Alibaba Cloud
Container Service for Kubernetes

quickstart

Issue: 20181008

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.	 **Note:** 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]*
{} or {a|b}	It indicates that it is a required value, and only one item can be selected.	`swich` *{stand | slave}*

Contents

1 Basic operations

1.1 Create a cluster quickly

Prerequisites

Activate the following services: Container Service, Resource Orchestration Service (ROS), and Resource Access Management (RAM). For more information about limits and instructions, see *Create a Kubernetes cluster*.

Log on to the *Container Service console*, *ROS console*, and *RAM console* to activate the corresponding services.

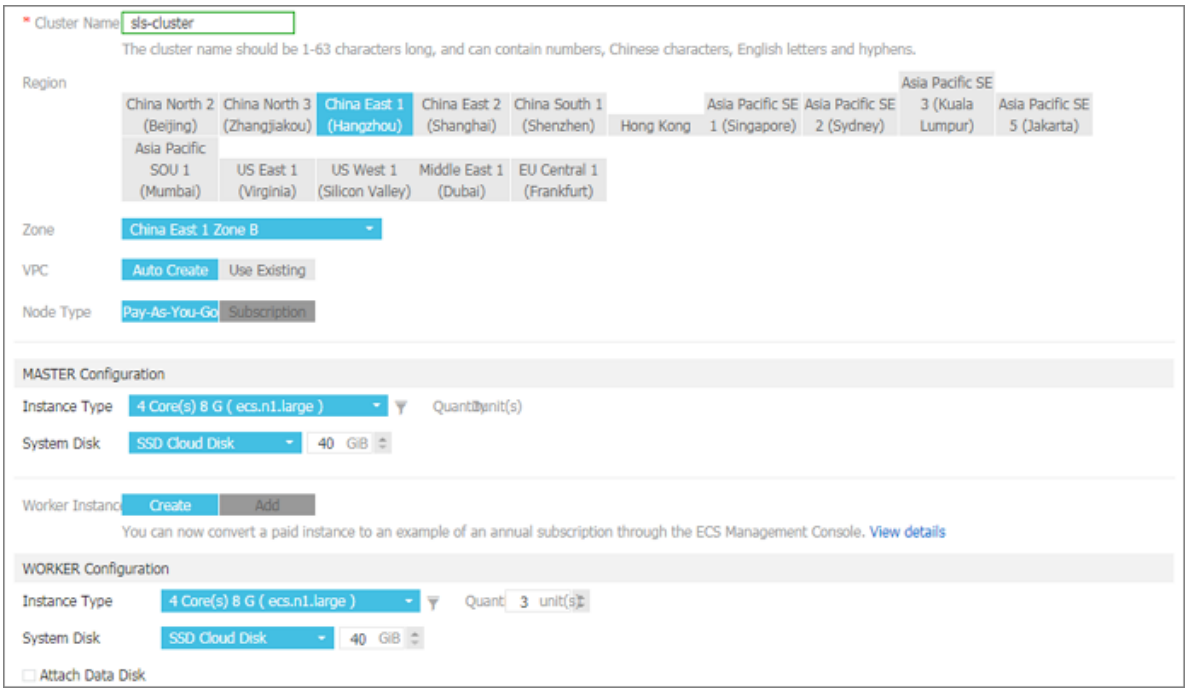
Context

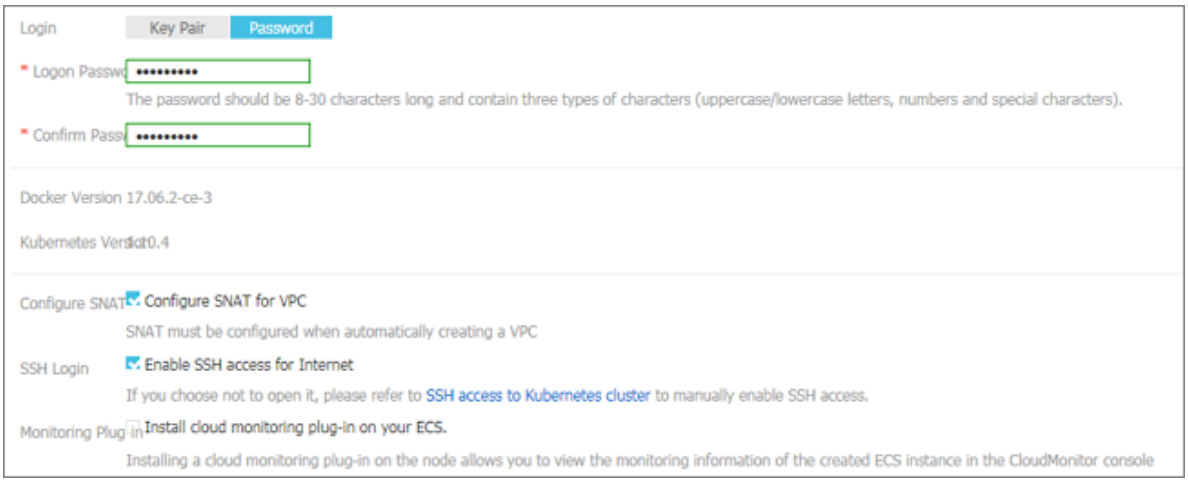
This example shows how to quickly create a Kubernetes cluster. