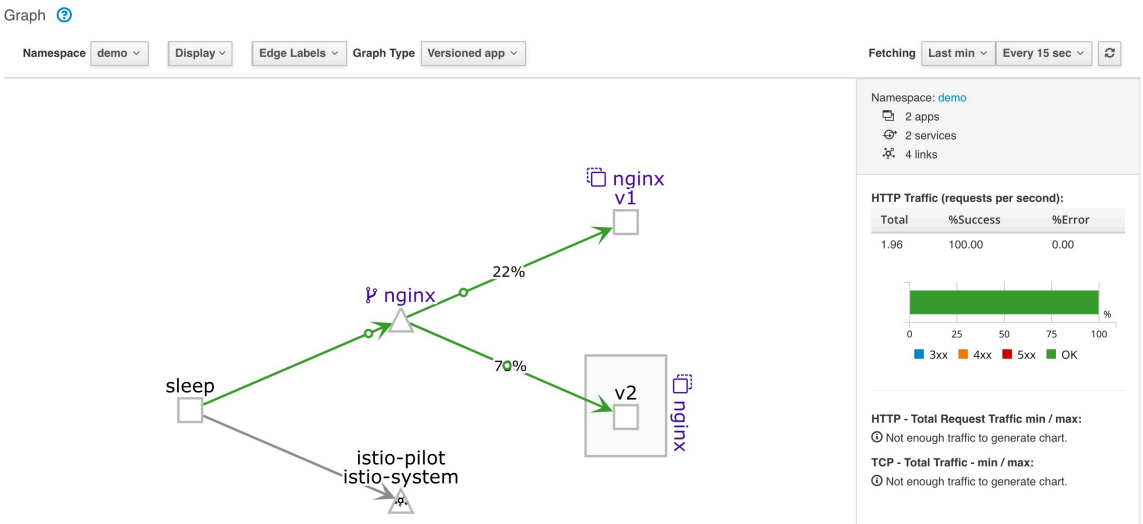
d. Run the `kubectl exec -it -n demo <podName> bash` command to log on to the pod where the `sleep` application runs, and then run the following commands to call the `nginx` service:

```
for i in `seq 1000`
do
curl -I http://nginx.demo:8080;
echo '';
sleep 1;
done;
```

If the service is called, the content shown in the following figure is displayed.

```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

e. In your browser, enter *http://localhost:20001* to log on to the Kiali console to check how the target service is called.



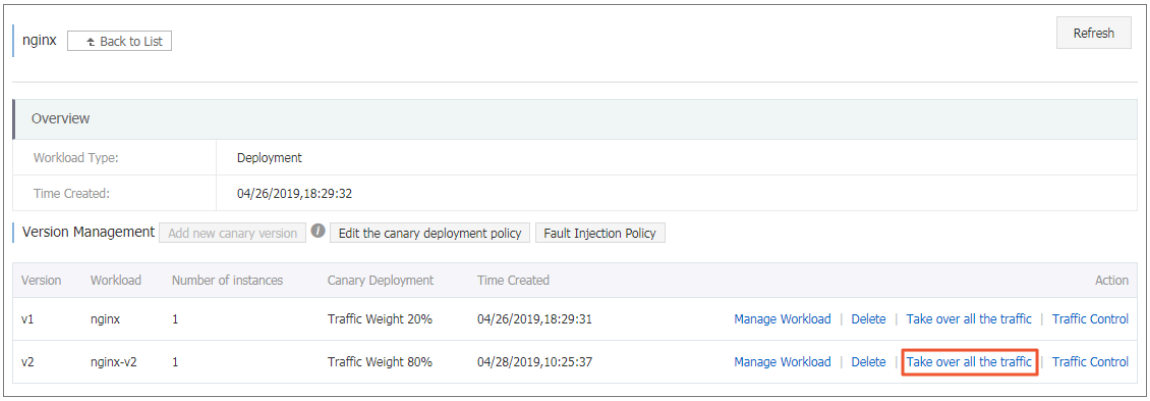
6. Use the latest version of the target service to take over all the traffic that is destined for the earlier version.

   a. Log on to the *Container Service console*.

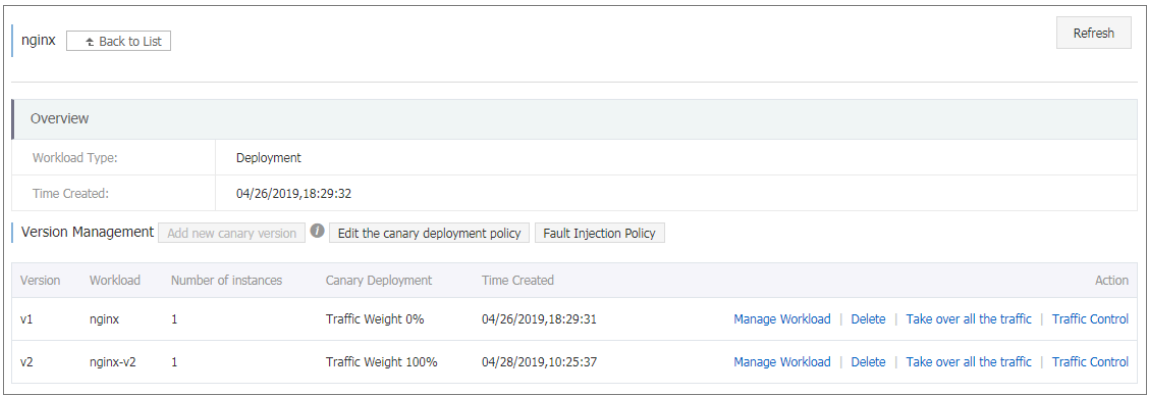
   b. In the left-side navigation pane, choose Service Mesh > Virtual Service.

   c. On the right of the target virtual service, click Manage.

   d. In the Version Management area, find v2, and click Take over all the traffic in the Action column.



   e. In the displayed dialog box, click Confirm. Then, in the Canary Deployment column of v2, Traffic Weight 100% is displayed.



   f. Run the `kubectl exec -it -n demo <podName> bash` command to log on to the pod where the `sleep` application runs, and then run the following commands to call the `nginx` service:

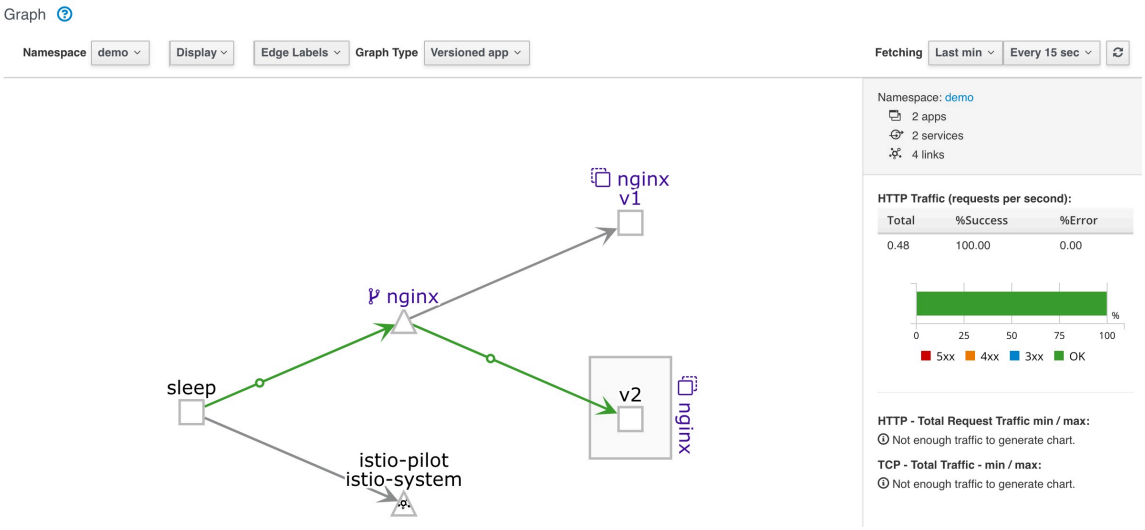
```
for i in `seq 1000`
do
curl -I http://nginx.demo:8080;
echo '';
sleep 1;
```

```
done ;
```

If the service is called, the content shown in the following figure is displayed.

```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

g. In your browser, enter `http :// localhost : 20001` to log on to the Kiali console to check how the target service is called.



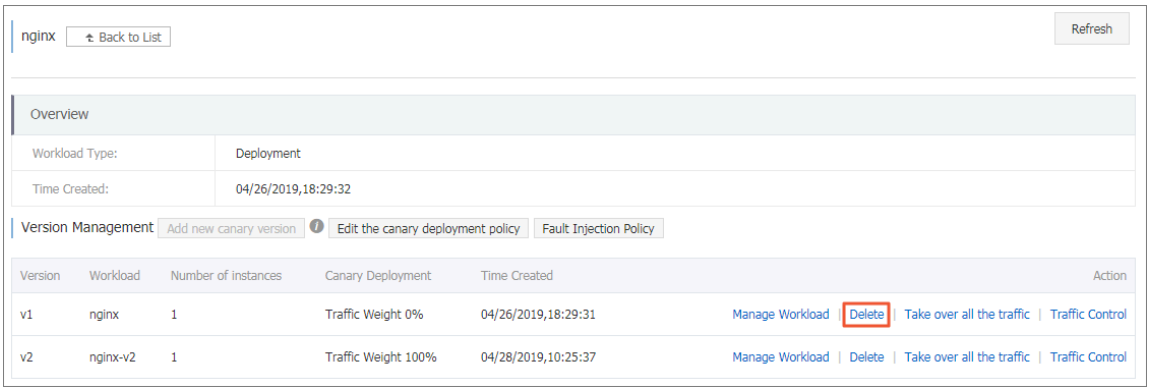
7. Delete the earlier version of the target service.

   a. Log on to the *Container Service console*.

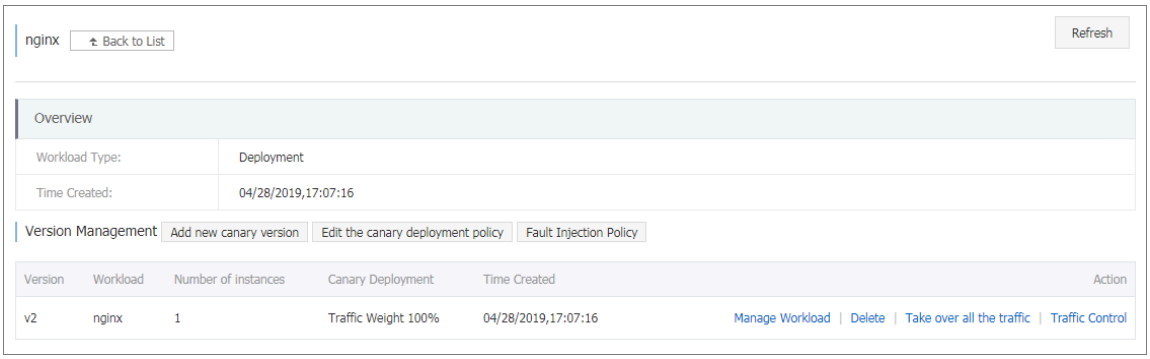
   b. In the left-side navigation pane under Container Service-Kubernetes, choose Service Mesh > Virtual Service.

   c. On the right of the target virtual service, click Manage.

   d. In the Version Management area, find v1, and click Delete in the Action column.



Then, only the latest version of the target service operates.



Note:

If you want to release one more version of the target service by using the Canary deployment method, you can repeat the preceding steps.

6 Monitoring

6.1 Use ARMS to monitor an application running in a Kubernetes cluster

This topic describes how to use Application Real-Time Monitoring Service (ARMS) to monitor an application running in Alibaba Cloud Container Service for Kubernetes.

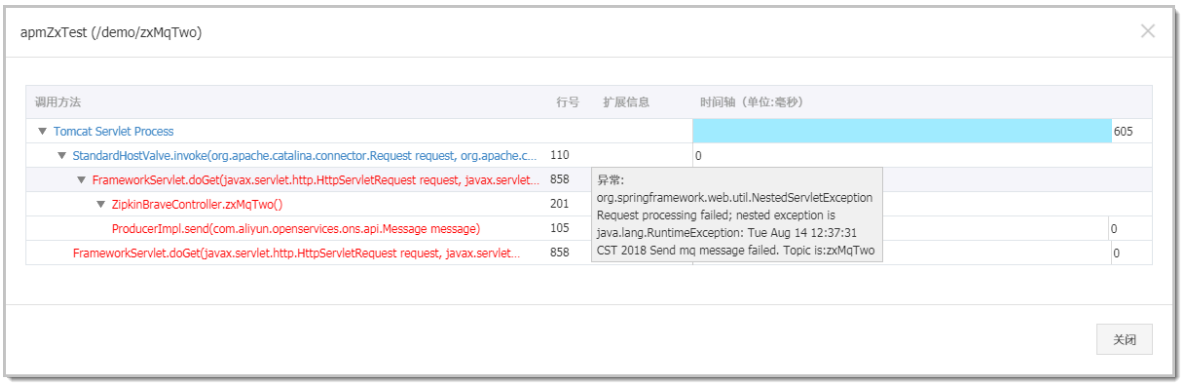
Overview about ARMS

ARMS is a Java application performance management (APM) monitoring product developed

by Alibaba Cloud. If you use ARMS to monitor a Java application, you only need to mount a probe in the application startup script without modifying any code. By comprehensively monitoring the application, the probe helps you quickly locate faulty and slow interfaces, reproduce parameter calling, detect memory leaks, and discover system bottlenecks, more efficiently diagnosing problems online. For more information, see *ARMS*.

ARMS monitors an application as follows:

- Automatically discovers the application topology
- Automatically discovers and monitors interfaces
- Captures exception transactions and slow transactions, and provides SQL analysis
- Provides a Java exception report
- Provides trace-based transaction snapshot queries



- Provides multidimensional ad-hoc troubleshooting, including multidimensional trace searches and exception trace searches

- Integrates PaaS platforms

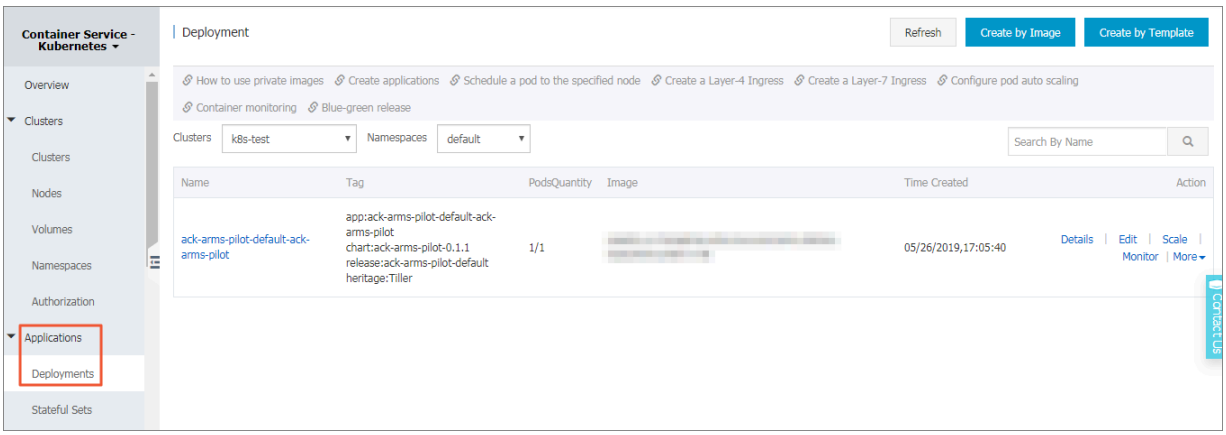
Prerequisites

- A Kubernetes cluster is created. For more information, see *Create a Kubernetes cluster*.
- ARMS is activated. For more information, see *Activate ARMS*.

Install ARMS components

1. Log on to the *Container Service console*.
2. In the left-side navigation pane, choose Store > App Catalog. On the App Catalog page, select ack-arms-pilot.
3. On the App Catalog - ack-arms-pilot page, click DEPLOY.

In the left-side navigation pane, choose Applications > Deployments, and then select the target cluster and namespace. An application named `ack-arms-pilot-default-ack-arms-pilot` is displayed on the page.



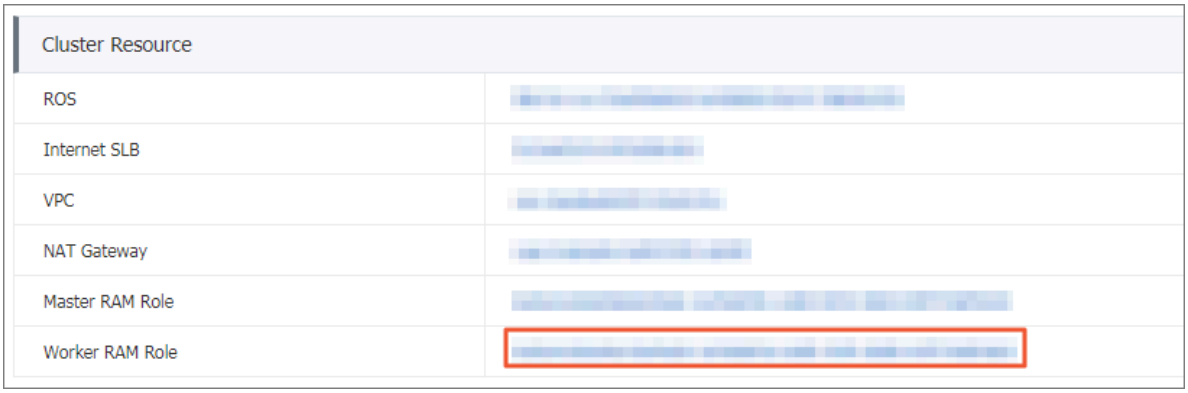
Grant permission to use ARMS

1. In the left-side navigation pane under Container Service-Kubernetes, choose Clusters > Clusters.

> **Note:**
> To perform the operations required in this section, you must use the primary account to log on to the Container Service console.

2. Click the target cluster name to view the cluster details.

3. In the Cluster Resources area, click Worker RAM Role.



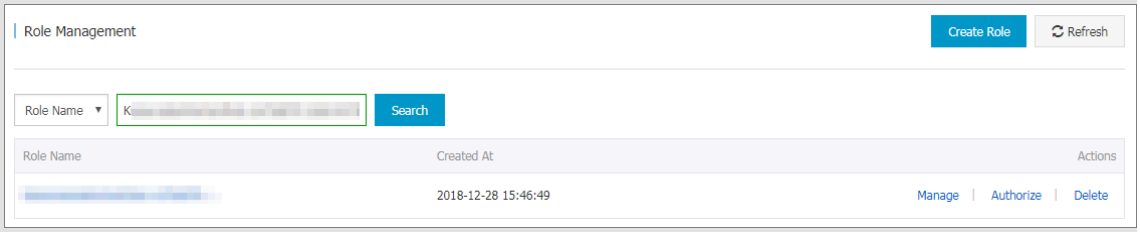
Note:

This topic uses the latest version of the RAM console.

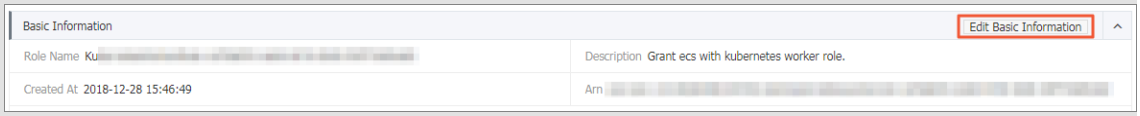
If you use an earlier version of the RAM console, you can modify the target policy document by using one of the following methods:

Method 1

a. In the left-side navigation pane, click Roles, and then enter the Worker RAM Role name in the Role Name box. Click the target Role Name.

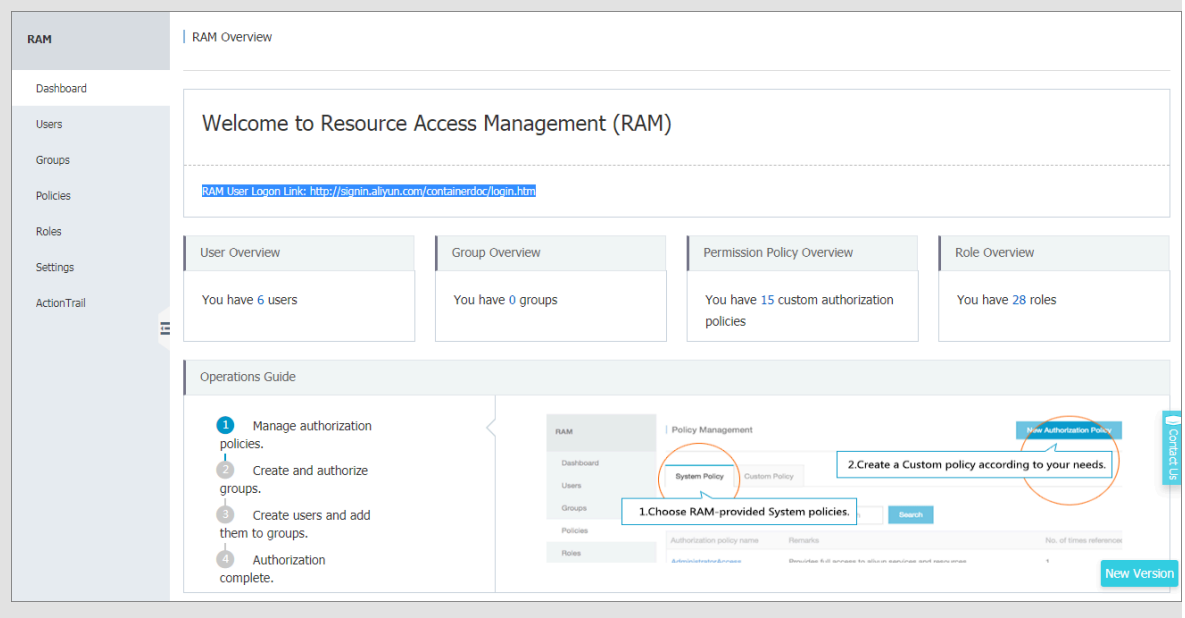


b. In the Basic Information area, click Edit Basic Information in the upper-right corner.

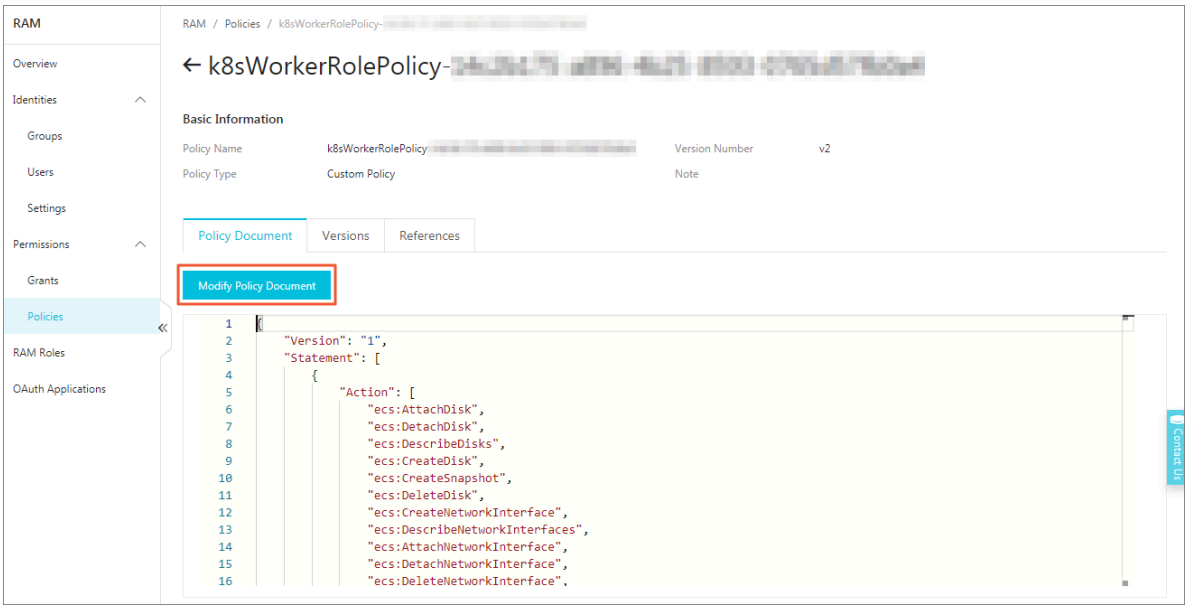


Method 2

In the lower-right corner of the RAM dashboard page, click New Version to switch to the latest version of the RAM console. In the Container Service console, click Worker RAM Role to log on to the RAM console.



4. On the RAM Roles page, click the policy name on the Permissions tab page.
5. On the Policies page, click Modify Policy Document on the Policy Document tab page.



6. In the Policy Document area, add the following fields and then click OK.

```
{
    " Action ": " arms :*",
    " Resource ": "*",
    " Effect ": " Allow "
```

```
}
```

Deploy ARMS monitoring for an application

In a YAML file used to create the target deployment, add the following `annotation s` to deploy ARMS application monitoring.

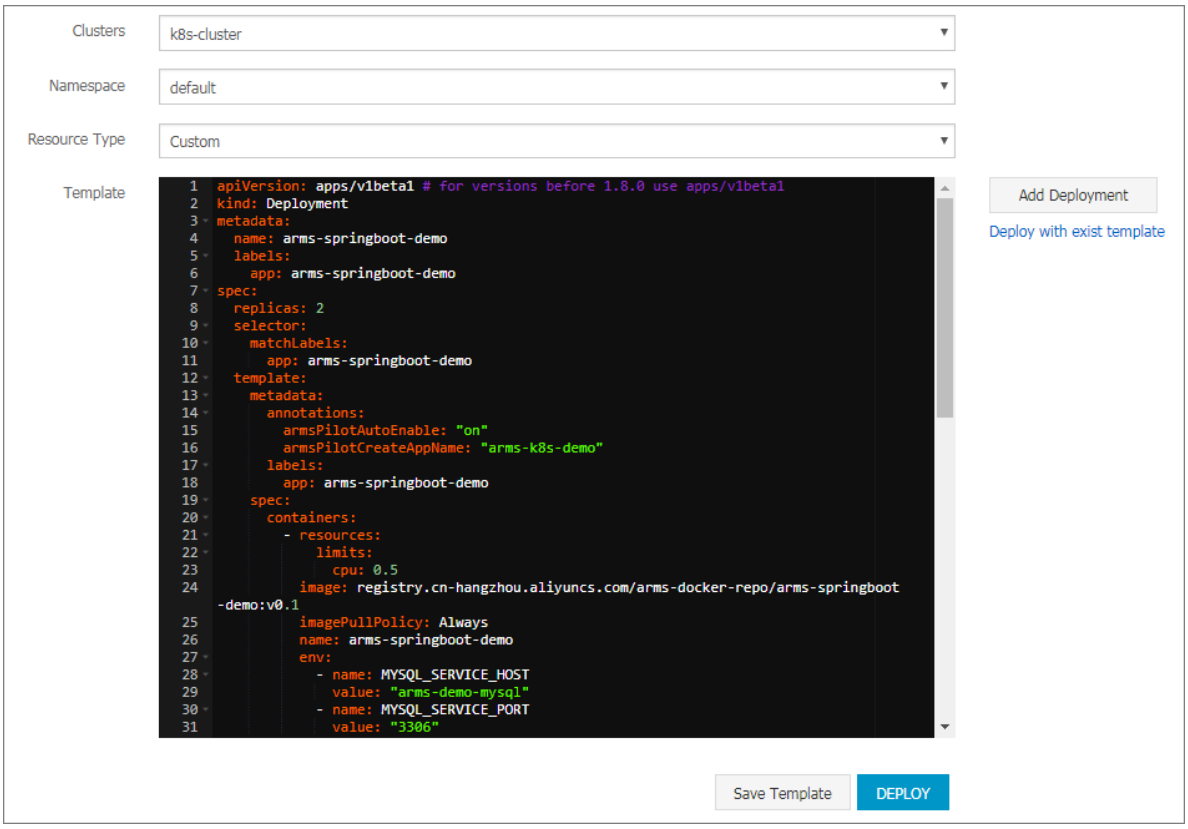
```
annotation  s :
  armsPilotA  utoEnable : " on "
```

```
armsPilotCreateAppName: "< your-deployment-name >"
```

Note:

- In the YAML file, you must add `annotations` under `metadata` of `template` in the `spec` field.
- The value of `armsPilotCreateAppName` is the name of the application monitored by ARMS.

1. In the left-side navigation pane of the Container Service console, choose Applications > Deployments.
2. In the upper-right corner, click Create by Template.
3. Select the target cluster and namespace to create a deployment.




```
apiVersion: apps/v1beta1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: arms-springboot-demo
  labels:
    app: arms-springboot-demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: arms-springboot-demo
  template:
    metadata:
      annotations:
        armsPilotAutoEnable: "on"
        armsPilotCreateAppName: "arms-k8s-demo"
      labels:
        app: arms-springboot-demo
    spec:
      containers:
        - resources:
            limits:
              cpu: 0.5
          image: registry.cn-hangzhou.aliyuncs.com/arms-docker-repo/arms-springboot
-demo:v0.1
          imagePullPolicy: Always
          name: arms-springboot-demo
          env:
            - name: MYSQL_SERVICE_HOST
              value: "arms-demo-mysql"
            - name: MYSQL_SERVICE_PORT
              value: "3306"
```

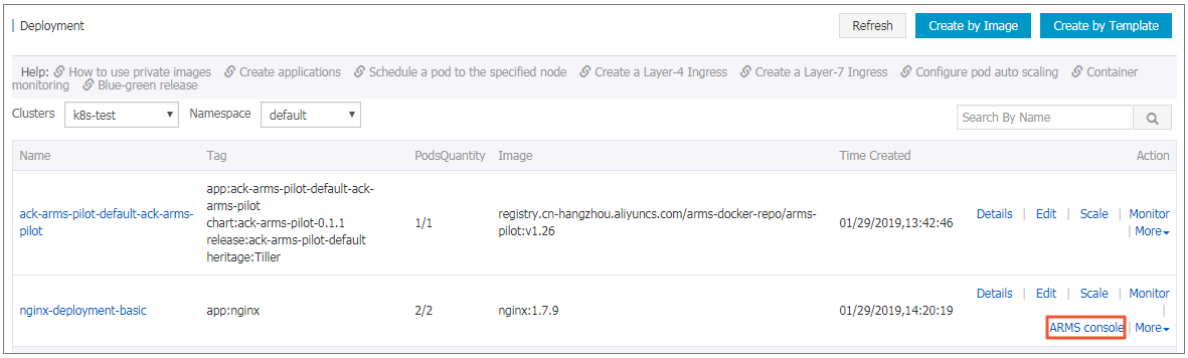
```
    metadata :
      annotation  s :
        armsPilotA  utoEnable : " on "
        armsPilotC  reateAppNa  me : " arms - k8s - demo "
      labels :
        app :  arms - springboot - demo
    spec :
      containers :
       -  resources :
            limits :
              cpu :  0 . 5
          image :  registry . cn - hangzhou . aliyuncs . com / arms
- docker - repo / arms - springboot - demo : v0 . 1
          imagePullP  olicy :  Always
          name :  arms - springboot - demo
          env :
           -  name :  MYSQL_SERV  ICE_HOST
              value : " arms - demo - mysql "
           -  name :  MYSQL_SERV  ICE_PORT
              value : " 3306 "
---
apiVersion :  apps / v1beta1  #  for  versions  before  1 . 8 .
0  use  apps / v1beta1
kind :  Deployment
metadata :
  name :  arms - demo - mysql
  labels :
    app :  mysql
spec :
  replicas :  1
  selector :
    matchLabel  s :
      app :  mysql
  template :
    metadata :
      labels :
        app :  mysql
    spec :
      containers :
       -  resources :
            limits :
              cpu :  0 . 5
          image :  registry . cn - hangzhou . aliyuncs . com / arms
- docker - repo / arms - demo - mysql : v0 . 1
          name :  mysql
          ports :
           -  containerP  ort :  3306
              name :  mysql
---
apiVersion :  v1
kind :  Service
metadata :
  labels :
    name :  mysql
  name :  arms - demo - mysql
spec :
  ports :
   #  the  port  that  this  service  should  serve  on
   -  name :  arms - mysql - svc
      port :  3306
      targetPort :  3306
 #  label  keys  and  values  that  must  match  in  order
  to  receive  traffic  for  this  service
  selector :
```

```
      app :  mysql
---
```

Verify the results

1. In the left-side navigation pane of the Container Service console, choose Applications > Deployments, and then select the target cluster and namespace to view the created deployment.
2. In the Action column of the target deployment, click ARMS console to log on to the ARMS console to view application details such as Application Overview, Interface Invocation, and other information.

> **Note:**
> If ARMS console is not displayed in the Action column, check whether you have granted Container Service permission to use ARMS. For more information, see *Grant permission to use ARMS*.



7 Operation and maintenance

7.1 Check a Kubernetes cluster to troubleshoot exceptions

This topic describes how to quickly check a Kubernetes cluster in the Container Service console to troubleshoot exceptions.

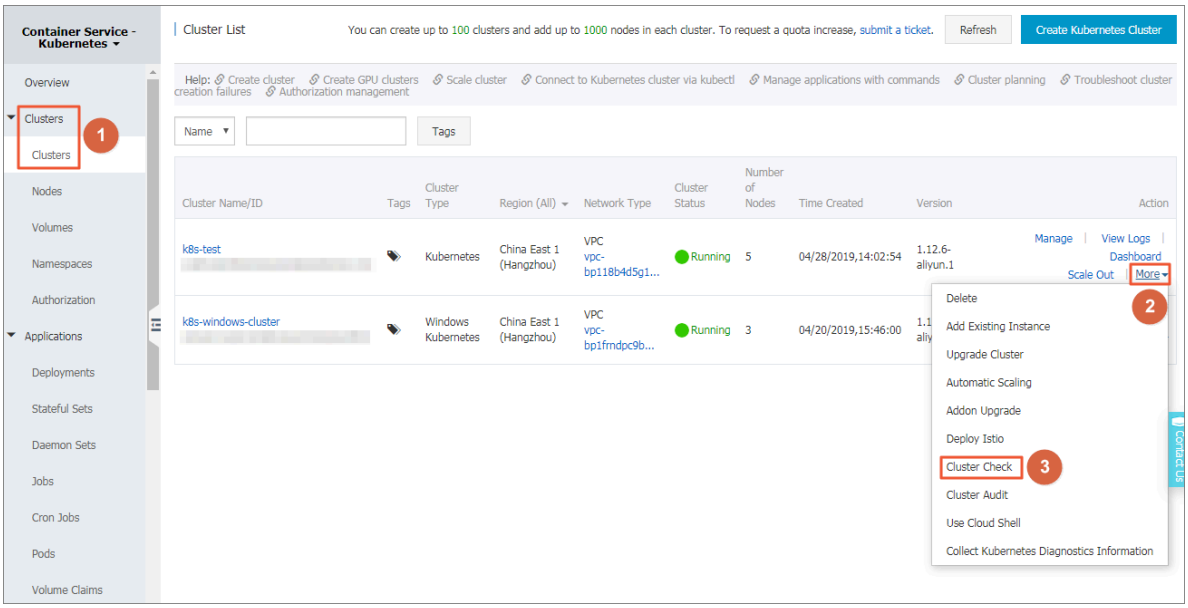
Prerequisites

- A Kubernetes cluster is created. For more information, see *Create a Kubernetes cluster*.
- The Kubernetes cluster is in the running status.

> **Note:**
> On the Cluster List page, you can check whether the Kubernetes cluster status is in the `Running` status.

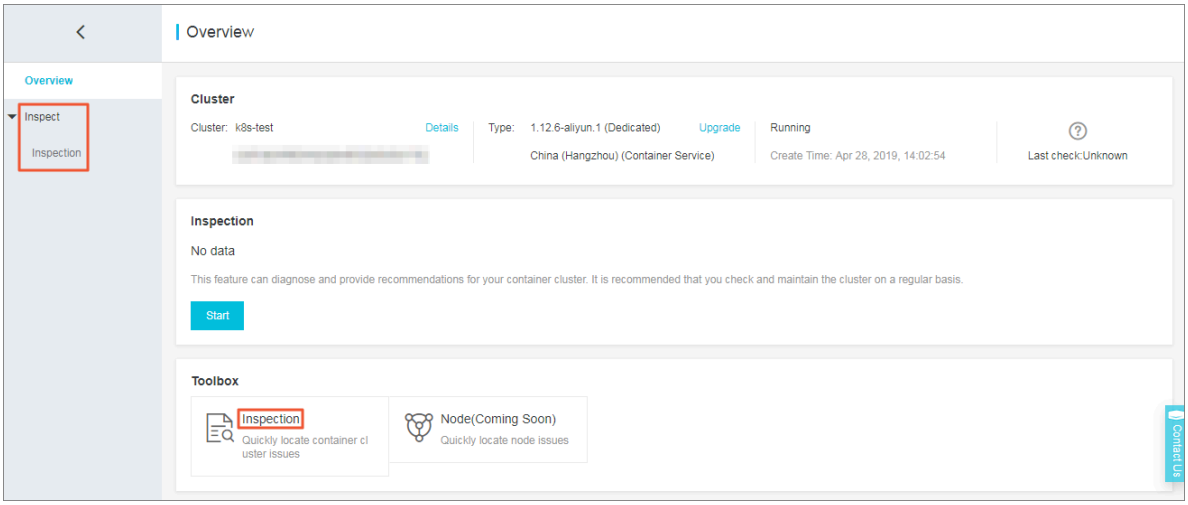
Procedure

1. Log on to the *Container Service console*.
2. In the left-side navigation pane under Container Service-Kubernetes, choose Clusters > Clusters. Find the target Kubernetes cluster, and choose More > Cluster Check in the action column.



3. In the left-side navigation pane, choose Inspect > Inspection.

> **Note:**
> You can also click Inspection in the toolbox area.



- Cluster: displays the cluster name, type, status, and the result of the latest check.
- Inspection: displays the corresponding time and results for the latest five cluster checks.

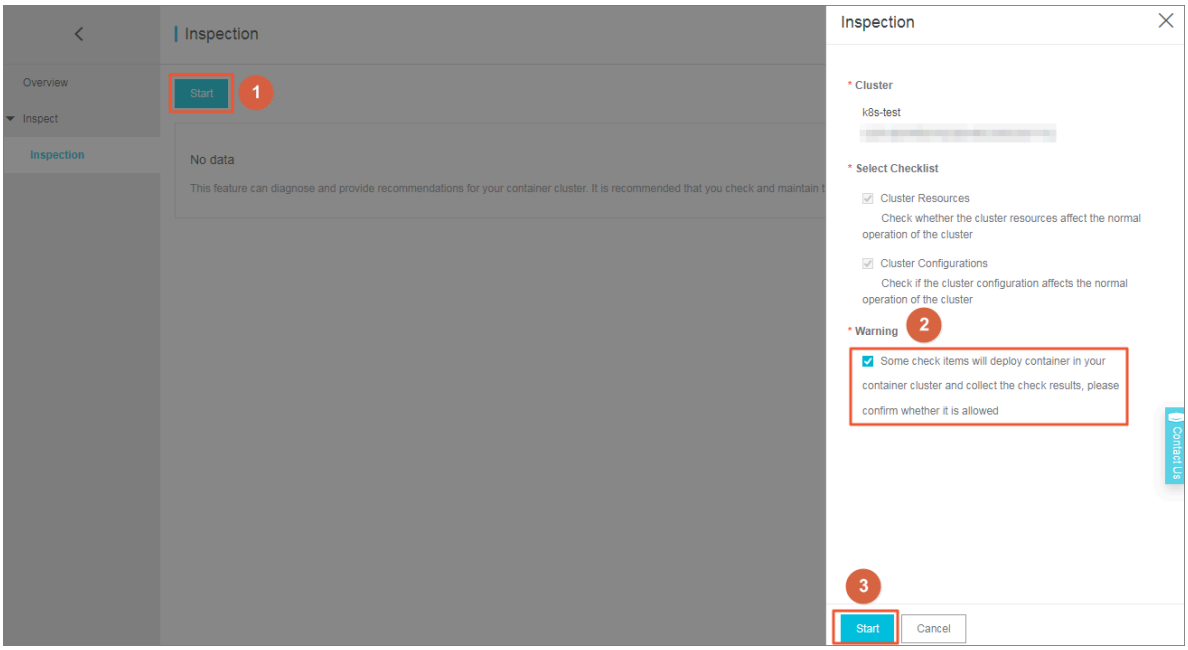
  > **Note:**
  > If you check a Kubernetes cluster for the first time, this area does not display any data except for the Start button. Clicking this button opens the the Inspection page where you can then enable cluster checking.

- Toolbox: displays the cluster check module.

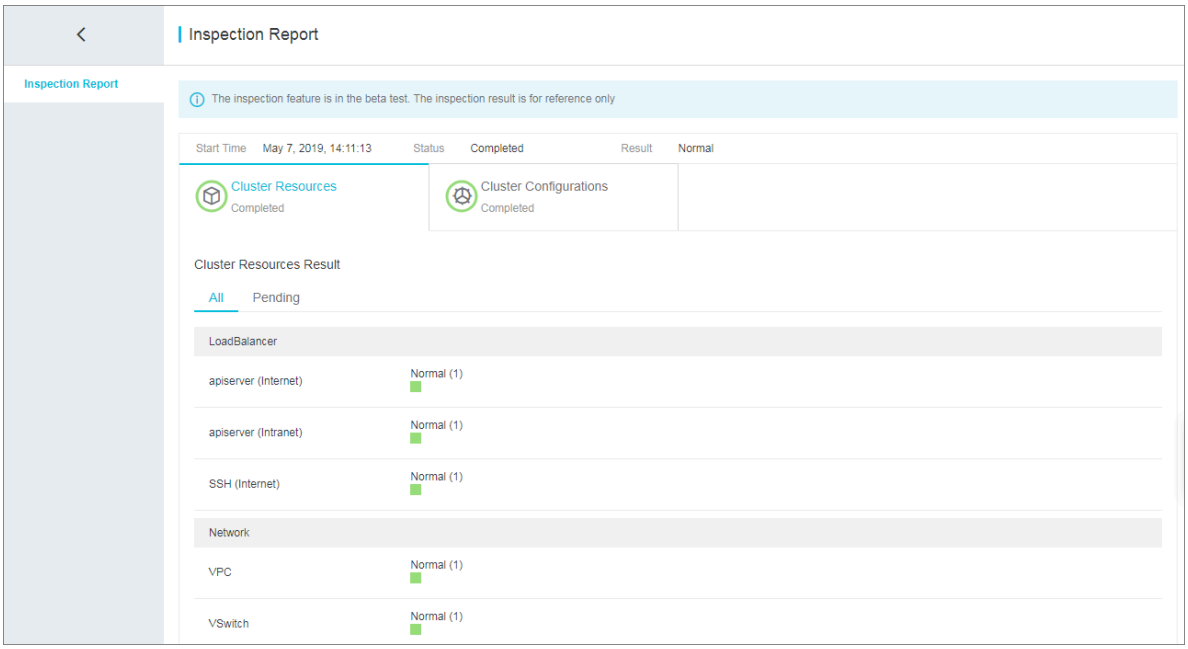
4. Click Start. Then, in the displayed page, select the Warning check box, and click Start.

> **Note:**

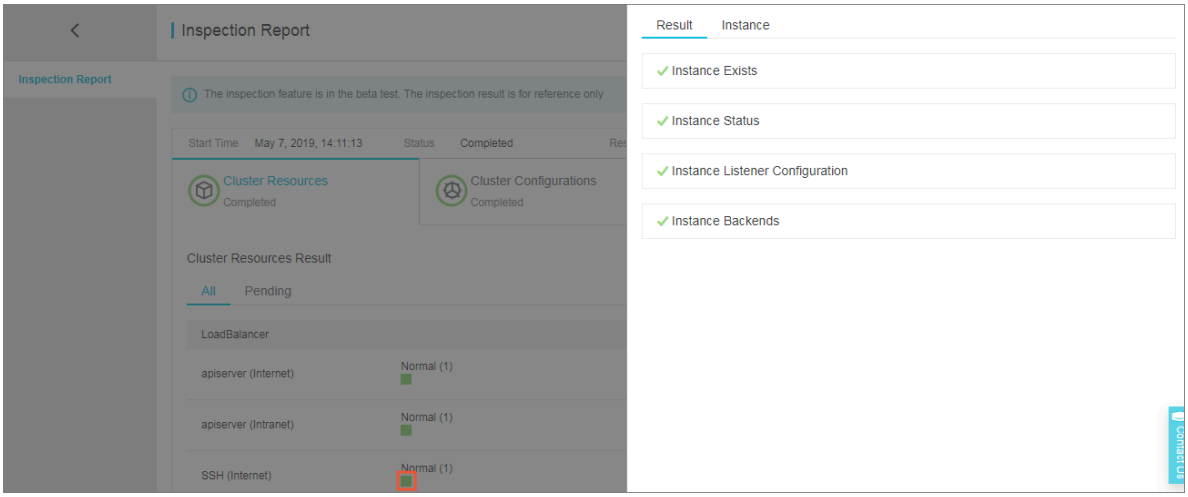
After you enable the cluster checks, the page displays the progress for the cluster checks in real time.



5. Click Details to view the results for the cluster resource and configuration checks.



6. Perform the action that corresponds to the result returned from the check.
   - Normal: No action is required.
   - Warning: Confirm the severity of the issue. If the issue may result in cluster exceptions, perform the appropriate actions to troubleshoot the issue.
   - Exception: Perform the required actions to troubleshoot the issue immediately.



Note:

- For each resource or configuration (which contains result and instance information) in the result of a check, you can click the corresponding box icon on the right to view the instance and result information.
- If a resource or configuration is not in the normal state, the result displays the checked items, severity of exceptions, and recommended solutions. You can perform actions to troubleshoot as needed.

8 Serverless

8.1 Use a GPU container instance

This topic describes how to use a GPU container instance by using an example in which Tensorflow is used to identify a picture. ACK serverless Kubernetes provides GPU container instances by integrating with Elastic Container Instance (ECI). As a part of this service, GPU container instances can be used in a serverless Kubernetes cluster or on a virtual node of a dedicated or managed Kubernetes cluster.

Billing

GPU and CPU resources provided by ACK serverless Kubernetes are charged with the Pay-As-You-Go billing method.

GPU instances provided by ACK serverless Kubernetes are charged the same as the ECS GPU instances. You are not charged additional fees.

Overview

In an ACK serverless Kubernetes cluster, you can easily create a pod to which a GPU is attached. Specifically, for the pod configurations, you must set the `annotation s` parameter to specify the GPU type that you want, and set the `limits` parameter to specify the number of GPU.

 **Note:**
A GPU is exclusive to one pod.

Prerequisites

Either a serverless Kubernetes cluster is created, or a virtual node is created for a Kubernetes cluster. For more information, see *Create a serverless Kubernetes cluster* or *Deploy a virtual node*.

Procedure

In the following steps, Tensorflow is used in a serverless Kubernetes cluster to identify the following picture.

1. Log on to the *Container Service console*.
2. In the left-side navigation pane under Container Service-Kubernetes, choose Applications > Deployments. Then, in the upper-right corner, click Create by Template.
3. Select the target cluster and namespace, select an example template or customize a template, and then click DEPLOY.

   You can use the following YAML template to create a pod:

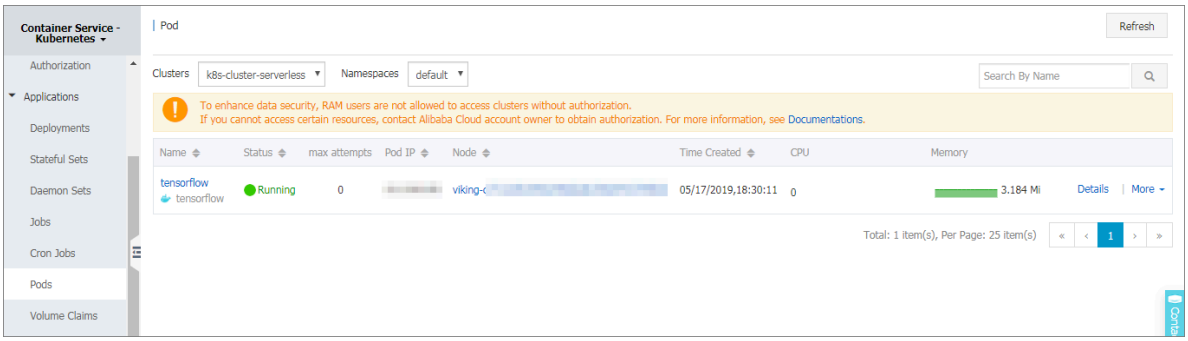
   > **Note:**
   > In this pod template, the GPU type is set to `P4` , and the number of GPU is set to 1.

   ```
   apiVersion :  v1
   kind :  Pod
   metadata :
     name :  tensorflow
     annotation  s :
       k8s . aliyun . com / eci - gpu - type  : " P4 "
   spec :
     containers :
    -  image :  registry - vpc . cn - hangzhou . aliyuncs . com / ack
   - serverless / tensorflow
       name :  tensorflow
       command :
      - " sh "
      - "- c "
      - " python   models / tutorials / image / imagenet / classify_i
   mage . py "
       resources :
         limits :
           nvidia . com / gpu : " 1 "
     restartPol  icy :  OnFailure
   ```

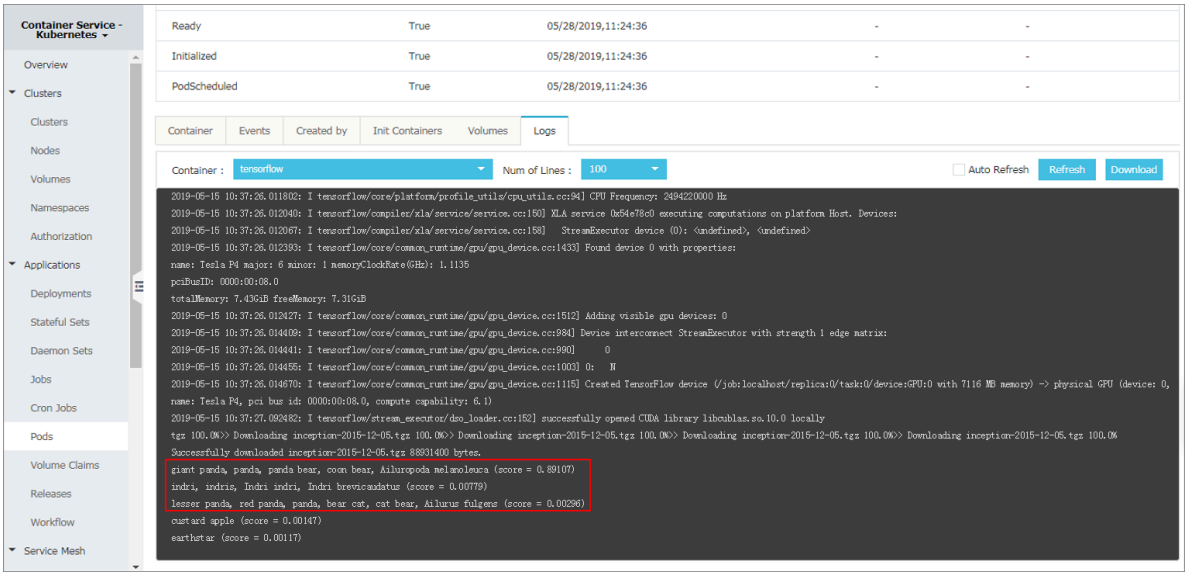
4. In the left-side navigation pane under Container Service-Kubernetes, choose Applications > Pods to verify that the pod is created.

   > **Note:**

It may take a few minutes to create a pod. If you the pod is not displayed, you can refresh the page.



5. Click the target pod, and click the Logs tab to verify that the picture is identified according to the logs marked in the red frame.



If you want to use a GPU container instance on a virtual node of a dedicated or managed Kubernetes cluster, follow these steps:

Note:
A GPU container instance used on a virtual node supports a larger variety of deep learning platforms such as Kubeflow, Arena, or other customized CRD.

1. Create a pod on the virtual node or in a namespace to which the `virtual-node-affinity-injection=enabled` label tag is added. For more information, see *Deploy a virtual node*.
2. Replace the template used in step *3* with the following template:

```
apiVersion: v1
kind: Pod
metadata:
```

```
  name :  tensorflow
  annotation  s :
    k8s . aliyun . com / eci - gpu - type  : " P4 "
spec :
  containers :
 -  image :  registry - vpc . cn - hangzhou . aliyuncs . com / ack
- serverless / tensorflow
    name :  tensorflow
    command :
   - " sh "
   - "- c "
   - " python   models / tutorials / image / imagenet / classify_i
mage . py "
    resources :
      limits :
        nvidia . com / gpu : " 1 "
  restartPol  icy :  OnFailure
  nodeName :  virtual - kubelet
```

8.2 Associate an EIP address with a pod

This topic describes how to associate an EIP address with a pod in a serverless Kubernetes cluster or with a pod on a virtual node of a dedicated or managed Kubernetes cluster.

Benefits

Associating an EIP address with a target pod offers the following benefits:

- You can easily deploy an application that uses serverless containers.
- Services provided by the application can be more quickly accessed.
- Your pod can access the Internet without the need of a VPC NAT gateway.
- Your target application can be accessed through the Internet without the need of an SLB instance.

Prerequisites

- Either a serverless Kubernetes cluster is created, or a virtual node is created in a dedicated or managed Kubernetes cluster. For more information, see *Create a serverless Kubernetes cluster* or *Deploy a virtual node*.
- The corresponding port for the target application that you want to run is enabled in the security group of the target Kubernetes cluster.

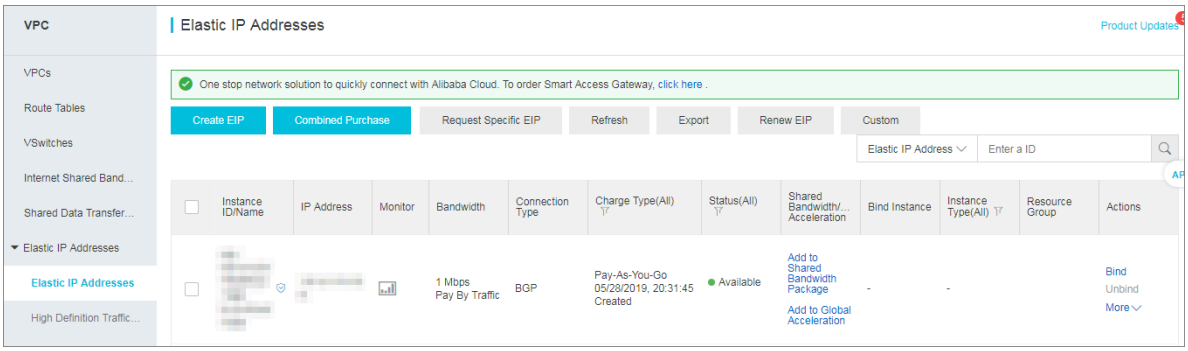
Note:

The Nginx application is used as an example. Therefore, port 80 must be enabled in the security group of the target Kubernetes cluster.

Procedure

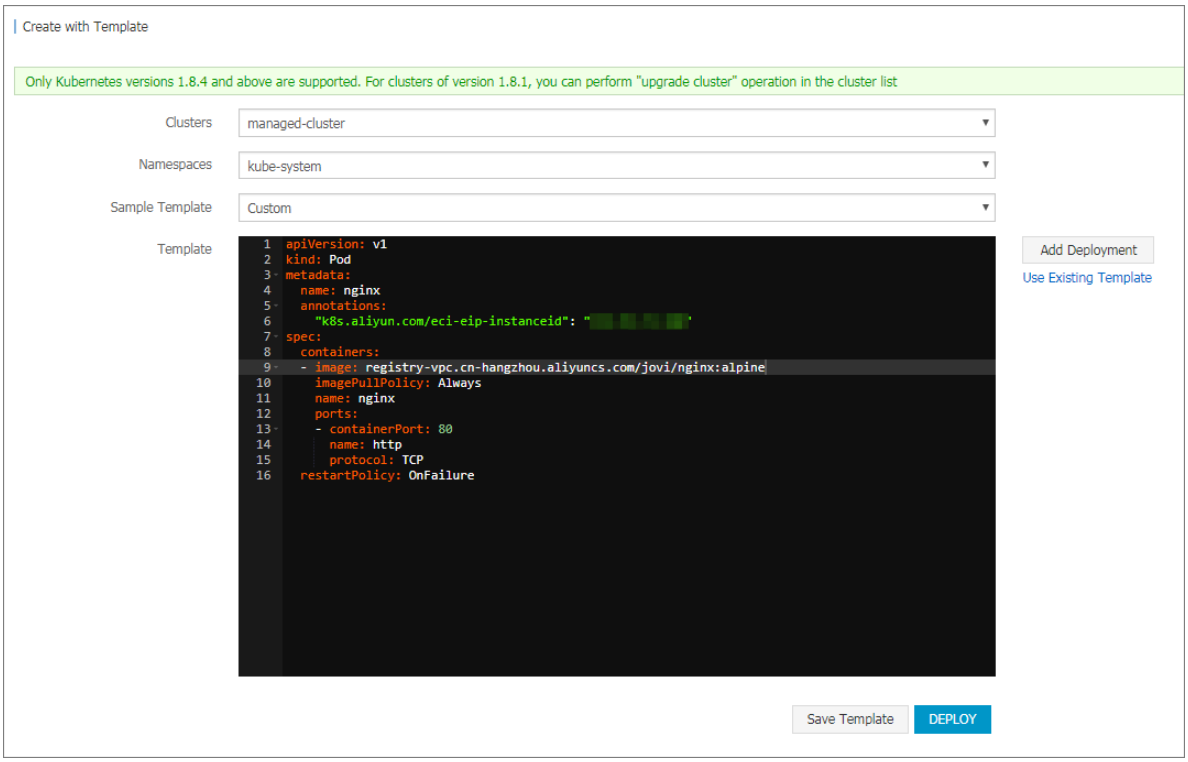
1. Log on to the VPC console to create an EIP address. For more information, see *Create an EIP*.

   > **Note:**
   > You must create an EIP address in the same region where the target Kubernetes cluster is located.

2. Log on to the *Container Service console*.
3. In the left-side navigation pane under Container Service-Kubernetes, choose Applications > Deployments.
4. In the upper-right corner, click Create by Template.

5. Select the target cluster and namespace, select an example template or customize a template, and then click DEPLOY.



You can use the following YAML template to create a pod:

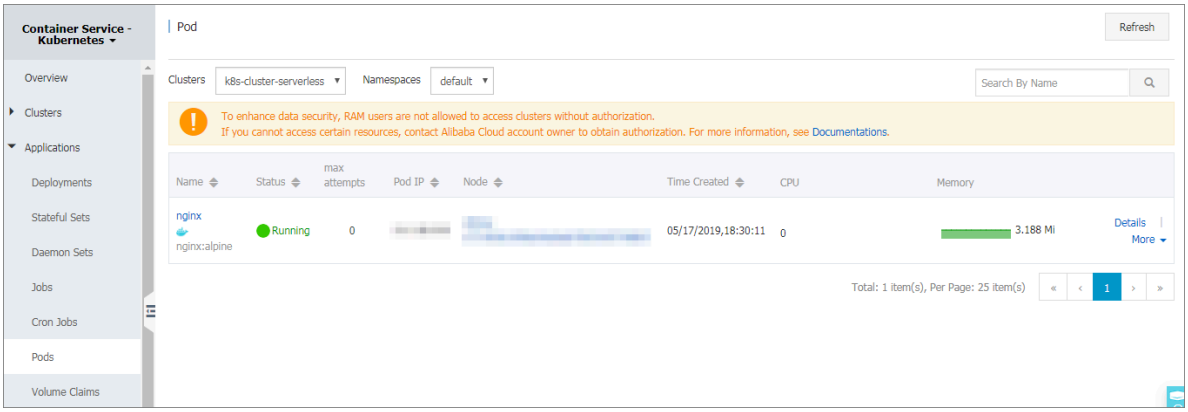
Note:

You must replace `< youreipIns tanceId >` with the EIP ID obtained in step 1.

```
apiVersion : v1
kind : Pod
metadata :
  name : nginx
  annotation s :
   " k8s . aliyun . com / eci - eip - instanceid ": "< youreipIns
tanceId >"
spec :
  containers :
 - image : registry - vpc . cn - hangzhou . aliyuncs . com / jovi
/ nginx : alpine
    imagePullP olicy : Always
    name : nginx
    ports :
   - containerP ort : 80
      name : http
      protocol : TCP
```

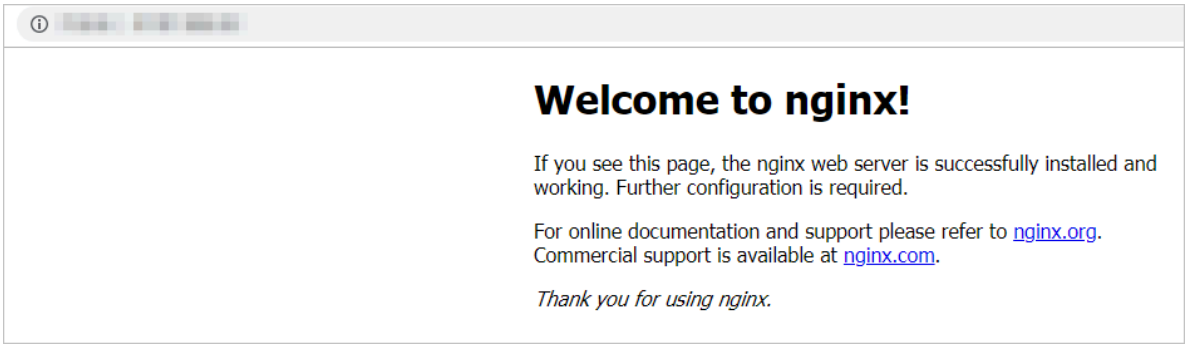
```
    restartPol  icy :  OnFailure
```

6. In the left-side navigation pane under Container Service-Kubernetes, choose Applications > Pods to view the pod status.



7. In your browser, enter the URL *http :// ip address* to access the welcome page of Nginx.

> **Note:**
> For this URL, you must enter the IP address obtained in step 1.



9 Auto Scaling

9.1 Deploy an Ingress application on a virtual node

This topic describes how to deploy an Ingress application on a virtual node of a Kubernetes cluster. With the virtual node, the Kubernetes cluster can provide the application with greater computing capability without the need for a new node needs to be created for the cluster.

Prerequisites

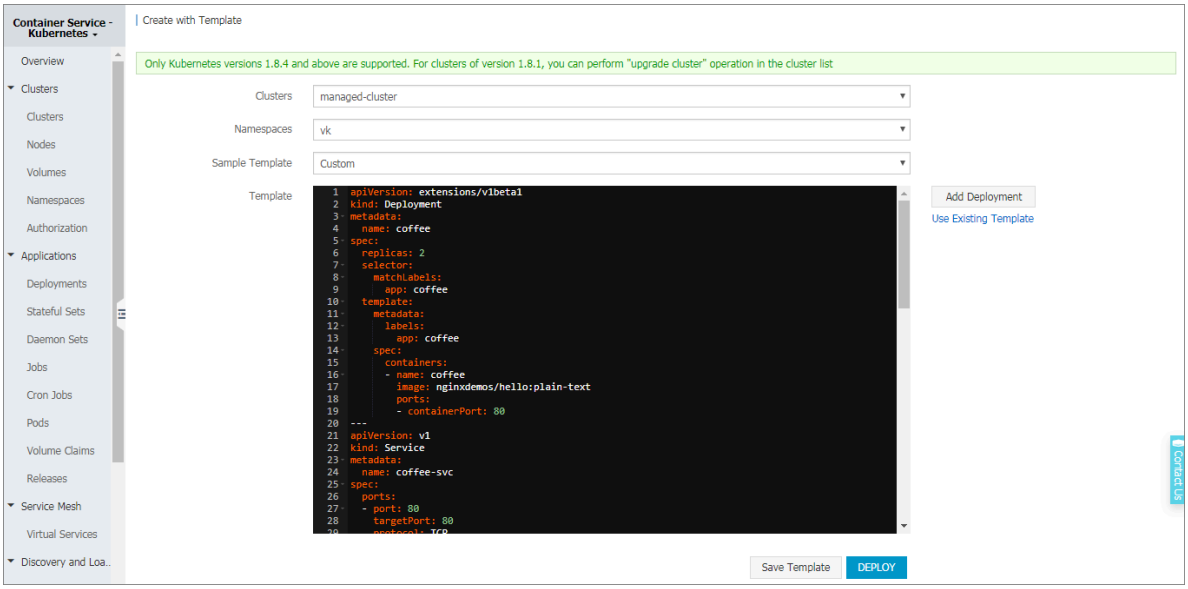
A virtual node is deployed in your target Kubernetes cluster. For more information, see *Deploy a virtual node*.

The tag `virtual-node-affinity-injection: enabled` is added to the target namespace. For more information, see *Set a namespace tag to create a pod*.

Procedure

1. Log on to the *Container Service console*.
2. In the left-side navigation pane under Container Service-Kubernetes, choose Applications > Deployments.
3. In the upper-right corner, click Create by Template.

4. Select the target cluster and namespace, select an example template or customize a template, and then click DEPLOY.



You can use the following template to customize a YAML template to create an Ingress application:

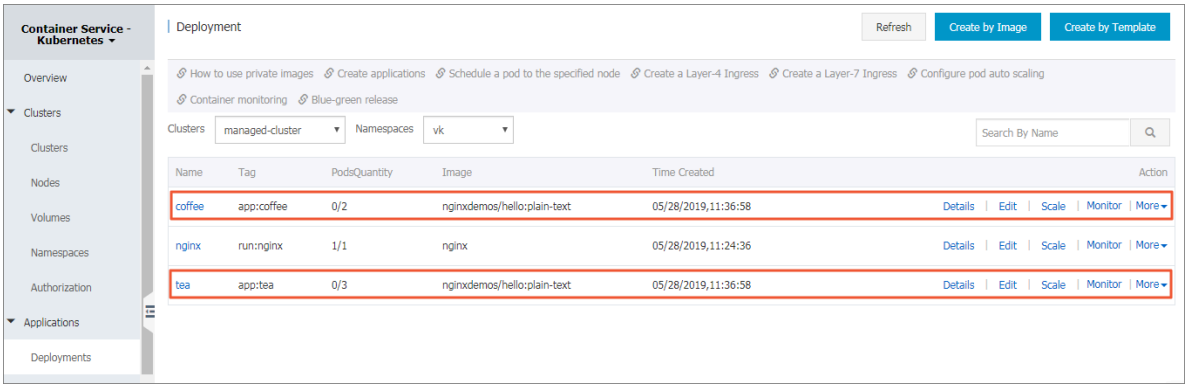
```
apiVersion : extensions / v1beta1
kind : Deployment
metadata :
  name : coffee
spec :
  replicas : 2
  selector :
    matchLabel s :
      app : coffee
  template :
    metadata :
      labels :
        app : coffee
    spec :
      containers :
      - name : coffee
        image : nginxdemos / hello : plain - text
        ports :
        - containerP ort : 80
---
apiVersion : v1
kind : Service
metadata :
  name : coffee - svc
spec :
  ports :
  - port : 80
    targetPort : 80
    protocol : TCP
  selector :
    app : coffee
  clusterIP : None
---
apiVersion : extensions / v1beta1
```

```
 kind :  Deployment
 metadata :
   name :  tea
 spec :
   replicas :  3
   selector :
     matchLabel  s :
       app :  tea
   template :
     metadata :
       labels :
         app :  tea
     spec :
       containers :
      -  name :  tea
         image :  nginxdemos / hello : plain - text
         ports :
        -  containerP  ort :  80
---
 apiVersion :  v1
 kind :  Service
 metadata :
   name :  tea - svc
   labels :
 spec :
   ports :
  -  port :  80
     targetPort :  80
     protocol :  TCP
   selector :
     app :  tea
   clusterIP :  None
---
 apiVersion :  extensions / v1beta1
 kind :  Ingress
 metadata :
   name :  cafe - ingress
 spec :
   rules :
  -  host :  cafe . example . com
     http :
       paths :
      -  path : / tea
         backend :
           serviceNam  e :  tea - svc
           servicePor  t :  80
      -  path : / coffee
         backend :
           serviceNam  e :  coffee - svc
```

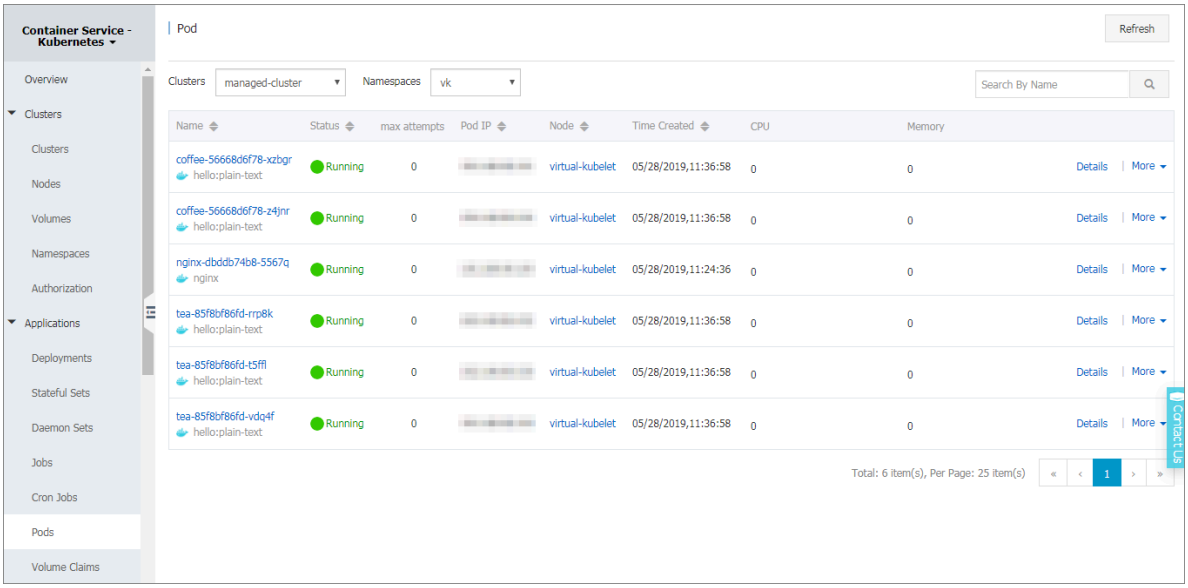
```
            servicePor  t :  80
```

Verify the results

- To verify that the `coffee` and `tea` deployments are created, follow these steps:
  1. In the left-side navigation pane under Container Service-Kubernetes, choose Applications > Deployments.
  2. Select the target cluster and namespace.

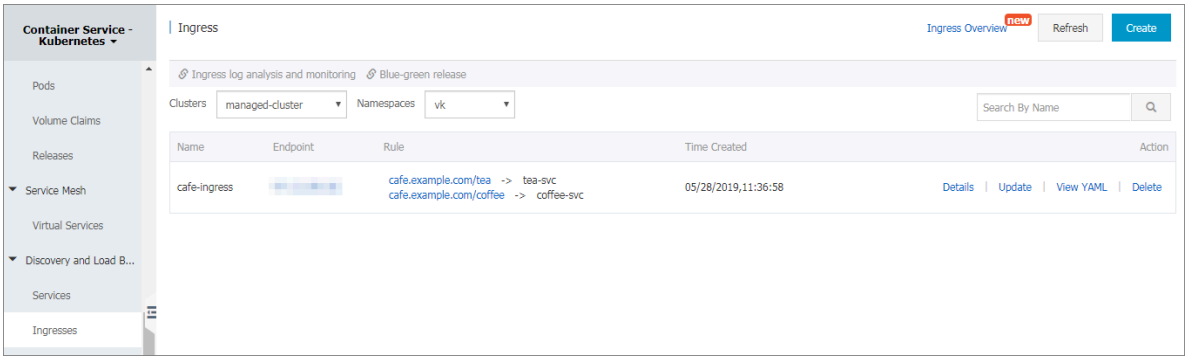


- To verify that all the pods of the Ingress application run on the virtual-kubelet node, follow these steps:
  1. In the left-side navigation pane under Container Service-Kubernetes, choose Applications > Pods.
  2. Select the target cluster and namespace.



- To verify that the target Ingress is created, follow these steps:

  1. In the left-side navigation pane under Container Service-Kubernetes, choose Discovery and Load Balancing > Ingresses.
  2. Select the target cluster and namespace.



- To verify that the created Ingress application can be accessed, run the following commands:

```
kubectl -n vk get ing
curl -H "Host:cafe.example.com" <EXTERNAL_IP>/tea
curl -H "Host:cafe.example.com" <EXTERNAL_IP>/coffee
```

```
shell@Alicloud:~$ kubectl -n vk get ing
NAME           HOSTS              ADDRESS          PORTS   AGE
cafe-ingress   cafe.example.com                    80      6m33s
shell@Alicloud:~$ curl -H "Host:cafe.example.com" /tea
Server address:
Server name: vk-tea-
Date: 28/May/2019:03:45:48 +0000
URI: /tea
Request ID: 0a78cffe790dd19ceea022255c7b8730
shell@Alicloud:~$ curl -H "Host:cafe.example.com" /coffee
Server address:
Server name: vk-coffee-
Date: 28/May/2019:03:46:52 +0000
URI: /coffee
Request ID: 977d8d6e38869d5a9e5603b6fdd2b5ed
shell@Alicloud:~$
```

10 DevOps

10.1 Deploy Jenkins in a Kubernetes cluster and perform a pipeline build

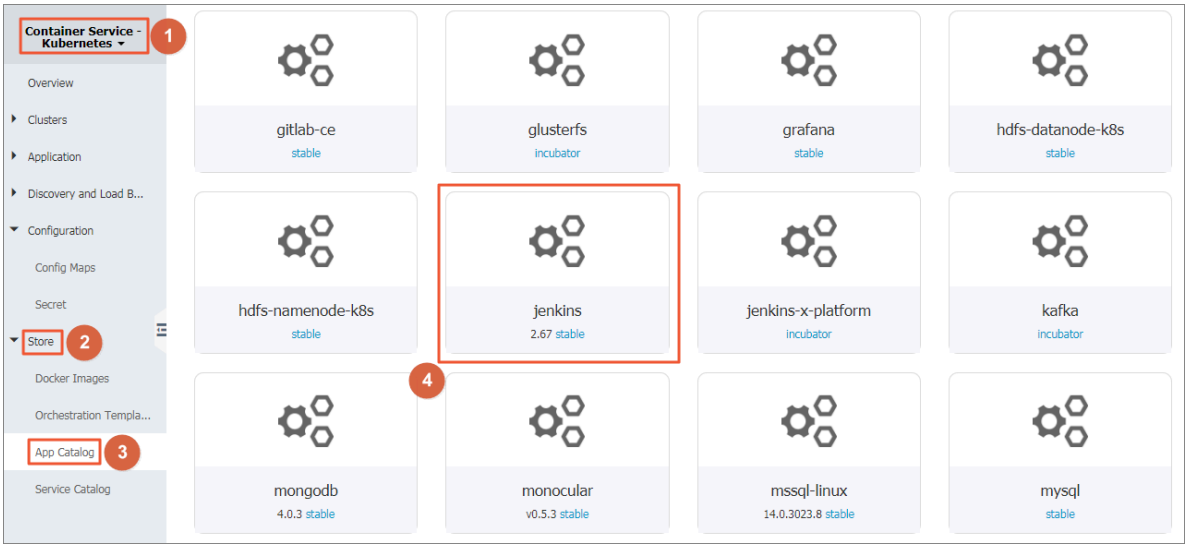
This topic describes how to deploy Jenkins, a continuous integration environment, in an Alibaba Cloud Kubernetes cluster, and how to perform an application pipeline build. The example in this topic details the pipeline build, including how to compile the source code of the application, build and push the application image, and deploy the application.

Prerequisites

You have created a Kubernetes cluster. For more information, see *Create a Kubernetes cluster*.

Deploy Jenkins

1. Log on to the *Container Service console*.
2. In the left-side navigation pane, choose Container Service-Kubernetes > Store > App Catalog. Then, click jenkins.



3. Click the Values tab.
4. Modify the `AdminPassw ord` field to set a password.

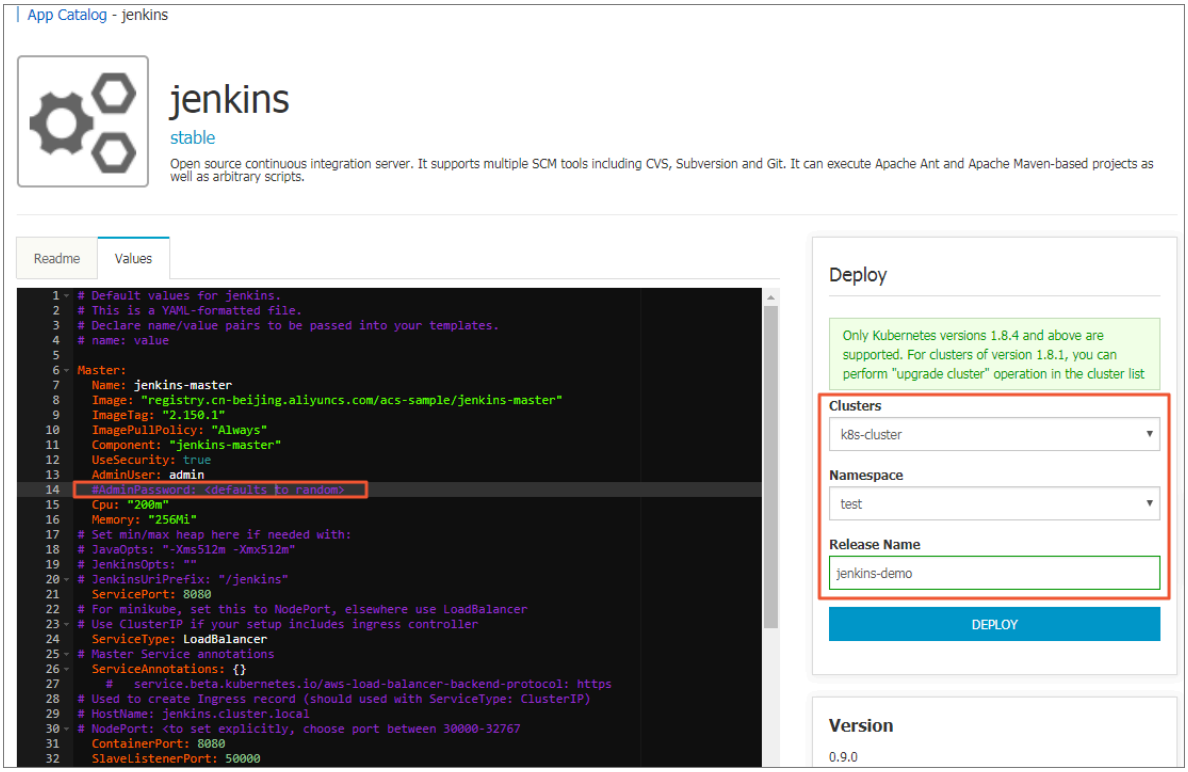
Note:

To ensure that your password takes effect, you must remove the pound sign (#) before `AdminPassw ord`

If you do not set any password, the system generates a password after Jenkins is deployed. You can run the following command to view the password:

```
$ printf $( kubectl get secret -- namespace ci jenkins - ci - jenkins - o jsonpath ="{. data . jenkins - admin - password }" | base64 -- decode ); echo
```

5. Select the target Cluster and Namespace, enter a Release Name, then click DEPLOY.

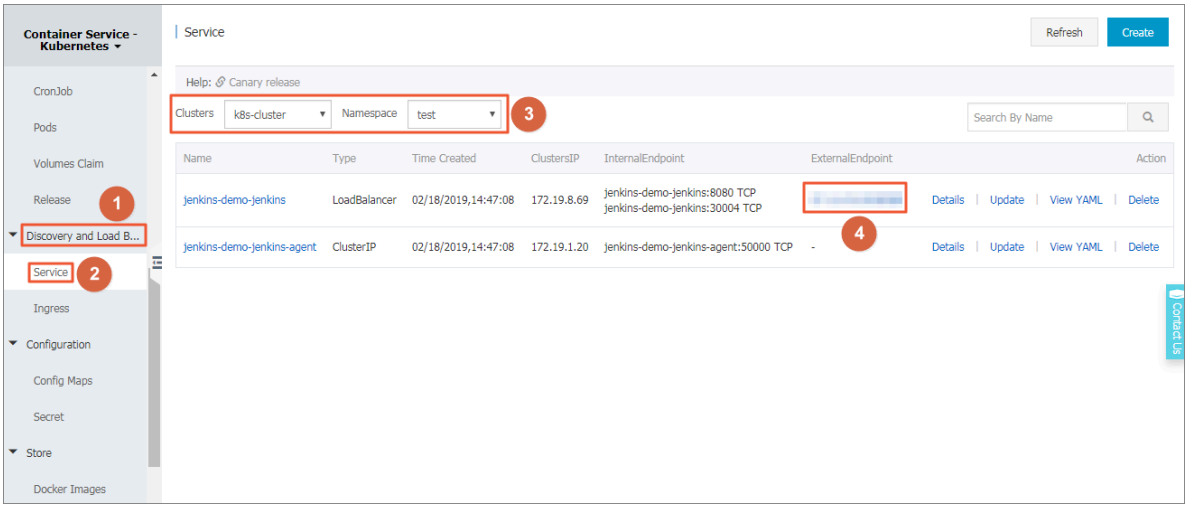


Note:

We recommend that you select a custom namespace or the default namespace. In this example, a custome namespace named test is selected.

6. In the left-side navigation pane, choose Discovery and Load Balancing > Service.
7. Select the cluster and the namespace used for deploying Jenkins.

8. Click the external endpoint of the Jenkins service to log on to Jenkins.



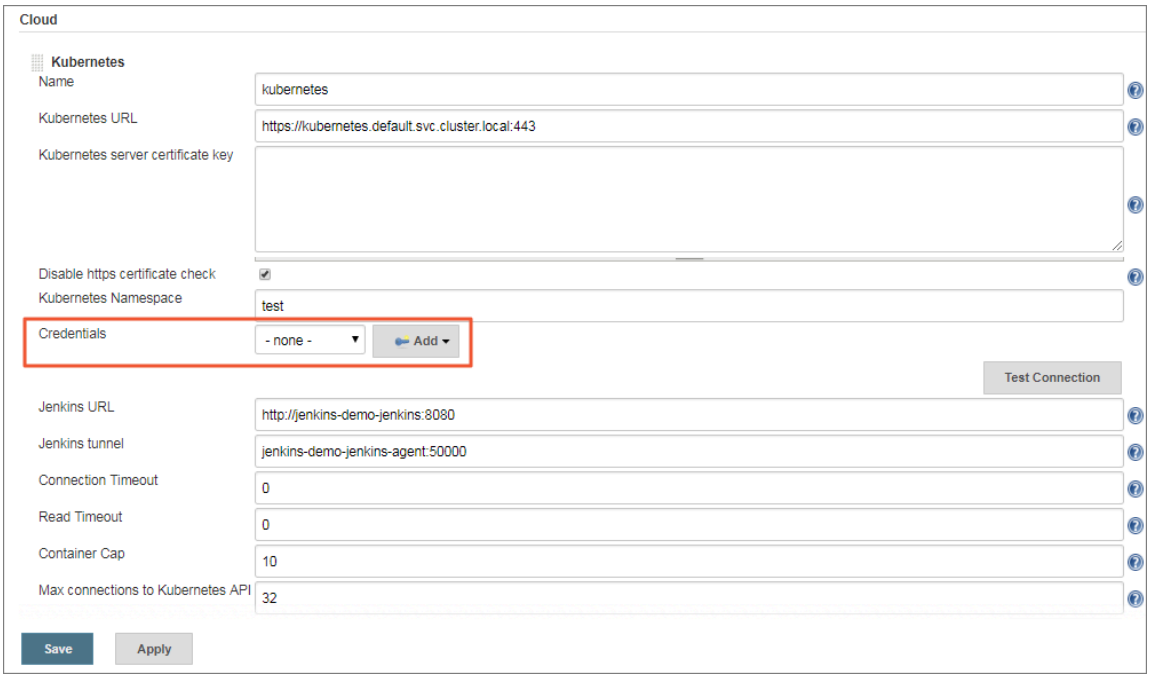
Create a cluster certificate and an image repository certificate, and build and deploy an application

1. Create a Kubernetes cluster certificate.

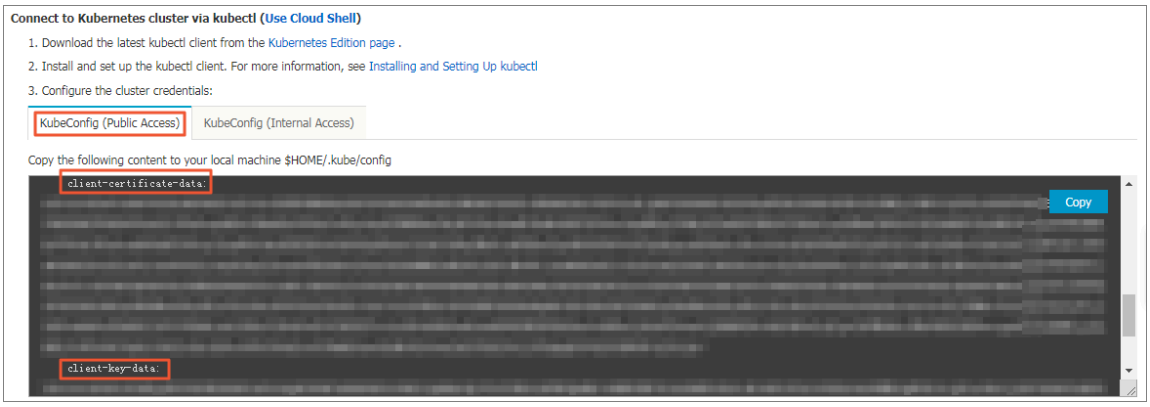
   a. In the left-side navigation pane, click Manage Jenkins.

   b. On the Manage Jenkins page, click Configure System.

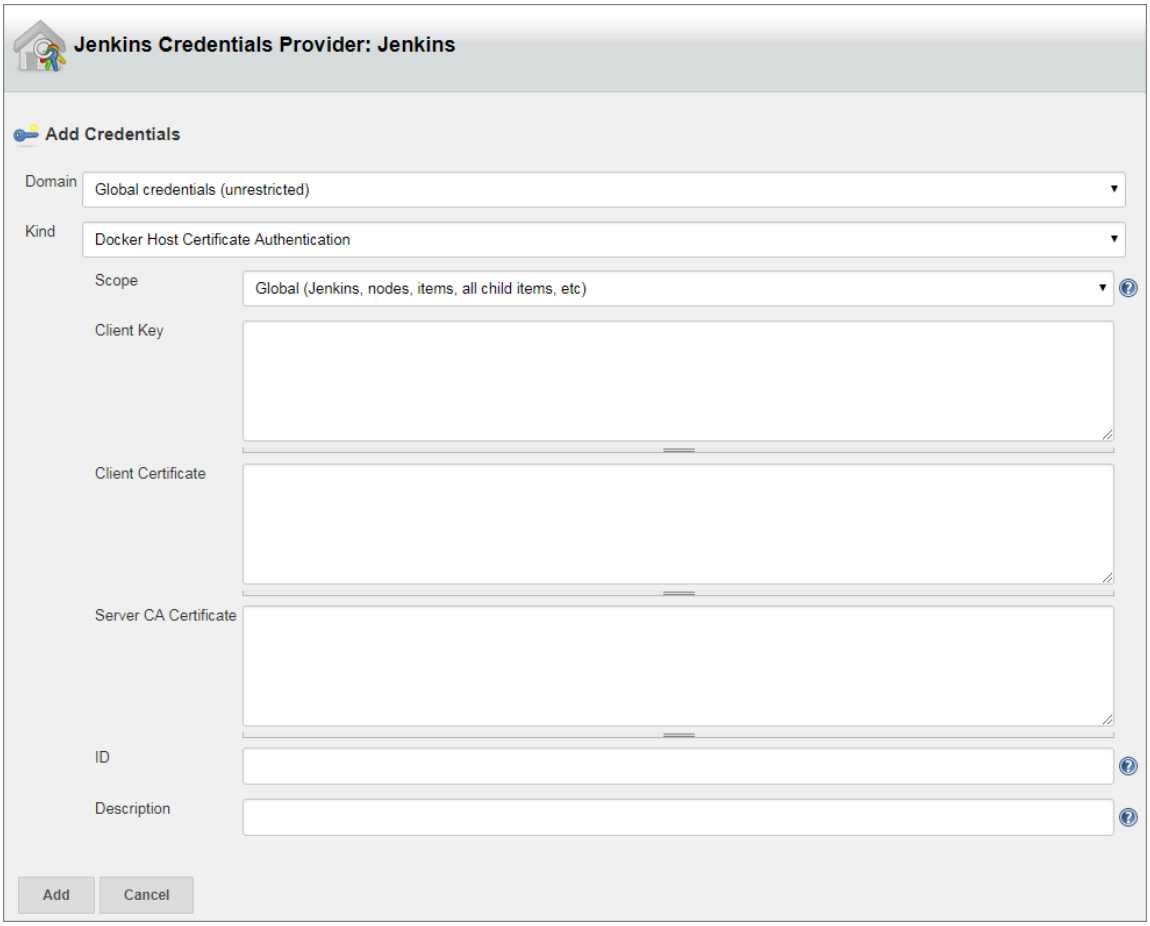
   c. In the Cloud area, click Add on the right of Credentials.



   Before adding a credential, you must obtain KubeConfig on the Basic Information tab page of the target Kubernetes cluster.



   d. In the diaplayed dialog box, set the following parameters:



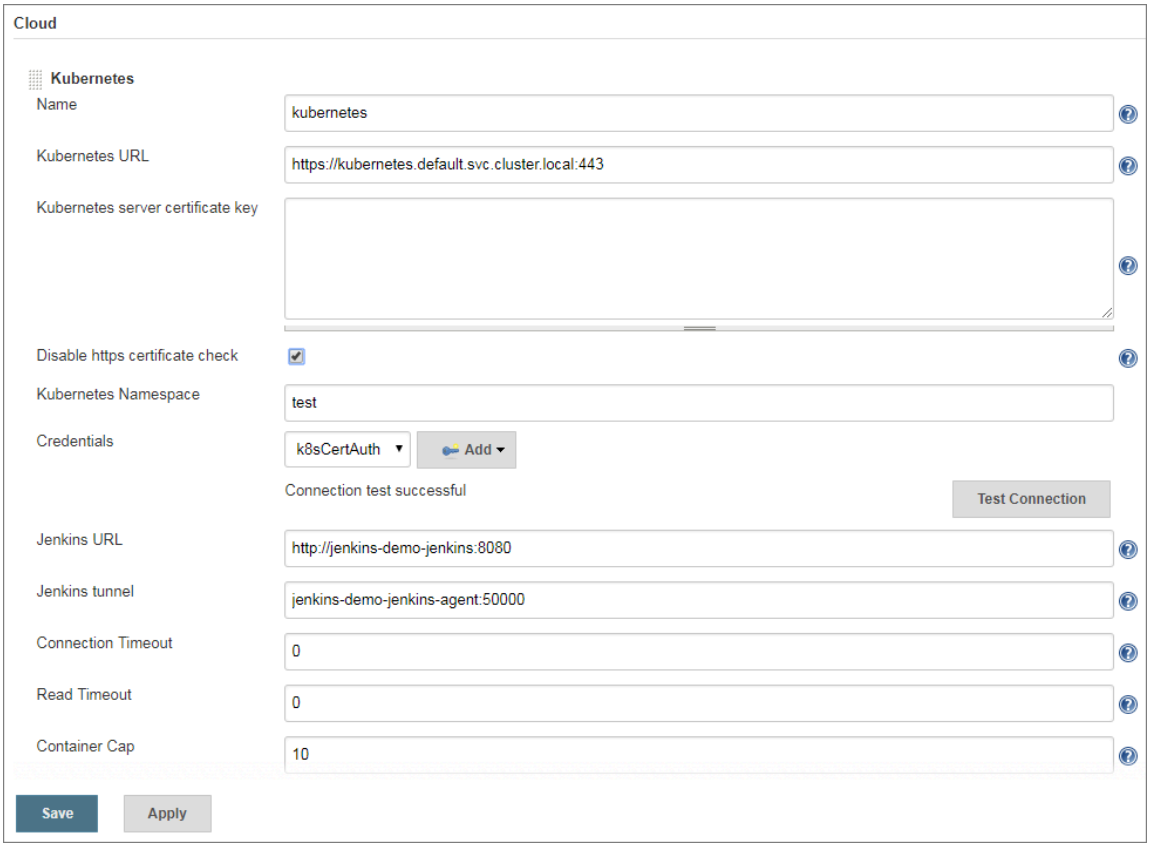
- Kind: Select Docker Host Certificate Authentication.
- Client Key: Paste the copied client-key-data content in KubeConfig.
- Client Certificate: Paste the copied client-certificate-data content in KubeConfig.
- ID: Enter the certificate ID. In this example, k8sCertAuth is entered.
- Description : Enter description content.

e. Click Add.

f. Test connectivity.

Select the added credential in the preceding step from the Credentials drop-down list, and then click Test Connection.

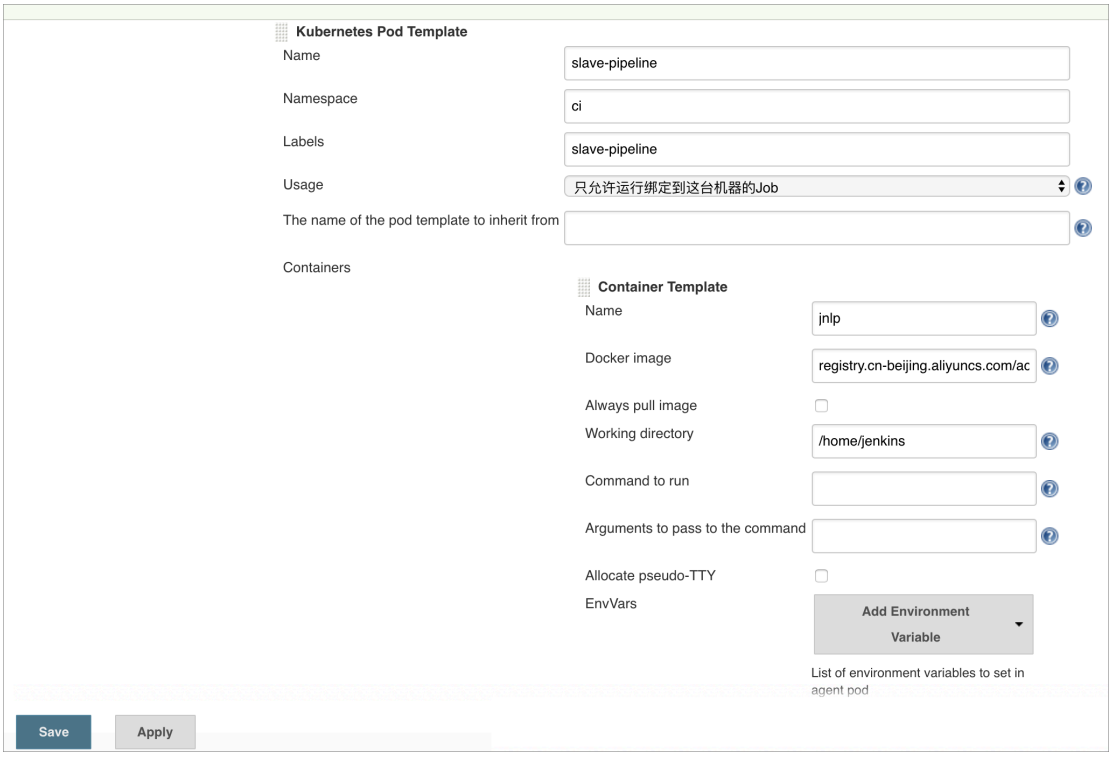
g. Set the Kubernetes cluster to dynamically build pods as follows.



h. Set the Kubernetes pod template.

The slave-pipeline uses four containers to create each corresponding stage of the pipeline.

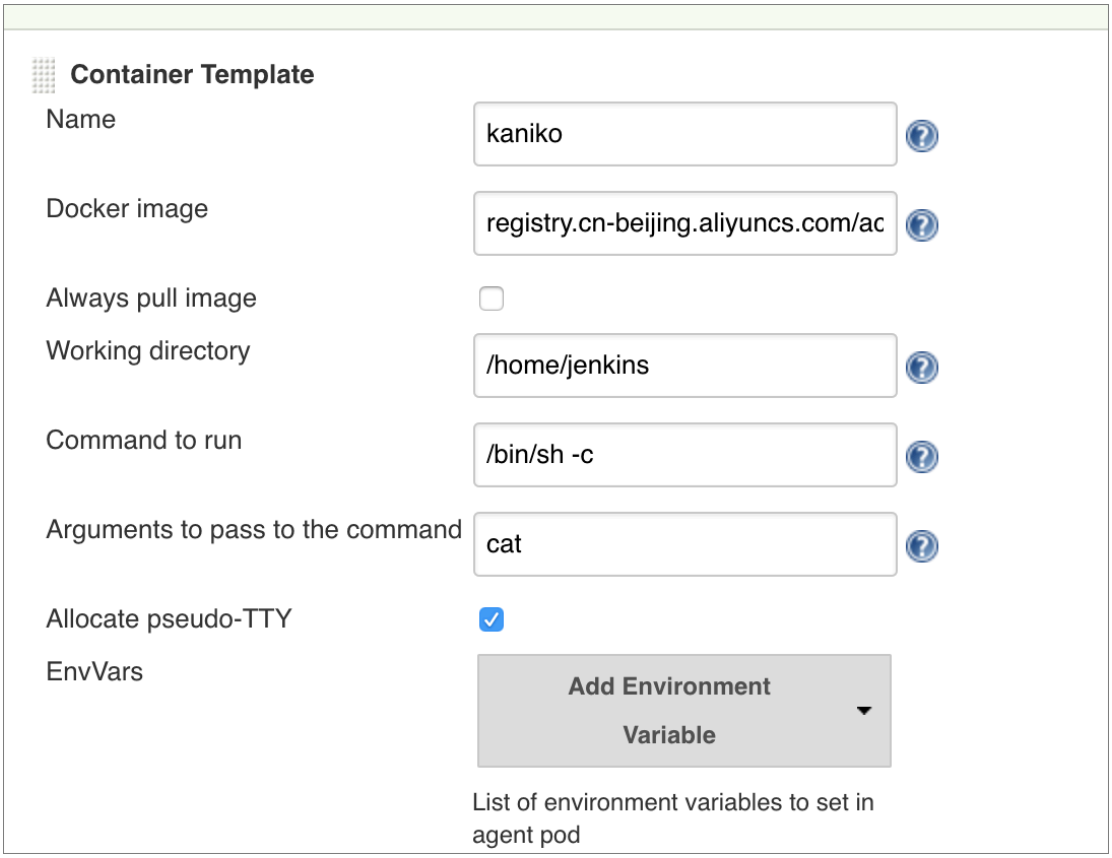
- Set container jnlp as follows.



Use jenkins-slave-jnlp as a Docker image. The jenkins-slave-jnlp image is used to create the jnlp node to connect the Jenkins master.

```
registry . cn - beijing . aliyuncs . com / acs - sample /
jenkins - slave - jnlp : 3 . 14 - 1
```

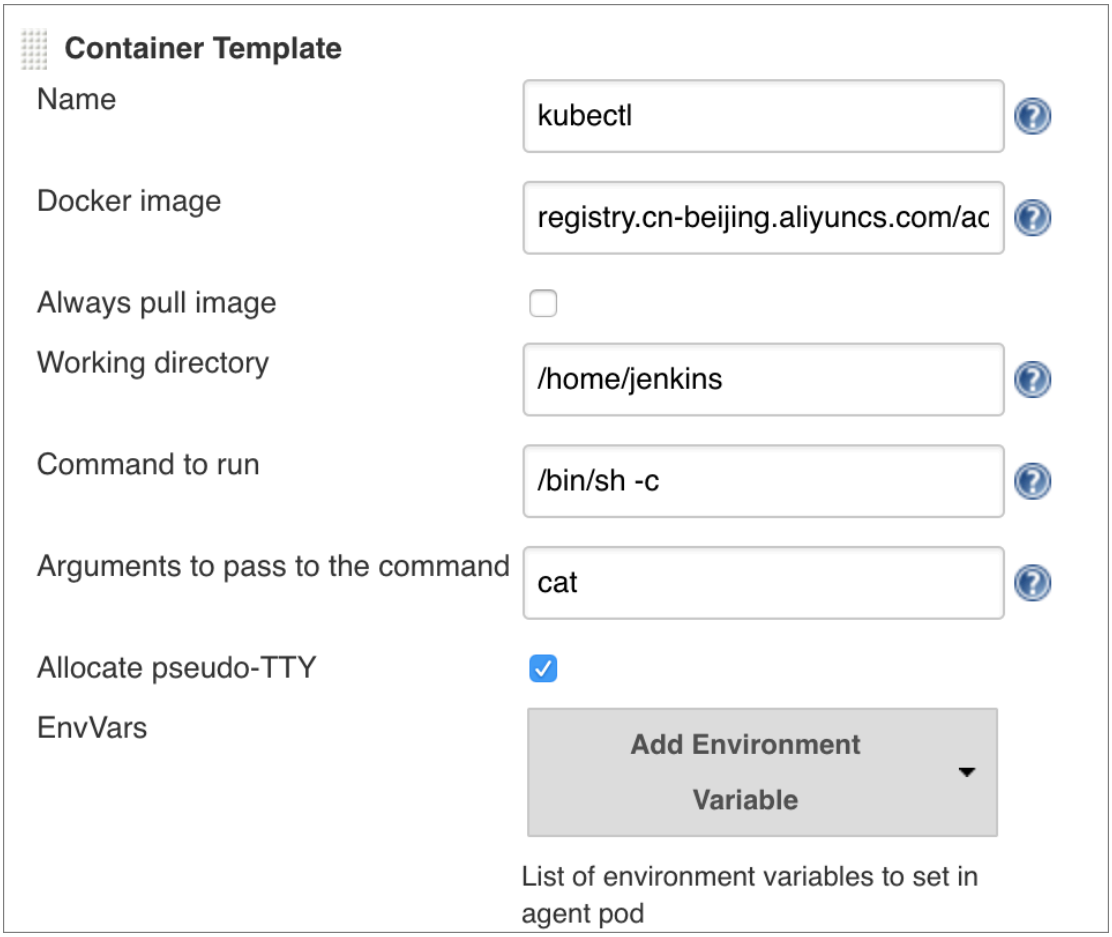
- Set container kaniko as follows.



Use jenkins-slave-kaniko as a Docker image. The jenkins-slave-kaniko image is used to create and push an image.

```
registry . cn - beijing . aliyuncs . com / acs - sample /
jenkins - slave - kaniko : 0 . 6 . 0
```

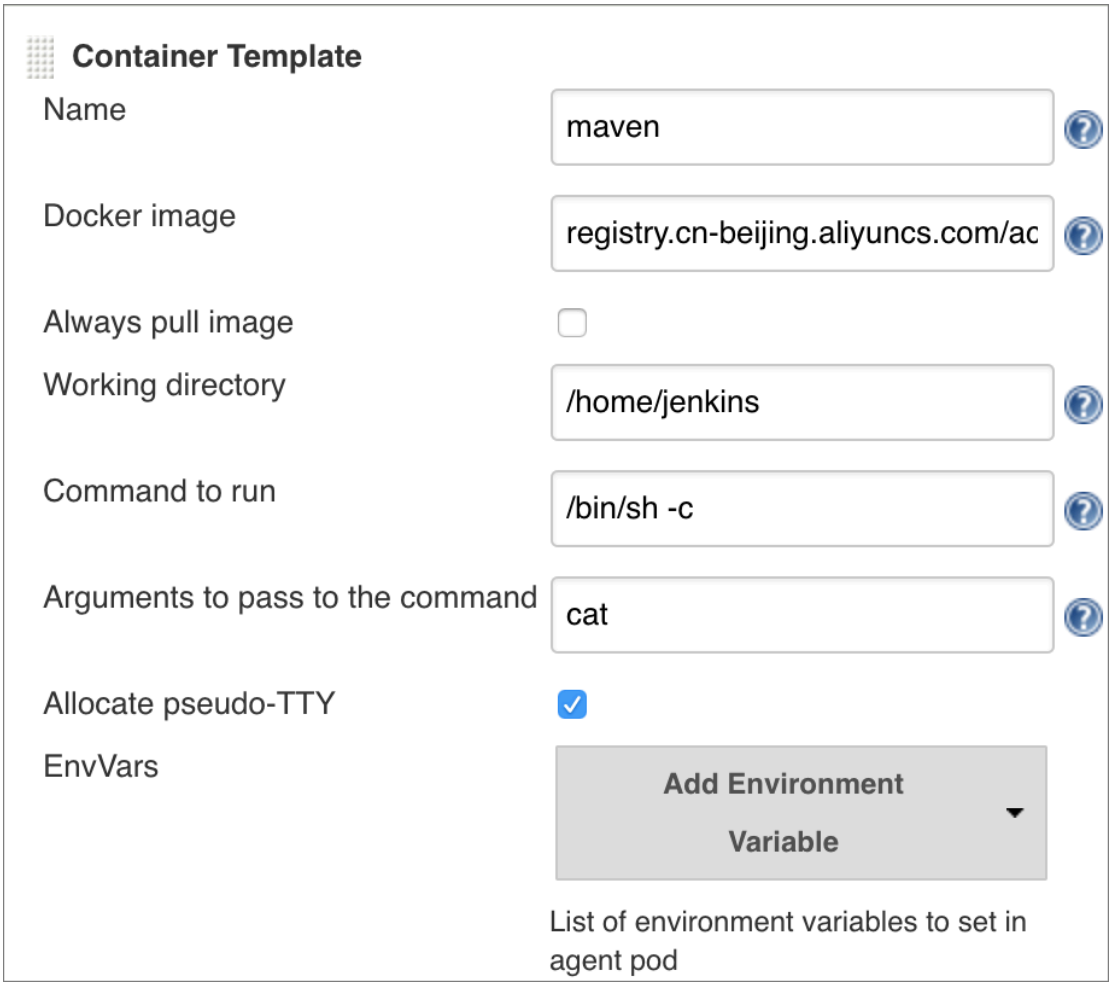
- Set container kubectl as follows.



Use jenkins-slave-kubectl as a Docker image. The jenkins-slave-kubectl image is used to deploy the application.

```
registry . cn - beijing . aliyuncs . com / acs - sample /
jenkins - slave - kubectl : 1 . 11 . 5
```

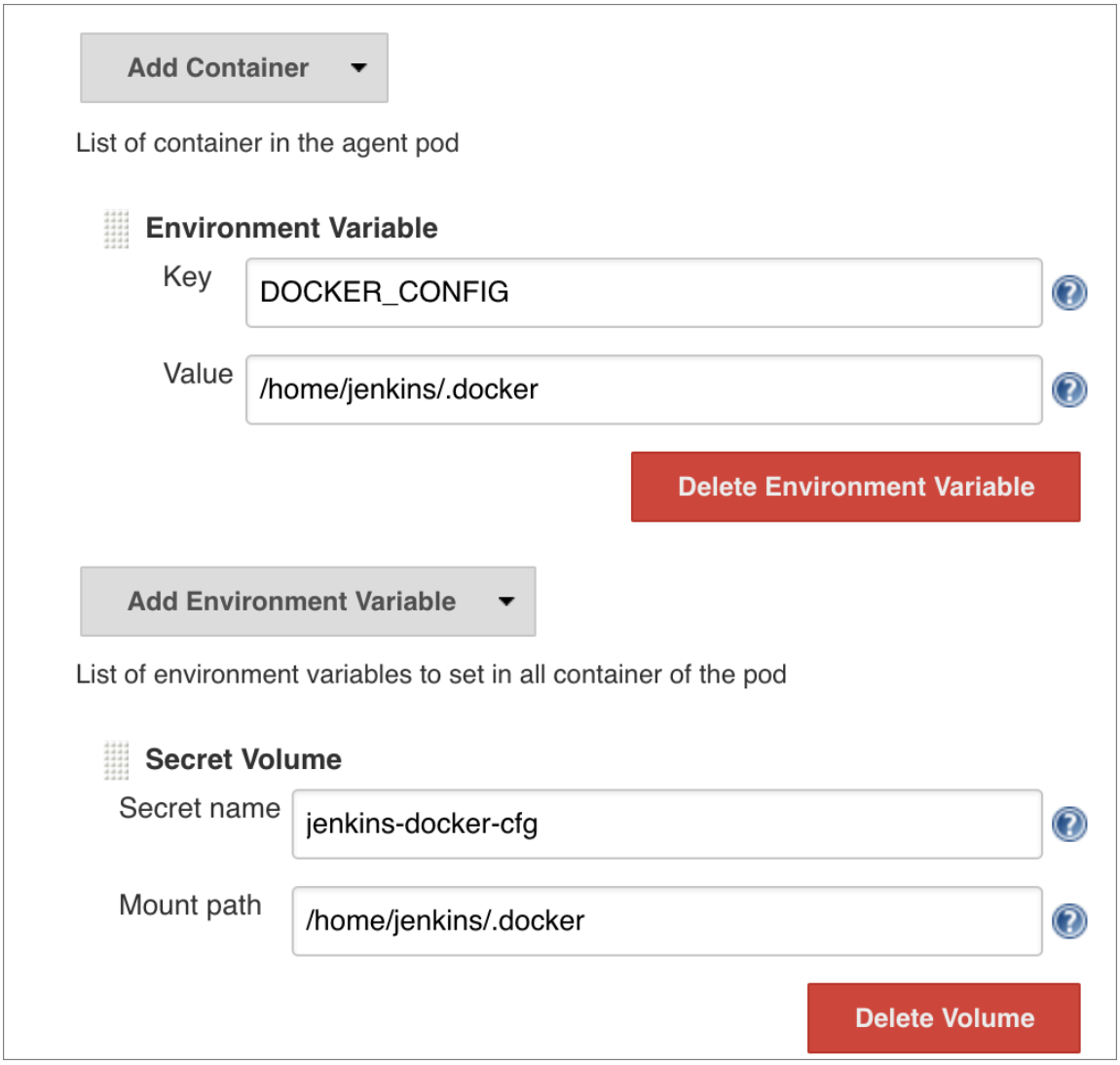
- **Set container maven as follows.**



Use jenkins-slave-maven as a Docker image.The jenkins-slave-maven image is used by mvn to package and build the application.

```
registry . cn - beijing . aliyuncs . com / acs - sample /
jenkins - slave - maven : 3 . 3 . 9 - jdk - 8 - alpine
```

i. **Set image repository permission for kaniko as follows.**



j. Click Save.

2. To set image repository permission, use kubectl to create jenkins-docker-cfg secret in the target Kubernetes cluster.

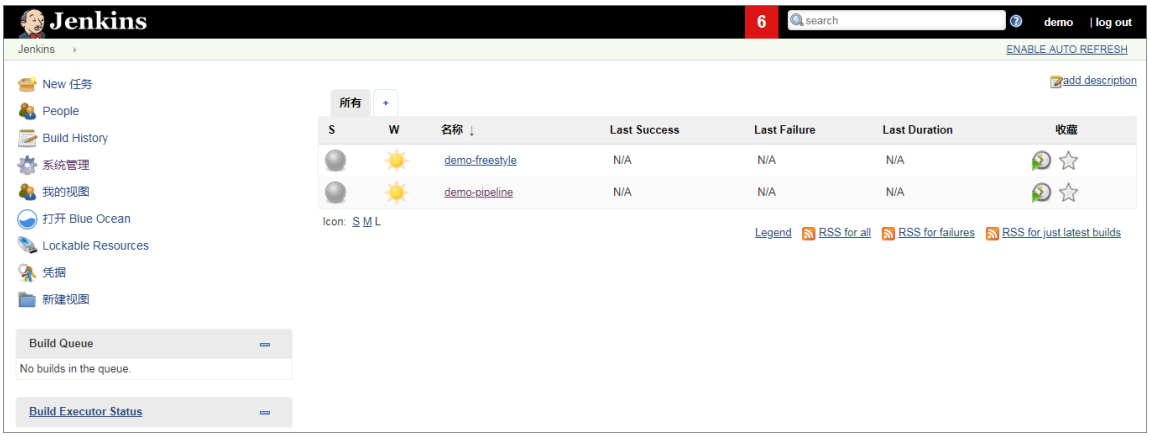
   In this example, the Beijing image repository provided by Alibaba Cloud is used.

```
$ docker login -u xxx -p xxx registry.cn-beijing.aliyuncs.com
Login Succeeded
```

```
$ kubectl create secret generic jenkins-docker-cfg -n
  ci --from-file=/root/.docker/config.json
```

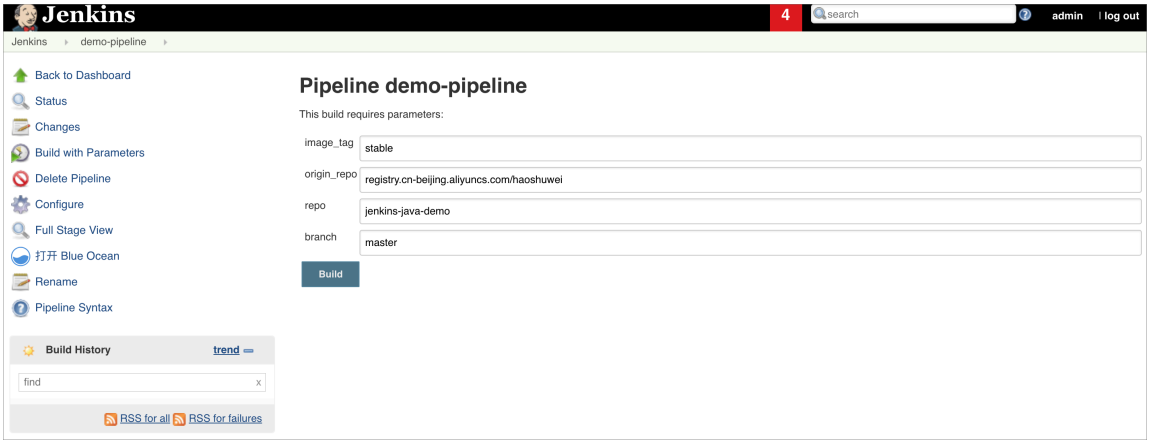
3. Create demo-pipeline and access the application service.

   a. On the Jenkins home page, click demo-pipeline.



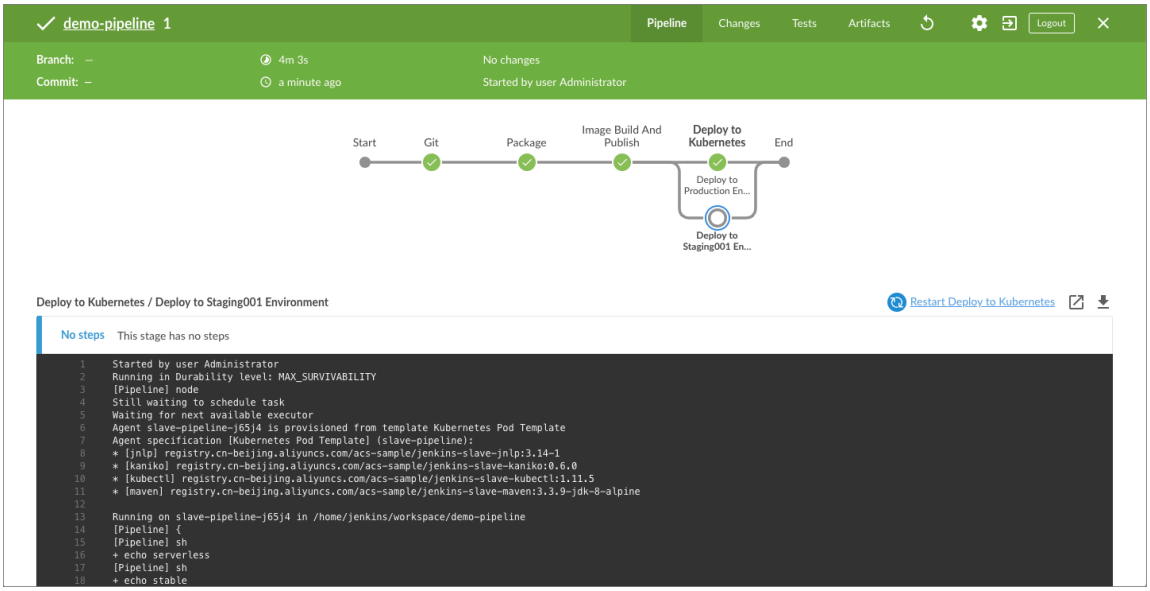
   b. In the left-side navigation pane, click Build with Parameters.

   c. Modify the parameters according to your image repository information. In this example, the source code repository is master, and the image is registry.cn-beijing.aliyuncs.com/haoshuwei:stable.



   d. Click Build.

   e. Click Build History to check the result. The following figure indicates a success.



f. Log on to the *Container Service console* to view the IP addresses of the services provided by the application.

Click *Here* to obtain the source code repository used in the example.

For more information, see *Container Service*.

For more information, see *kaniko*.

10.2 Use GitLab CI to run a GitLab runner and activate a pipeline in a Kubernetes cluster

This topic describes how to install and register a GitLab runner in a Kubernetes cluster and add a Kubernetes executor to build an application. After completing the preceding operations, you can build a Java application source code project. The example in this topic shows a CI/CD pipleline that includes the following actions in order: compiling and building the code, packaging the application image, and deploying the application.

Background

In the following example, a Java software project is built and deployed in a Kubernetes cluster that runs on Alibaba Cloud Container Service for Kubernetes. The example shows how to use GitLab CI to run a GitLab runner, set a Kubernetes executor, and activate a pipeline in an Alibaba Cloud Kubernetes cluster.

Create the GigLab source code project and upload the example code

1. Create the GitLab source code project.

   In this example, the address of the created GitLab source code project is as follows:

   ```
   http :// xx . xx . xx . xx / demo / gitlab - java - demo . git
   ```

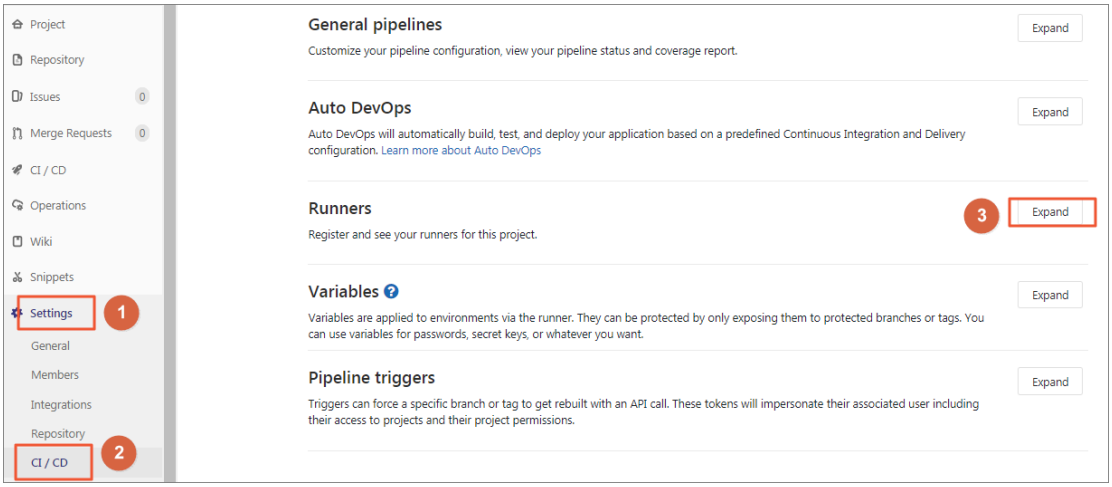
2. Run the following commands to obtain the example code and upload it to GitLab:

   ```
   $ git clone https :// code . aliyun . com / CodePipeli ne /
    gitlabci - java - demo . git
   $ git remote add gitlab http :// xx . xx . xx . xx / demo /
    gitlab - java - demo . git
   ```

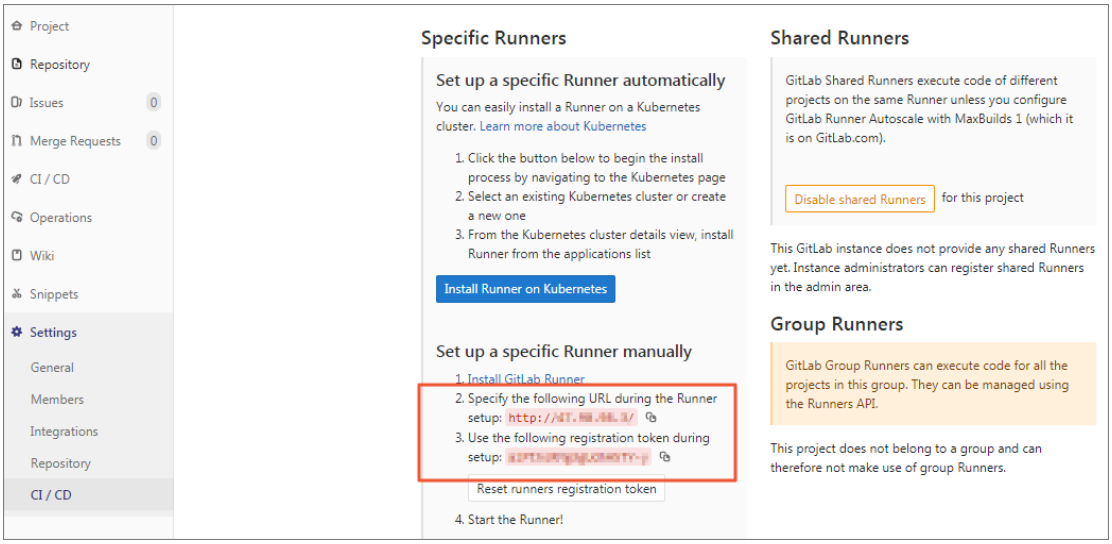
```
$ git push gitlab master
```

Install the GitLab runner in a Kubernetes cluster

1. Obtain the GitLab runner registration information.
   a. Obtain the registration information of the runner specific to the project.
      A. Log on to GitLab.
      B. In the top navigation bar, choose Projects > Your projects.
      C. On the Your projects tab page, select the target project.
      D. In the left-side navigation pane, choose Settings > CI / CD.
      E. Click Expand on the right of Runners.



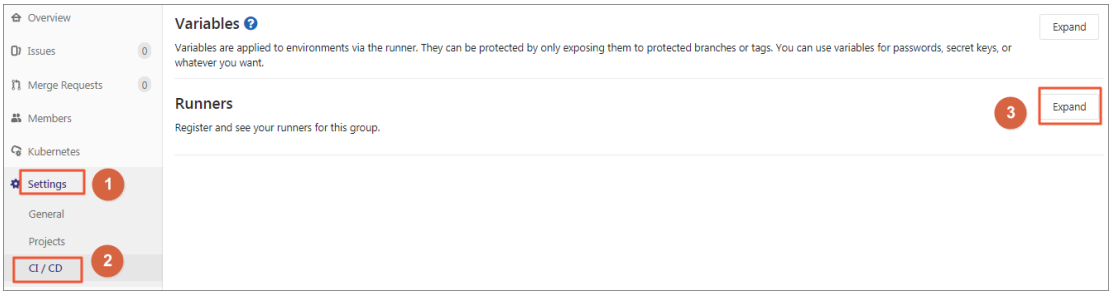
      F. Obtain the URL and the registration token.



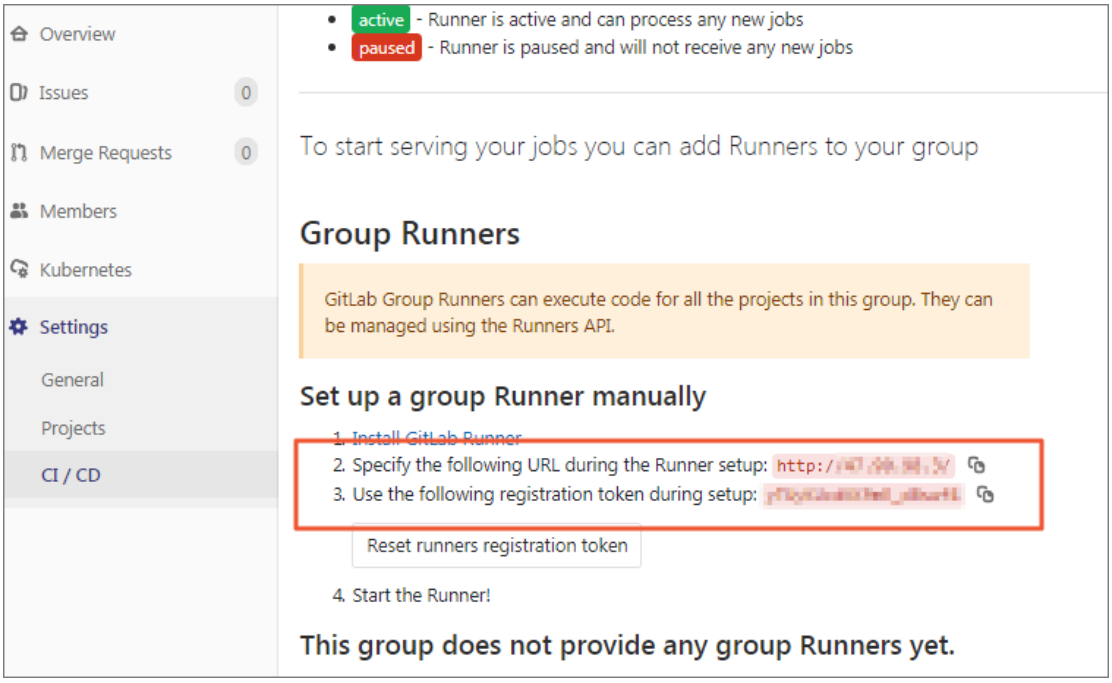
   b. Obtain the group runner registration information.
      A. In the top navigation bar, choose Groups > Your groups.
      B. On the Your groups tab page, select the target group.

C. In the left-side navigation pane, choose Settings > CI / CD.

D. Click Expand on the right of Runners.



E. Obtain the URL and the registration token.



c. Obtain the shared runner registration information.

Note:

Only the administrator has permission to perform this action.

A. In the top navigation bar, click  .

B. In the left-side navigation pane of the Admin Area page, choose Overview > Runners.

C. Obtain the URL and the registration token.

2. Run the following command to obtain and modify the Helm Chart of the GitLab runner:

```
$ git clone https://code.aliyun.com/CodePipeline/gitlab-runner.git
```

Modify the *values.yaml* file as follows:

```
## GitLab Runner Image
##
image: gitlab/gitlab-runner:alpine-v11.4.0

## Specify an 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 retrieved from your GitLab Instance.
##
runnerRegistrationToken: "AMvEWrBTBu-d8czEYyfY"
## 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
```

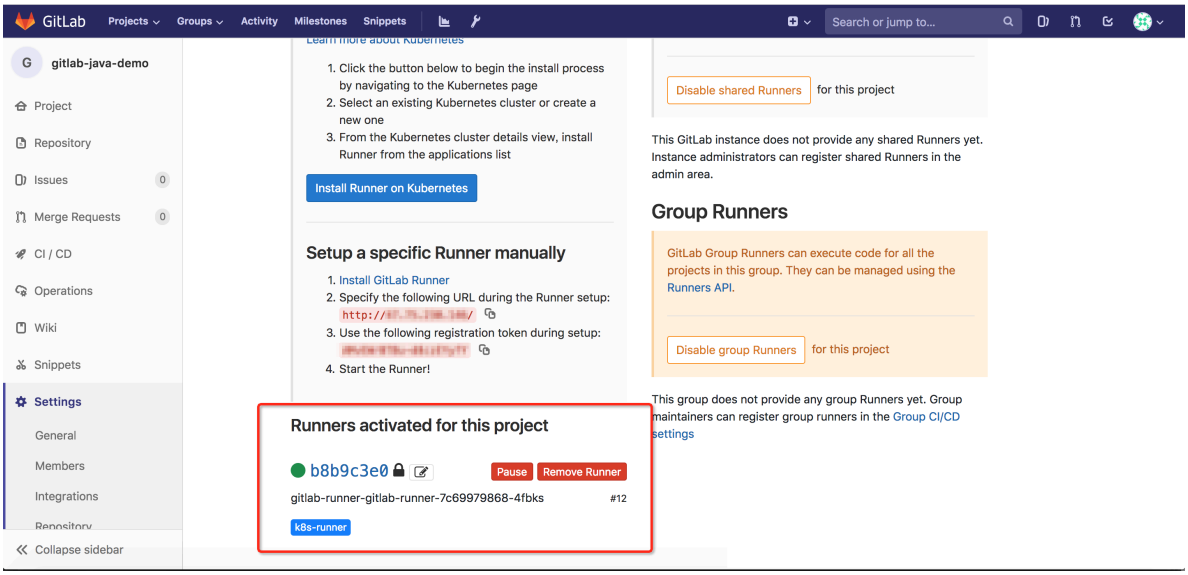
```
##
 checkInter  val :  30

##  For   RBAC   support :
##
 rbac :
   create :  true
   clusterWid  eAccess :  false

##  Configure   integrated   Prometheus   metrics   exporter
##
 metrics :
   enabled :  true

##  Configurat  ion   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   release )
  ##
   namespace :  gitlab

   cachePath : "/ opt / cache "

   cache : {}
   builds : {}
   services : {}
   helpers : {}

 resources : {}
```

3. **Run the following command to install the GitLab runner.**

```
$  helm   package  .
 Successful  ly   packaged   chart   and   saved   it   to : / root
 / gitlab / gitlab - runner / gitlab - runner - 0 . 1 . 37 . tgz
```

```
$ helm install -- namespace gitlab -- name gitlab - runner
 *. tgz
```

Check whether the related deployment/pod has been started. If the related deployment/pod has been started, the GitLab runner that has been registered in GitLab is displayed.



Set the GtiLab runner cache

GitLab runners have a limited cache capacity. Therefore, you need to mount a volume to your GitLab runner so that the volume functions as the GitLab runner cache. In this example, the */ opt / cache* directory is used as the GitLab runner cache by default. You can modify the `runners . cachePath` field of the *values . yaml* file to change the default cache directory.

For example, to create a maven cache, add the `MAVEN_OPTS` variable to `variables` and specify a local cache directory as follows:

```
variables :
  KUBECONFIG : / etc / deploy / config
  MAVEN_OPTS : "- Dmaven . repo . local =/ opt / cache /. m2 /
repository "
```

To mount a new volume, modify the following fields in the *templates / configmap . yaml* file:

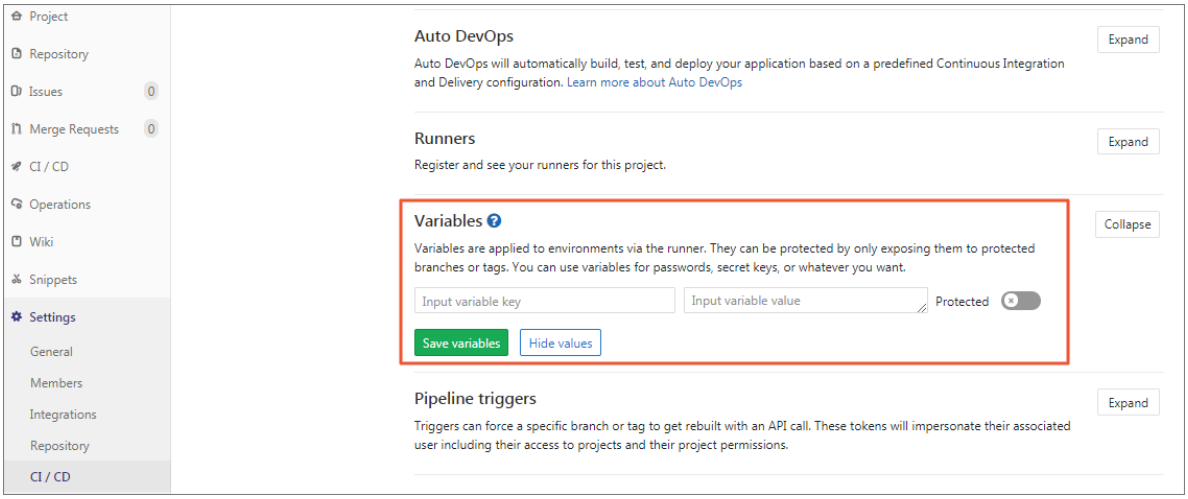
```
cat >>/ home / gitlab - runner /. gitlab - runner / config . toml
<< EOF
          [[ runners . kubernetes . volumes . pvc ]]
             name = " gitlab - runner - cache "
             mount_path = "{{ . Values . runners . cachePath }}"
```

```
EOF
```

This means that you need to modify the settings of `config.toml` during the period between the time at which the GigLab runner was registered and the time at which the runner has not yet been started.

Set the global variables

1. In the top navigation bar, choose Projects > Your projects.
2. On the Your projects tab page, select the target project.
3. In the left-side navigation pane, choose Settings > CI / CD.
4. Click Expand on the right of Runners to add available to the GitLab runner.



In this example, add the following three variables:

- REGISTRY_USERNAME: indicates a registry username.
- REGISTRY_PASSWORD: indicates a registry password.
- kube_config: indicates a string of KubeConfig code characters.

Run the following command to generate a string of KubeConfig code characters:

```
echo $( cat ~/.kube/config | base64 ) | tr -d " "
```

Edit the `.gitlab-ci.yml` file

Edit the `.gitlab-ci.yml` file to compile and build the Java demo source code project, push the application image, and deploy the application. For more information, see the .gitlab-ci.yml.example of the gitlabci-java-demo source code project. This section first shows an example of an entire YAML file, and then describes key sections of the file in more detail.

The entire . *gitlab - ci . yml* **file is as follows:**

```
image :  docker : stable
stages :
 -  package
 -  docker_bui  ld
 -  deploy_k8s
variables :
  KUBECONFIG : / etc / deploy / config
mvn_build_  job :
  image :  registry . cn - beijing . aliyuncs . com / codepipeli
ne / public - blueocean - codepipeli  ne - slave - java : 0 . 1 -
63b99a20
  stage :  package
  tags :
   -  k8s - test
  script :
   -  mvn   package  - B  - DskipTests
   -  cp   target / demo . war  / opt / cache
docker_bui  ld_job :
  image :  registry . cn - beijing . aliyuncs . com / codepipeli
ne / public - blueocean - codepipeli  ne - slave - java : 0 . 1 -
63b99a20
  stage :  docker_bui  ld
  services :
   -  docker : dind
  variables :
    DOCKER_DRI  VER :  overlay
    DOCKER_HOS  T :  tcp :// localhost : 2375
  tags :
   -  k8s - test
  script :
   -  docker   login  - u  $ REGISTRY_U  SERNAME  - p  $ REGISTRY_P
ASSWORD   registry . cn - beijing . aliyuncs . com
   -  mkdir   target
   -  cp  / opt / cache / demo . war   target / demo . war
   -  docker   build  - t   registry . cn - beijing . aliyuncs . com
/ gitlab - demo / java - demo :$ CI_PIPELIN  E_ID  .
   -  docker   push   registry . cn - beijing . aliyuncs . com /
gitlab - demo / java - demo :$ CI_PIPELIN  E_ID
deploy_k8s  _job :
  image :  registry . cn - beijing . aliyuncs . com / codepipeli
ne / public - blueocean - codepipeli  ne - slave - java : 0 . 1 -
63b99a20
  stage :  deploy_k8s
  tags :
   -  k8s - test
  script :
   -  mkdir  - p  / etc / deploy
   -  echo  $ kube_confi  g  | base64  - d  > $ KUBECONFIG
   -  sed  - i  " s / IMAGE_TAG /$ CI_PIPELIN  E_ID / g "  deployment
. yaml
   -  cat   deployment . yaml
   -  kubectl   apply  - f   deployment . yaml
```

The . *gitlab - ci . yml* **file defines a pipeline that is executed in three stages.**

```
image :  docker : stable   #  If   no   image   is   specified   for
  each   step   of   the   pipeline ,  the   docker : stable   image
  is   used   by   default .
stages :
 -  package                 #  package   the   soucre   code
```

```
 - docker_bui ld          # build , package , and push the
applicatio n image
 - deploy_k8s           # deploy the applicatio n
variables :
  KUBECONFIG : / etc / deploy / config    # define the global
variable , namely , KUBECONFIG
```

- Package the maven source code

```
mvn_build_ job :     # the job name
  image : registry . cn - beijing . aliyuncs . com / codepipeli
ne / public - blueocean - codepipeli ne - slave - java : 0 . 1 -
63b99a20   # the image used in this stage
  stage : package       # the stage name
  tags :                    # GitLab Runner tag
   - k8s - test
  script :
   - mvn package - B - DskipTests   # the script to
execute the build
   - cp target / demo . war / opt / cache    # save the
build to the cache
```

- Build, package, and push the application image

```
docker_bui ld_job :  # the job name
  image : registry . cn - beijing . aliyuncs . com / codepipeli
ne / public - blueocean - codepipeli ne - slave - java : 0 . 1 -
63b99a20  # the image used in this stage
  stage : docker_bui ld   # the stage name
  services :                  # Use the docker : dind
service . You must set the runners . privileged in
GitLab Runner Helm Chart to true .
   - docker : dind
  variables :
    DOCKER_DRI VER : overlay
    DOCKER_HOS T : tcp :// localhost : 2375    # Connect the
Docker Daemon .
  tags :                    # GitLab Runner tag
   - k8s - test
  script :
   - docker login - u REGISTRY_U SERNAME - p $ REGISTRY_P
ASSWORD registry . cn - beijing . aliyuncs . com    # log on
to the registry
   - mkdir target
   - cp / opt / cache / demo . war target / demo . war
   - docker build - t registry . cn - beijing . aliyuncs .
com / gitlab - demo / java - demo :$ CI_PIPELIN E_ID .     #
Package the Docker image . The used tag is the
pipeline ID .
   - docker push registry . cn - beijing . aliyuncs . com /
gitlab - demo / java - demo :$ CI_PIPELIN E_ID       # Push the
  Docker image .
```

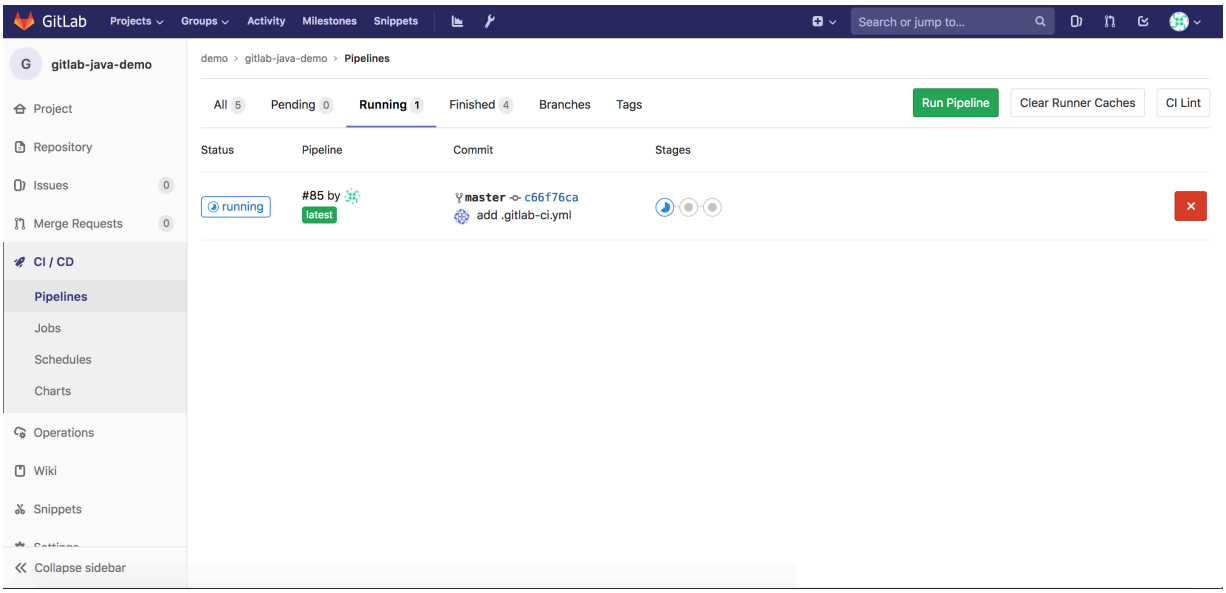
- Deploy the application

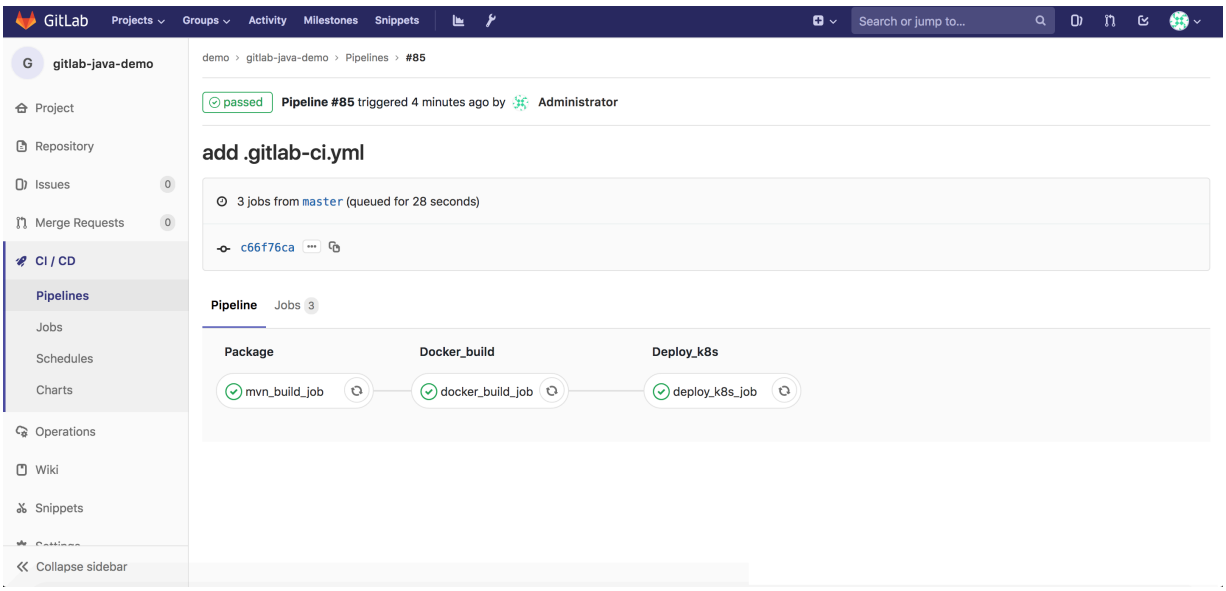
```
deploy_k8s _job :   # the job name
  image : registry . cn - beijing . aliyuncs . com / codepipeli
ne / public - blueocean - codepipeli ne - slave - java : 0 . 1 -
63b99a20    # the image used in this stage
  stage : deploy_k8s    # the stage name
  tags :                    # GitLab Runner tag
   - k8s - test
```

```
  script :
    -  mkdir  - p  / etc / deploy
    -  echo  $ kube_confi  g  | base64  - d  > $ KUBECONFIG    #
Set  the  config  file  that  is  used  to  connect  the
Kubernetes  cluster .
    -  sed  - i  " s / IMAGE_TAG /$ CI_PIPELIN  E_ID / g "
deployment . yaml  #  Dynamicall  y  replace  the  image  tag
  of  the  deployment  file .
    -  kubectl  apply  - f  deployment . yaml
```

Activate the pipeline

After you submit the `.gitlab-ci.yml` file, the gitlab-java-demo project automatically detects this file and activate the pipeline.





Access the application service

If no namespace is specified in the deployment file, the application is deployed in the GitLab namespace by default.

```
$ kubectl - n gitlab get svc
NAME TYPE CLUSTER - IP EXTERNAL - IP
PORT ( S ) AGE
java - demo LoadBalanc er 172 . 19 . 9 . 252 xx . xx .
xx . xx 80 : 32349 / TCP 1m
```

Visit xx.xx.xx.xx/demo in your browser to check the result.

For more information, see *Container Service* and *GitLab CI*.

10.3 Deploy Jenkins in a serverless Kubernetes cluster and perform an application pipeline build

This topic describes how to deploy Jenkins, a continuous integration environment, in an Alibaba Cloud serverless Kubernetes cluster, and how to perform an application pipeline build. The example in this topic details the pipeline build, including how to compile the source code of the application, build and push the application image, and deploy the application.

Prerequisites

You have created a serverless Kubernetes cluster. For more information, see *Create a serverless Kubernetes cluster*.

Deploy Jenkins

1. Run the following command to download the Jenkins deployment file:

```
$ git clone https :// github . com / AliyunCont ainerServi ce
/ jenkins - on - serverless . git
$ cd jenkins - on - serverless
```

2. Persist the jenkins_home directory.

   Serverless Kubernetes clusters do not support cloud disks. To persist the jenkins_home directory, create the nfs volume, then modify the `serverless - k8s - jenkins - deploy . yaml` file to add the following field annotations and set nfs parameters:

```
# volumeMoun ts :
# - mountPath : / var / jenkins_ho me
# name : jenkins - home
.....
```

```
      # volumes :
      #  -  name :  jenkins - home
      #     nfs :
      #       path : /
```

```
#       server :
```

3. Run the following command to deploy Jenkins:

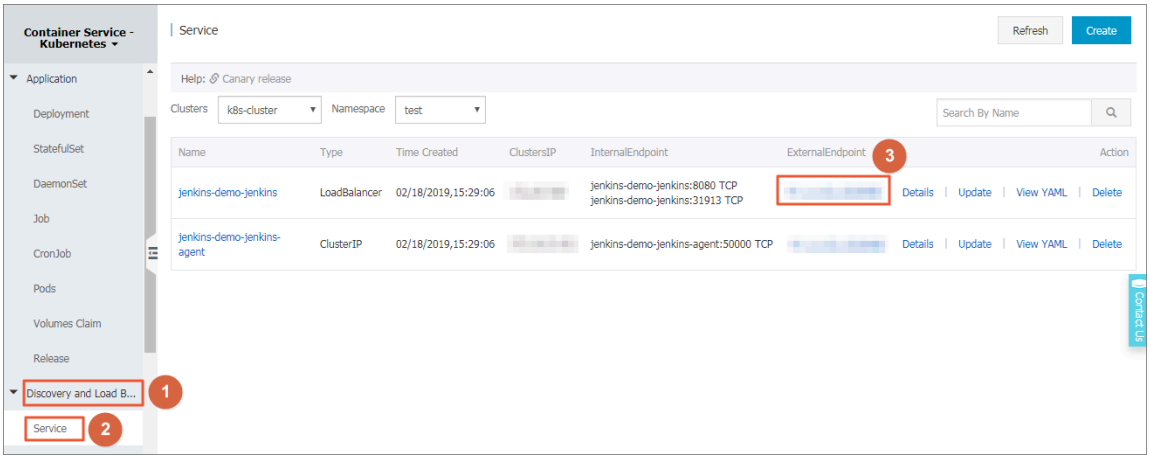
```
$ kubectl apply -f serverless-k8s-jenkins-deploy.yaml
```

4. Log on to Jenkins.

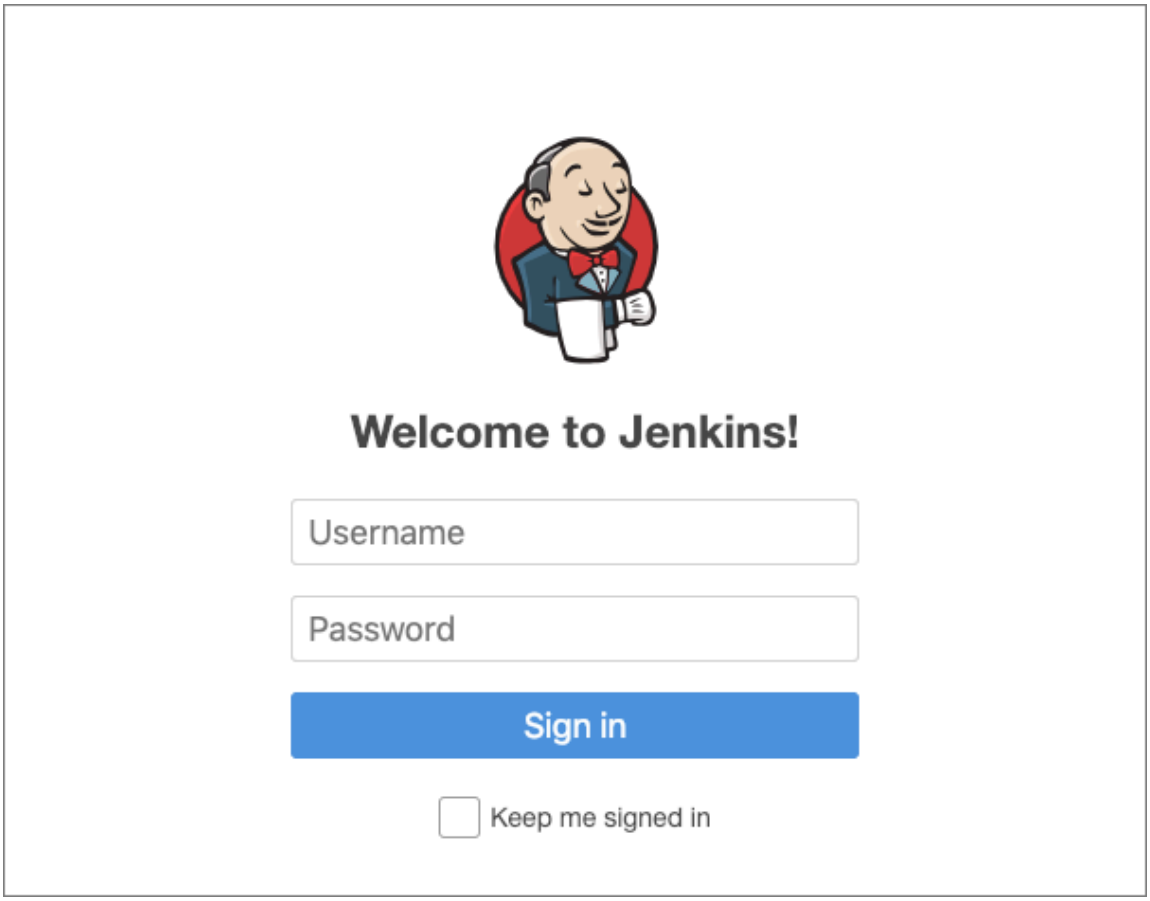
   a. Log on to the *Container Service console*.

   b. In the left-side navigation pane, choose Discovery and Load Balancing > Service.

   c. Click the external endpoint of the Jenkins service to log on to Jenkins.

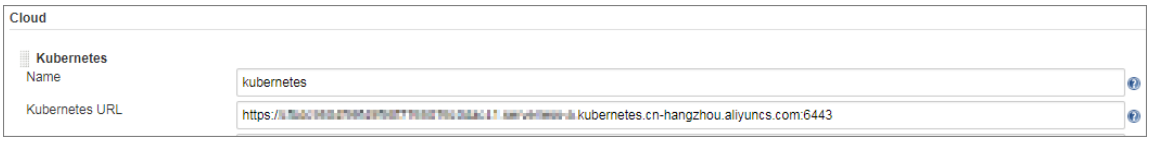


   d. On the Jenkins logon page, enter the user name and the password. The default user name and password are admin. We recommend that you modify them after you log on to Jenkins.


Welcome to Jenkins!
Username
Password
Sign in
Keep me signed in

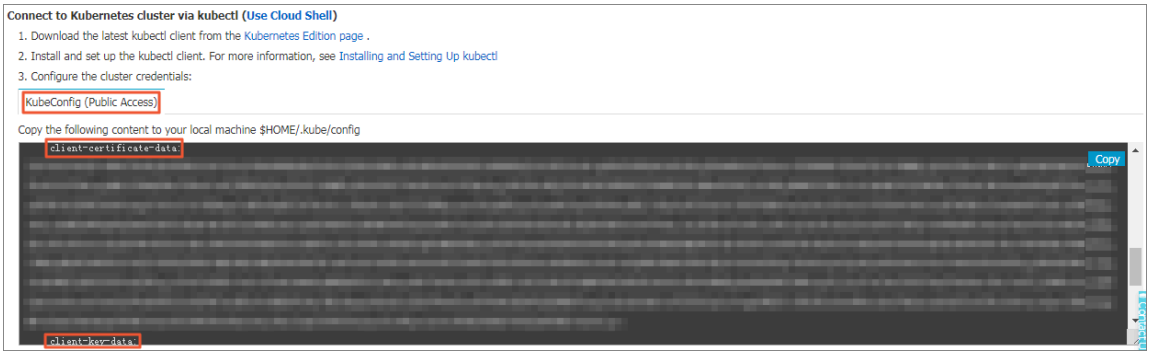
Create a cluster certificate and an image repository certificate, and build and deploy an application

1. Set Kubernetes Cloud parameters to dynamically create a slave pod.
   a. In the left-side navigation pane, click Manage Jenkins.
   b. Click Configure System.
   c. In the Cloud area, enter the API server URL in KubeConfig as the Kubernetes URL.

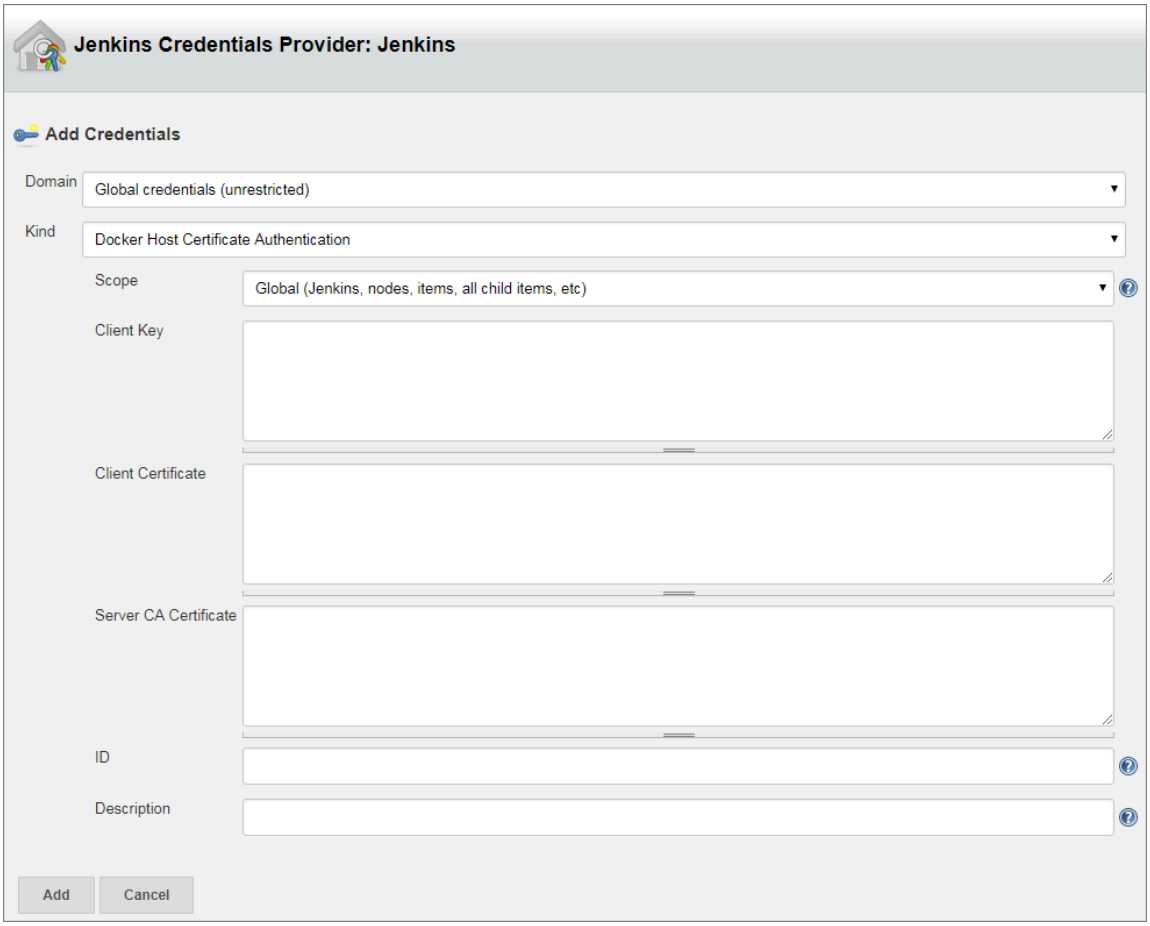


   d. Click Add on the right of Credentials.

   Before adding a credential, you must obtain KubeConfig on the Basic Information tab page of the target Kubernetes cluster.



   In the displayed dialog box, set the following parameters:

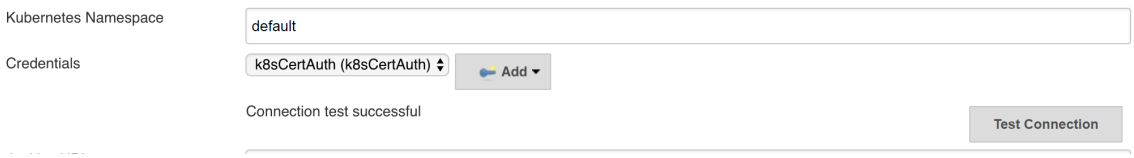


- Kind: Select Docker Host Certificate Authentication.
- Client Key: Paste the copied client-key-data content in KubeConfig.
- Client Certificate: Paste the copied client-certificate-data content in KubeConfig.
- ID: Enter the certificate ID. In this example, k8sCertAuth is entered.
- Description: Enter description content.

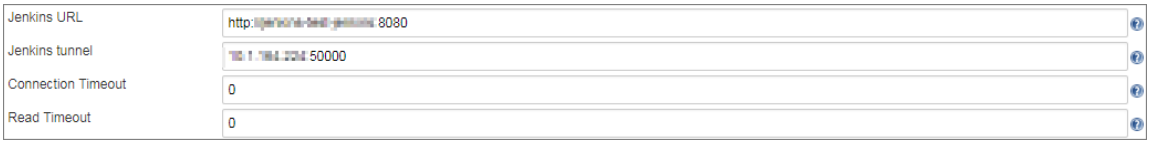
e. Click Add.

f. Test connectivity.

Select the added credential in the preceding step from the Credentials drop-down list, and then click Test Connection.



g. Enter the external endpoint of the jenkins service as Jenkins URL, and enter the external endpoint of the jenkins-agent service as Jenkins tunnel.



h. Click Save.

2. To set image repository permission, use kubectl to create jenkins-docker-cfg secret in the target serverless Kubernetes cluster.

   In this example, the Beijing image repository provided by Alibaba Cloud is used.

```
$ docker login -u xxx -p xxx registry.cn-beijing.
aliyuncs.com
Login Succeeded
```

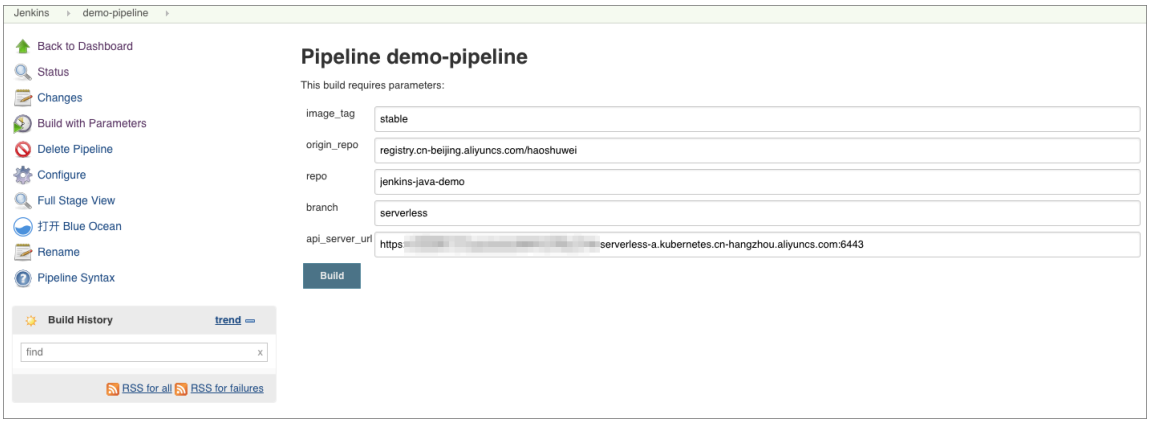
```
$ kubectl create secret generic jenkins-docker-cfg --
 from-file=/root/.docker/config.json
```

3. Create demo-pipeline and access the application service.

   a. On the Jenkins home page, click demo-pipeline.

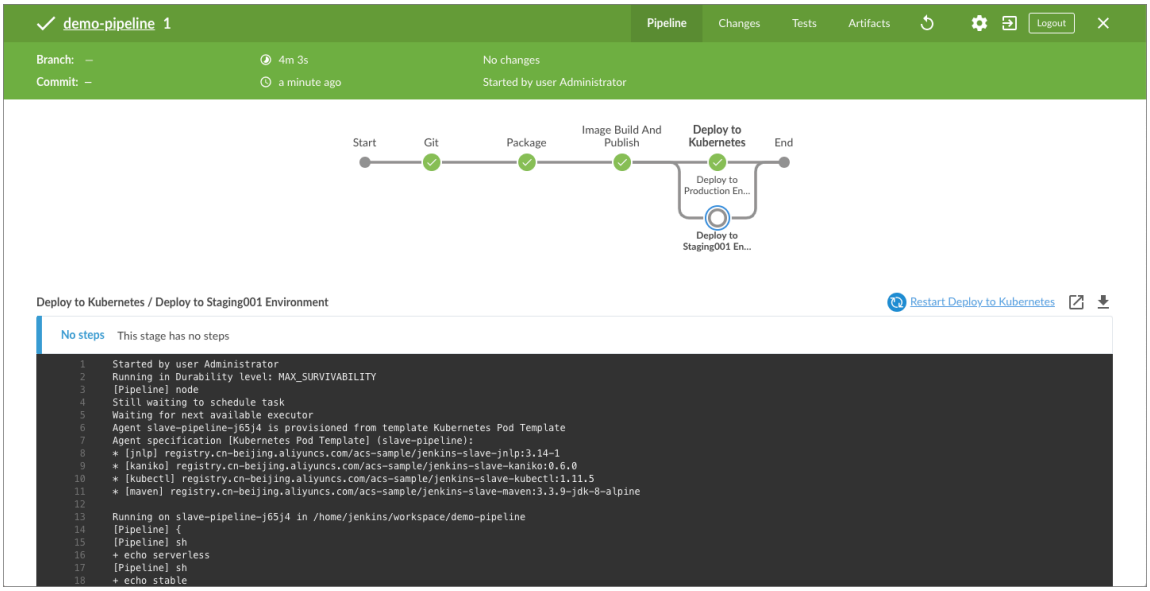
   b. In the left-side navigation pane, click Build with Parameters.

   c. Modify the parameters according to your image repository information. In this example, the source code repository branch is serverless, and the image is registry.cn-beijing.aliyuncs.com/haoshuwei:stable.



   d. Click Build.

   e. Click Build History to check the result. The following figure indicates a success.



   f. Log on to the *Container Service console* to view the IP addresses of the services provided by the application.

The source code repository used in this topic can be found at *https://github.com/AliyunContainerService/jenkins-demo*.

10.4 Use Bamboo to deploy a remote agent and run a build plan

This topic describes how to use Bamboo to deploy a remote agent in a Kubernetes cluster implemented with Alibaba Cloud Container Service for Kubernetes (ACK) and how to use the agent to run a build plan for an application. In this topic, an example application complied in Java is created and deployed to a Kubernetes cluster.

Prerequisites

- A Kubernetes cluster is created by using ACK. For more information, see *Create a Kubernetes cluster*.
- A Bamboo server is created.

Source code of the application to be deployed

The source code of the application created in this topic can be obtained with the following address:

```
https :// github . com / AliyunCont  ainerServi  ce / jenkins - demo
. git
```

After you access the corresponding GitHub page, you can find the source code in the `bamboo` branch.

Deploy a remote agent in a Kubernetes cluster

1. Create a kaniko-docker-cfg secret.

Note:

This secret is used to set the permissions for accessing the target image repository required by building tasks in the remote agent and using kaniko to push a container image.

a. Log on to your Linux server by using the root account to run the following command to create a `/ root /. docker / config . json` file:

```
docker   login   registry . cn - hangzhou . aliyuncs . com
```

b. Use *Cloud Shell* to connect to the target Kubernetes cluster, and then run the following command to create a kaniko-docker-cfg secret:

```
kubectl  - n   bamboo   create   secret   generic   kaniko -
docker - cfg   -- from - file =/ root /. docker / config . json
```

2. Create a Bamboo agent in the target Kubernetes cluster.

`ServiceAcc ount` and `ClusterRol eBinding` are created to set the permissions required for kubectl to deploy an application to the target Kubernetes cluster.

a. Create a file *bamboo - agent . yaml* and copy the following code to the file:

```
apiVersion :  v1
kind :  ServiceAcc  ount
metadata :
  namespace :  bamboo
  name :  bamboo

---
apiVersion :  rbac . authorizat  ion . k8s . io / v1
kind :  ClusterRol  eBinding
metadata :
  name :  bamboo - cluster - admin
subjects :
 -  kind :  ServiceAcc  ount
    name :  bamboo
    namespace :  bamboo
roleRef :
  kind :  ClusterRol  e
  name :  cluster - admin
  apiGroup :  rbac . authorizat  ion . k8s . io

---
apiVersion :  apps / v1beta2
kind :  Deployment
metadata :
  name :  bamboo - agent
spec :
  replicas :  1
  selector :
    matchLabel  s :
      app :  bamboo - agent
  template :
    metadata :
```

```
      labels :
        app :  bamboo - agent
    spec :
      serviceAcc  ountName :  bamboo
      containers :
     -  name :  bamboo - agent
        env :
       -  name :  BAMBOO_SER  VER_URL
          value :  http :// xx . xx . xx . xx : 8085
        image :  registry . cn - hangzhou . aliyuncs . com /
haoshuwei / docker - bamboo - agent : v1
        imagePullP  olicy :  Always
        volumeMoun  ts :
         -  mountPath : / root /. docker /
            name :  kaniko - docker - cfg
      volumes :
       -  name :  kaniko - docker - cfg
          secret :
            secretName :  kaniko - docker - cfg
```

b. Run the `kubectl - n bamboo apply - f bamboo - agent . yaml` command to create the Bamboo agent.

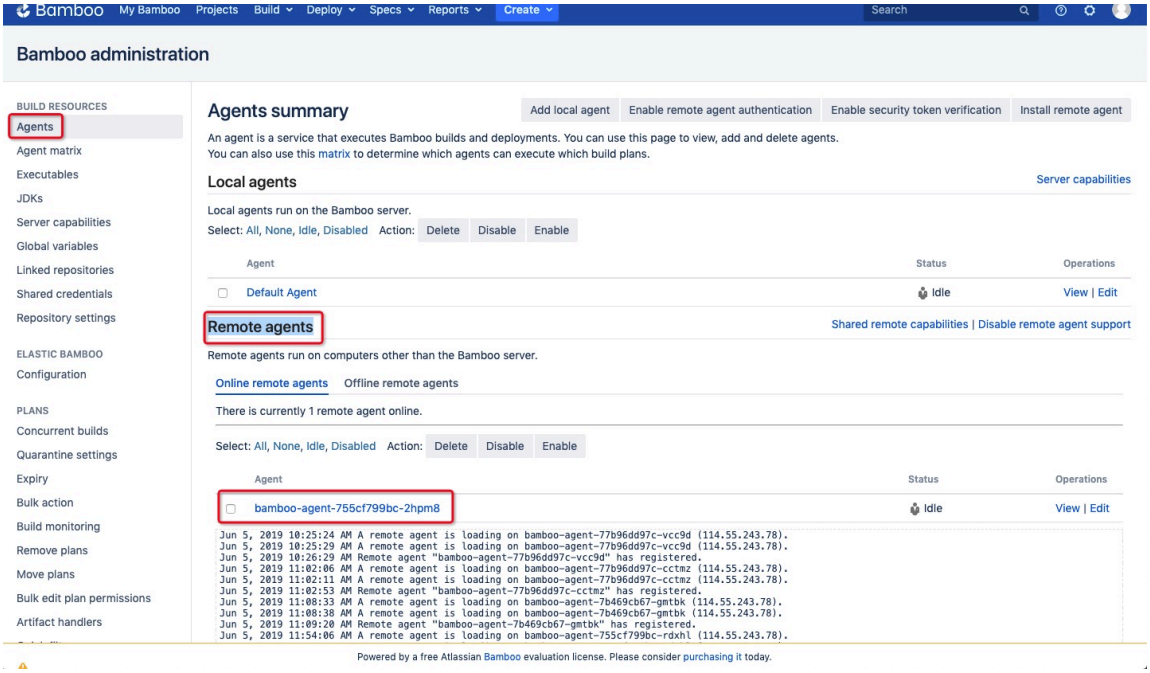
c. Run the following command to view logs of the agent:

```
kubectl - n bamboo logs - f < bamboo agent pod name
>
```

 **Note:**

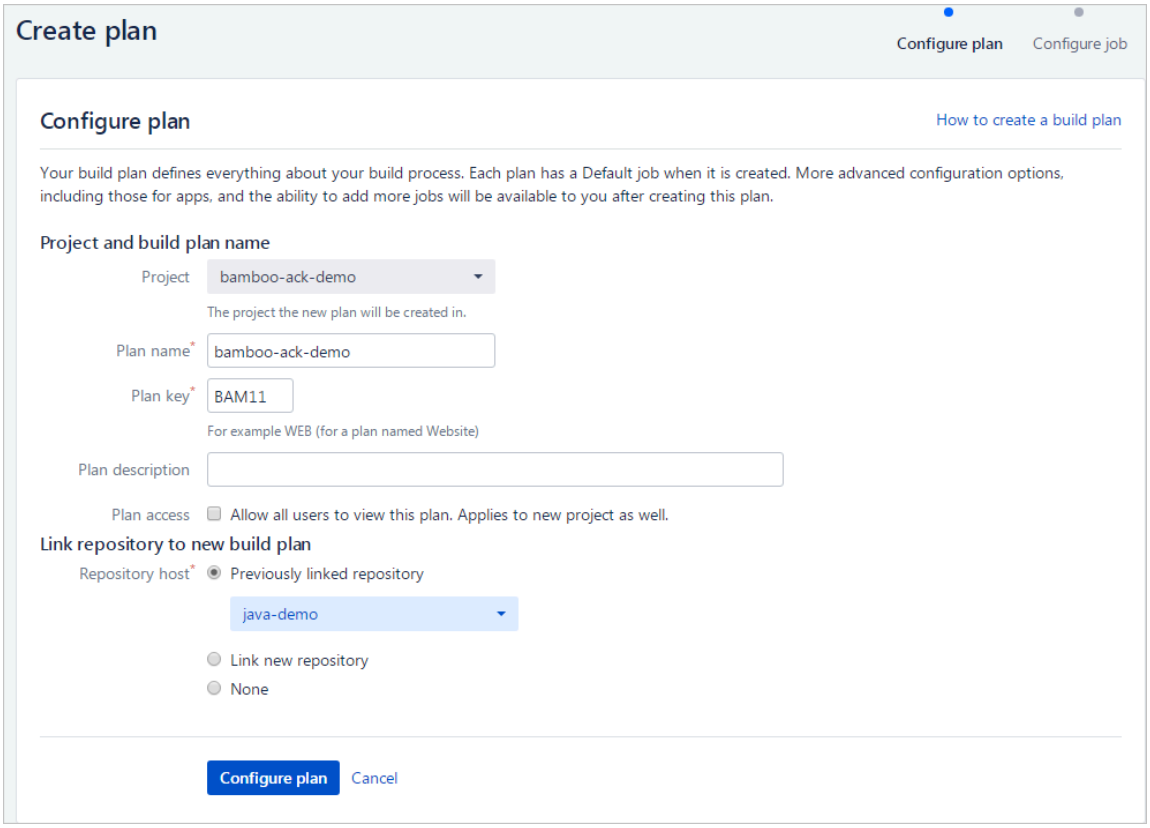
You must replace `< bamboo agent pod name >` with the file name that you use.

d. Log on to the Bamboo server you created. Then, in the upper-right corner, click , and select Agent to view the deployed agent.



Configure a build plan

1. Create a build plan.
   a. Log on to the Bamboo server you created, and choose Create > Create plan.
   b. 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, and then click Configure plan.

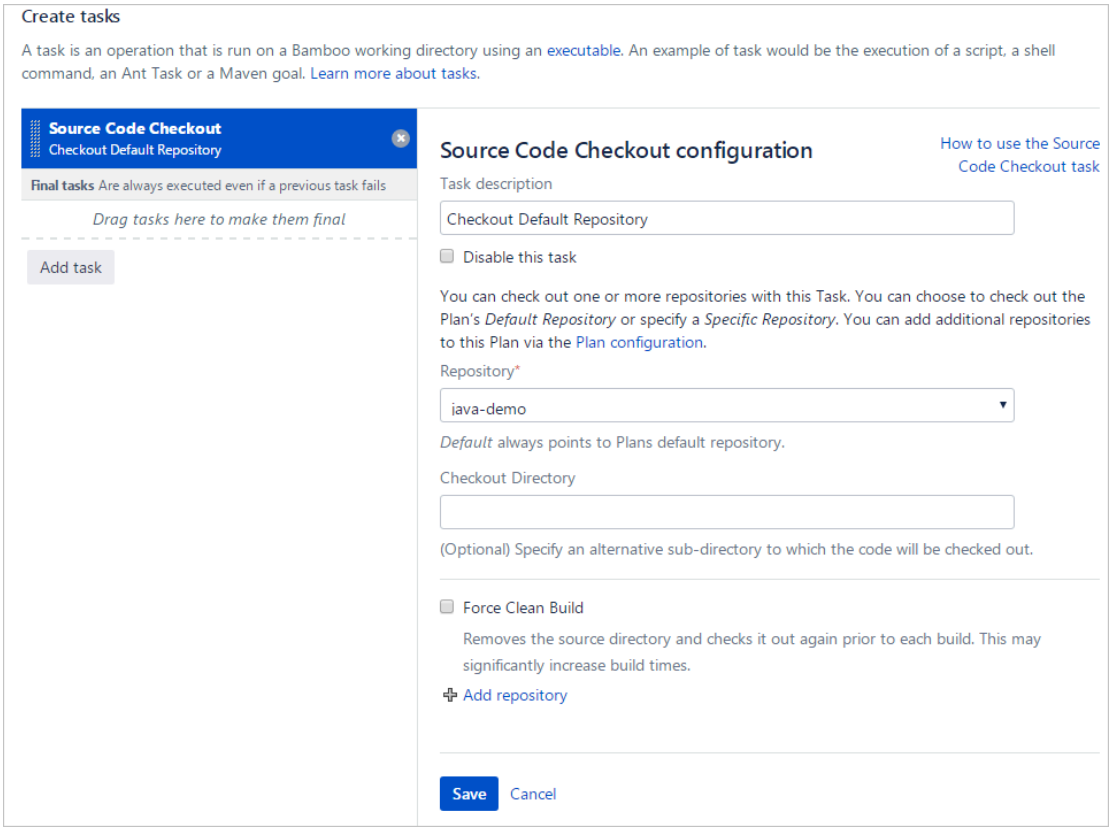


2. Configure a job that contains four required tasks for the build plan.
   a. Confirm and save the setting of the source code repository.

> **Note:**

In the preceding step where you create the build plan, your setting for Repository host specifies the source code repository. You can retain or modify this setting.

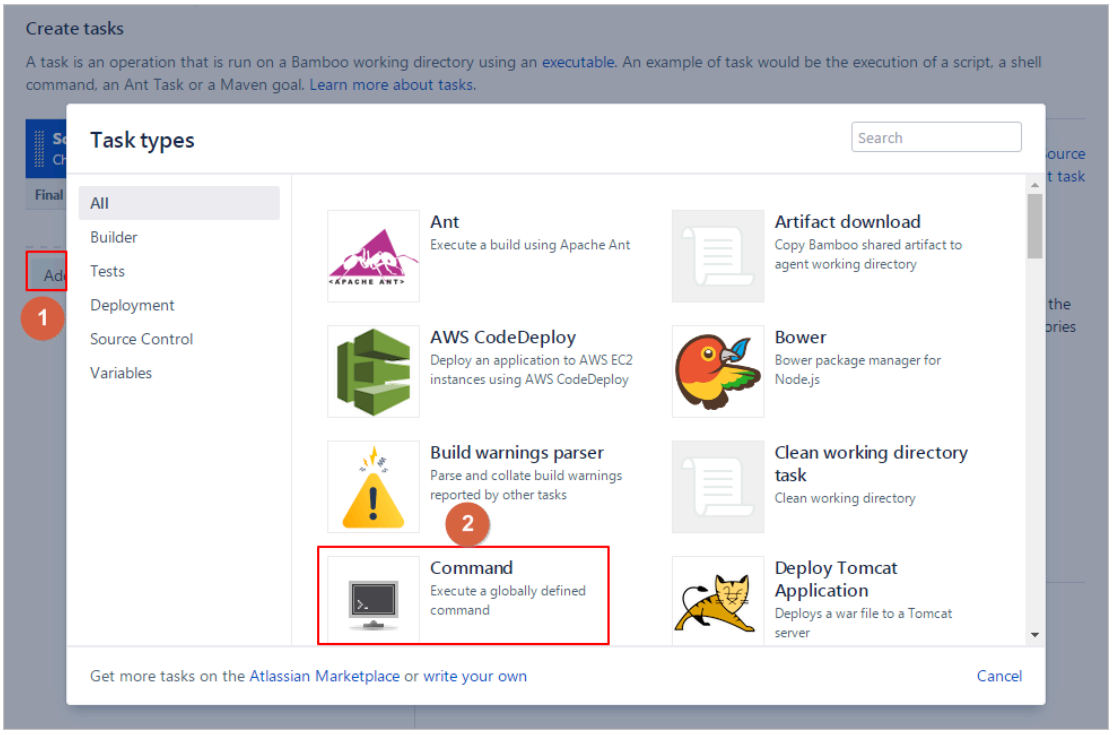
A. In the Create tasks area, click Source Code Checkout.



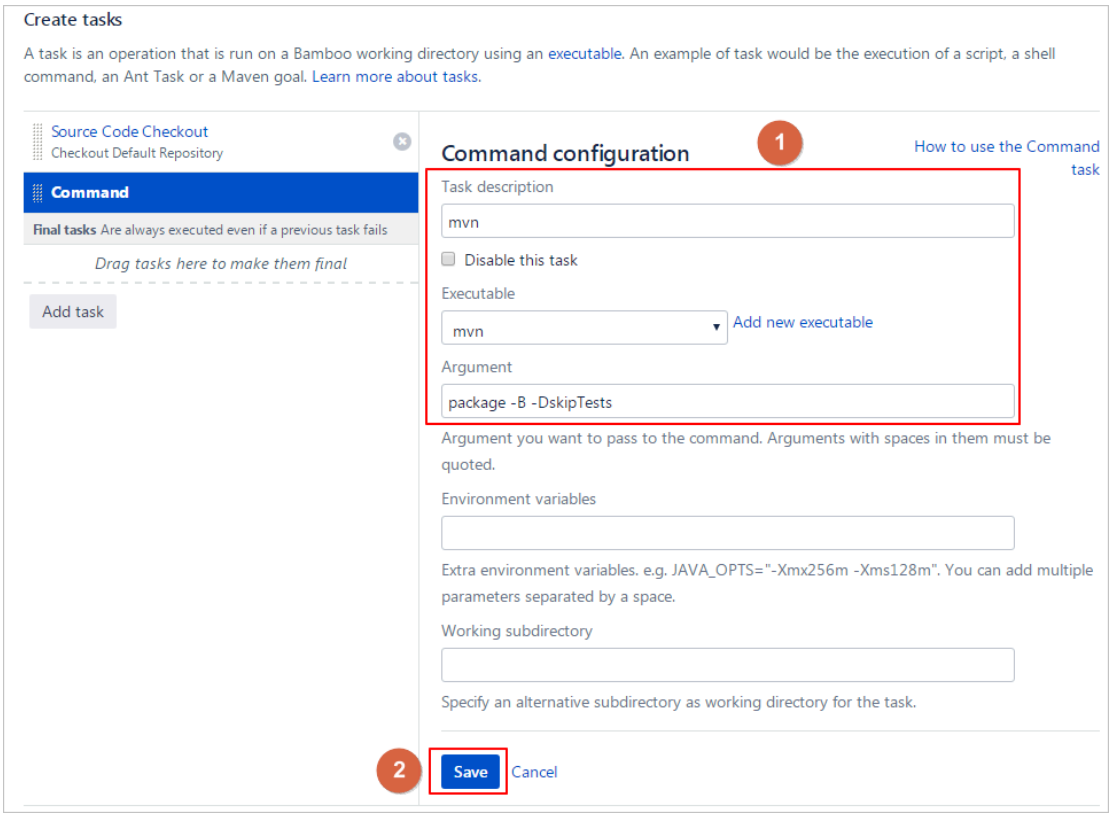
B. On the Source Code Checkout configuration page, select a new repository from the Repository drop-down list, and then click Save.

b. Add the command type of mvn to use the mvn tool.

A. In the Create tasks area, click Add task. Then, on the displayed Task types page, click Command.

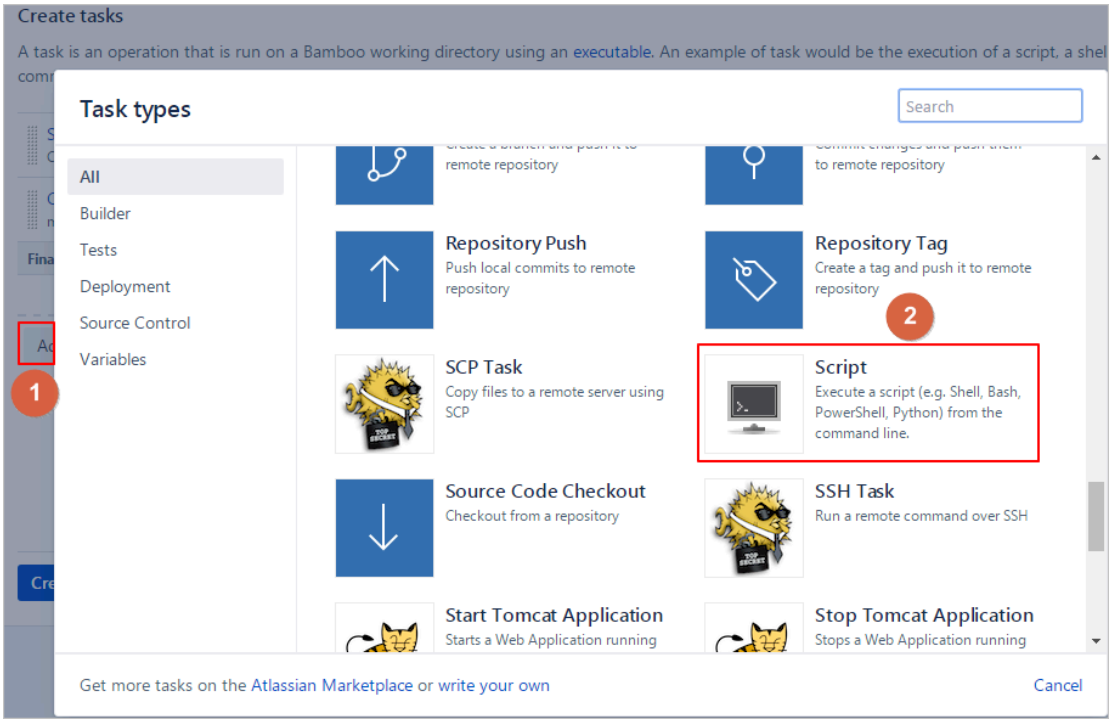


B. On the Command configuration page, set Task description, Executable, and Argument, and then click Save.



c. Use kaniko to package and push the required container image to the target image repository.

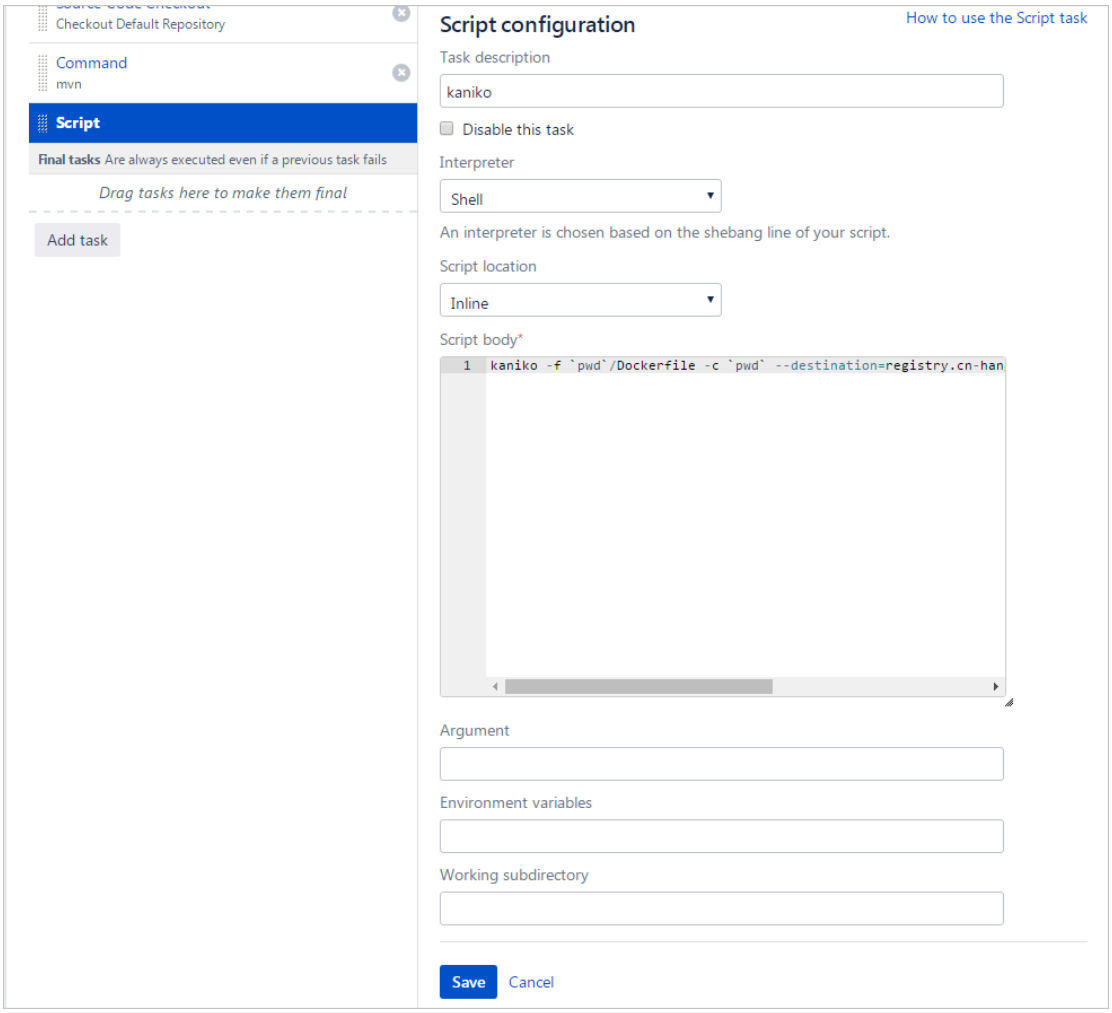
A. In the Create tasks area, click Add task. Then, on the displayed Task types page, click Script.



B. On the Script configuration page, set Task description, and Script location, then click Save.

Note:

For the remaining parameters of the script, you can retain the default settings.

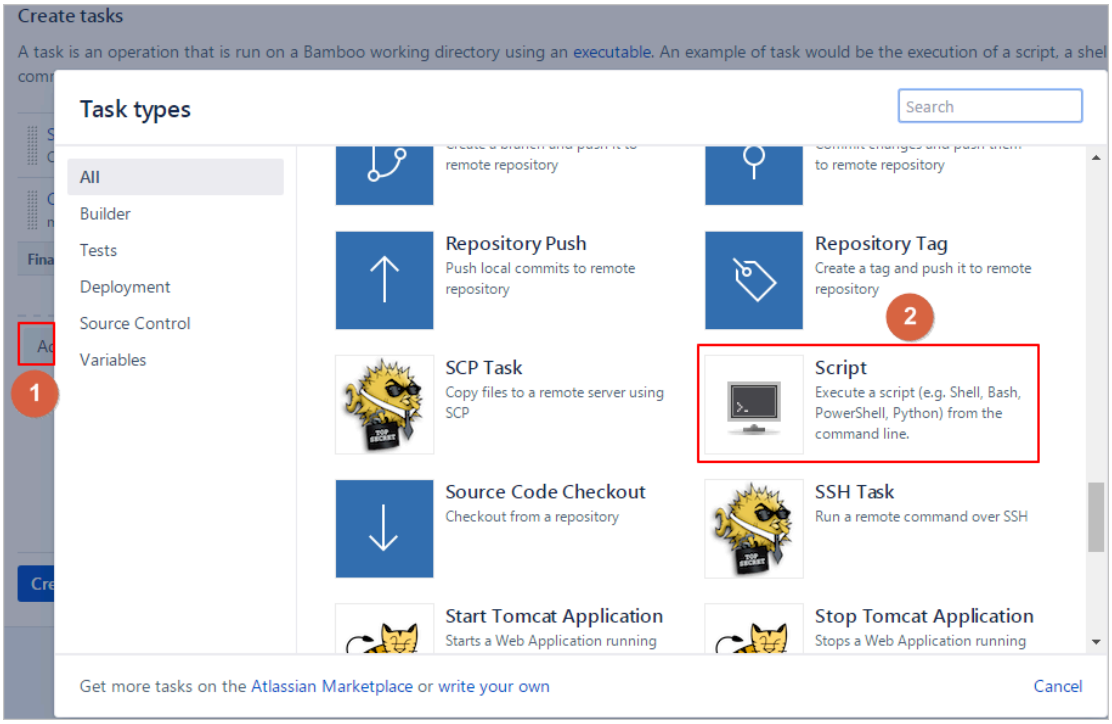


In this example, the Script location is set as follows:

```
kaniko -f `pwd`/Dockerfile -c `pwd` --destinatio
n=registry.cn-hangzhou.aliyuncs.com/haoshuwei/
bamboo-java-demo:latest
```

d. Use kubectl to deploy the application on the target Kubernetes cluster.

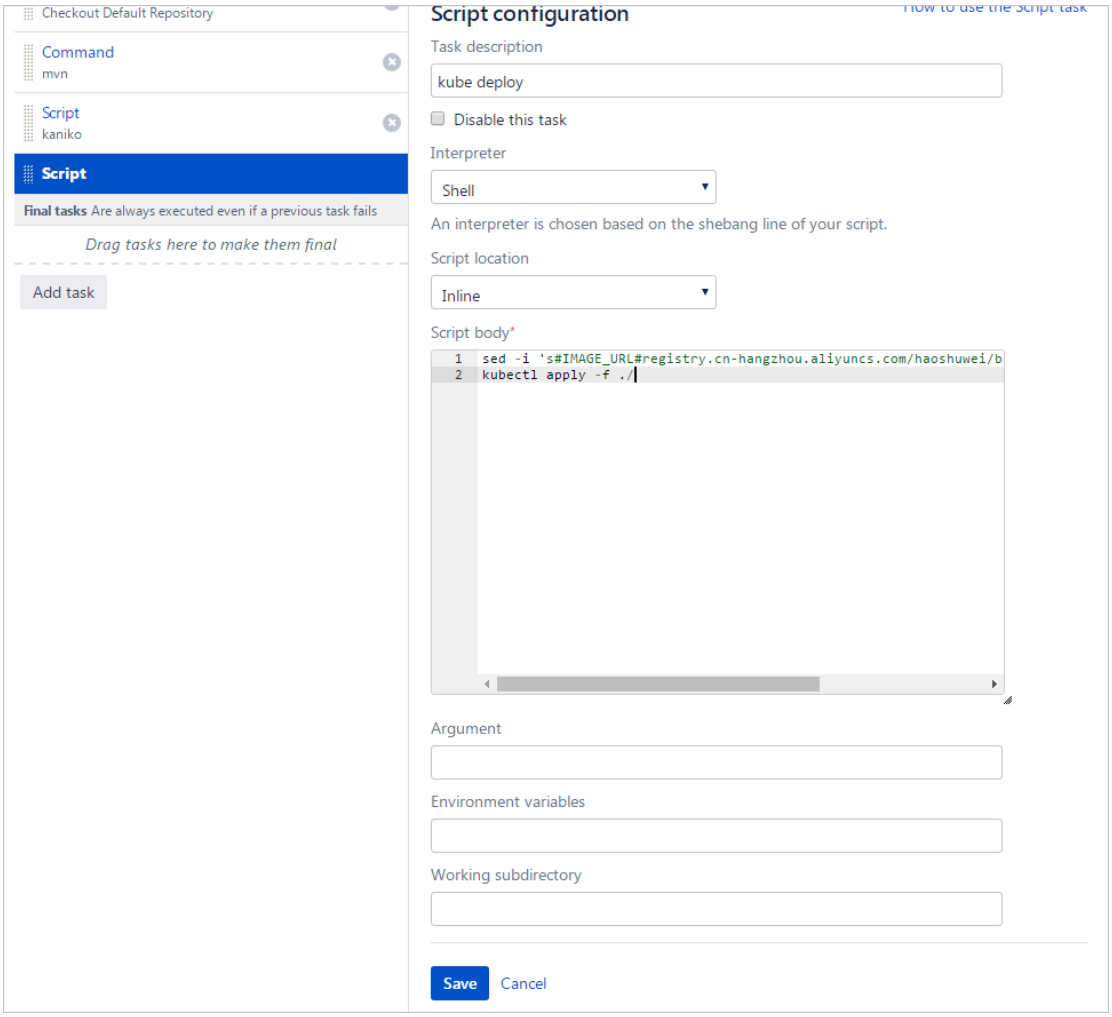
A. In the Create tasks area, click Add task. Then, on the displayed Task types page, click Script.



B. On the Script configuration page, set Task description and Script location, then click Save.

Note:

For the remaining parameters of the script, you can retain the default settings.



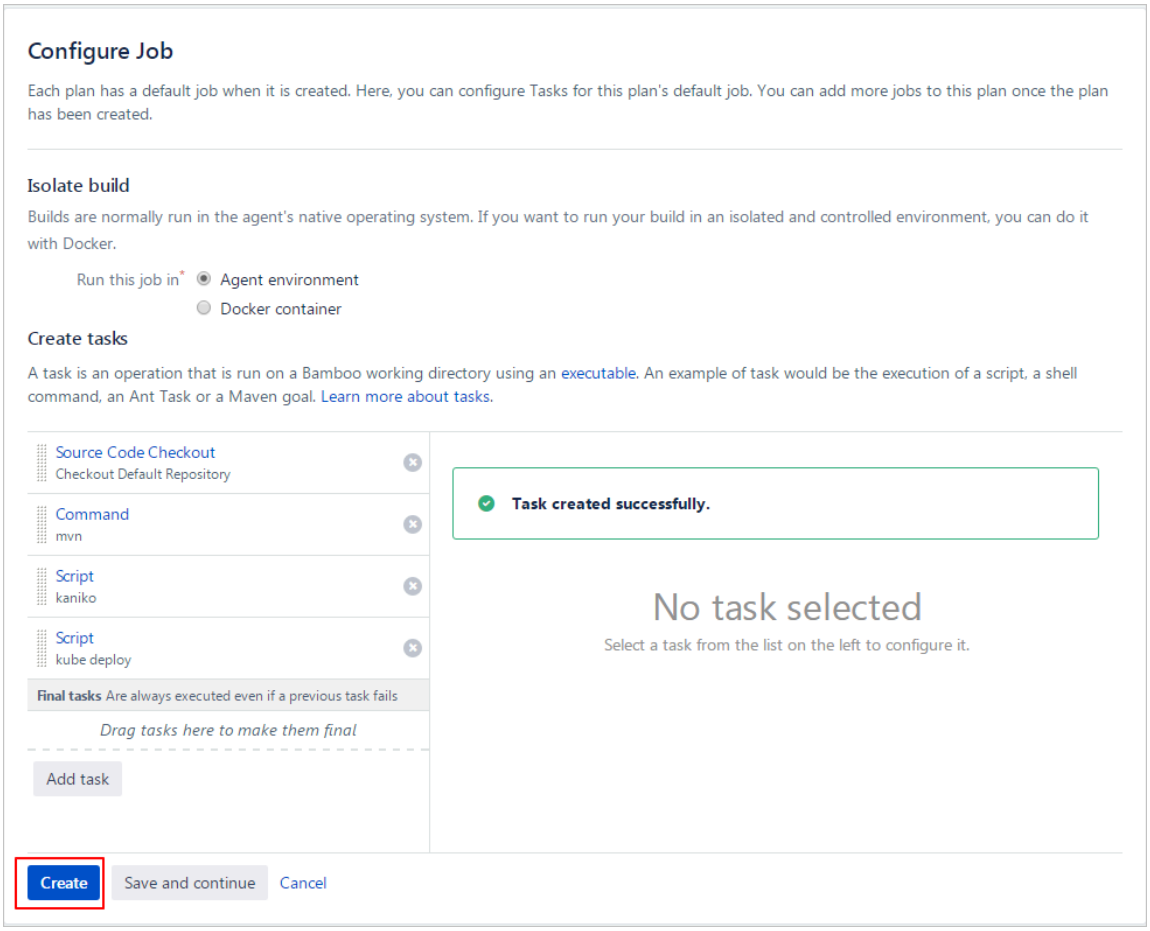
In this example, the Script location is set as follows:

```
sed  -i  's#IMAGE_URL#registry.cn-hangzhou.
aliyuncs.com/haoshuwei/bamboo-java-demo:latest
#' ./*.yaml
```

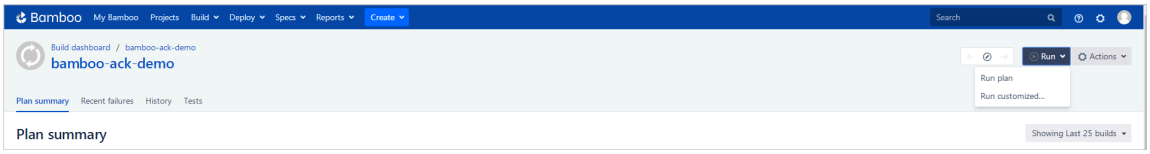
```
kubectl apply -f ./
```

3. Run the build plan.

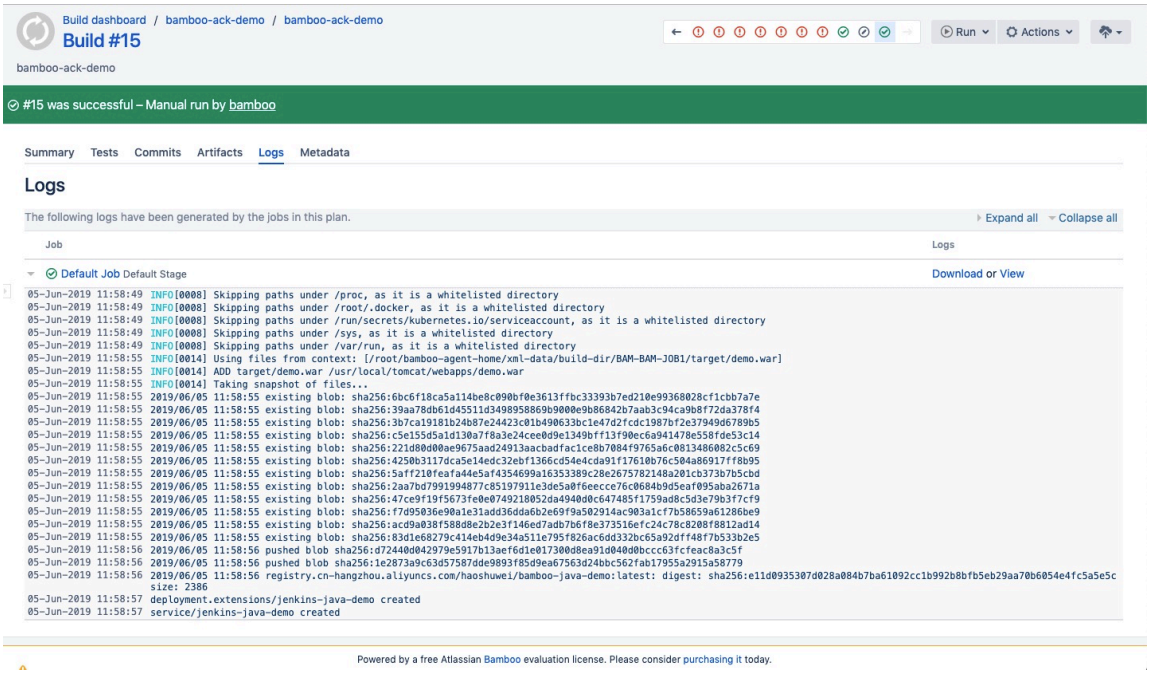
   a. On the Configure Job page, click Create.



   b. In the upper-right corner of the page, choose Run > Run plan.



   You can click the Logs tab to view the logs.

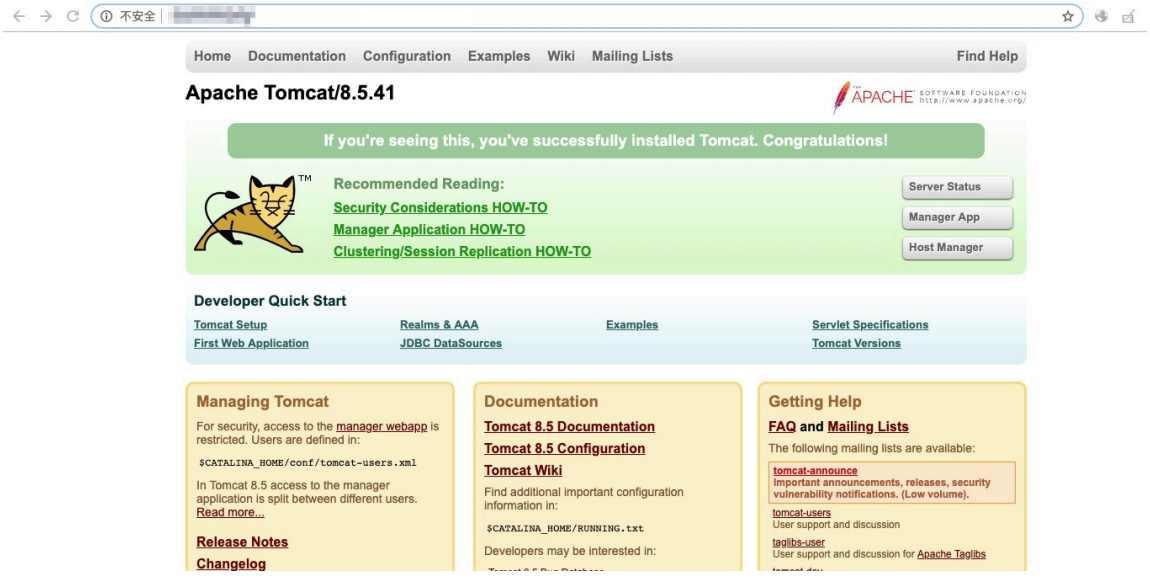


4. **Access the deployed application.**

   a. **Run the `kubectl -n bamboo get svc` command to view the Internet IP address of the application.**

```
[root@iZbp12i73koztp1cz75skaZ bamboo]# kubectl -n bamboo get svc
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
```

   b. **In your browser, enter *http://EXTERNAL-IP* to access the application.**



Reference

For information about how to create the `registry.cn-hangzhou.aliyuncs.com/haoshuwei/docker-bamboo-agent:v1` image, see *Docker bamboo agent*.

For more information, see *Bamboo*.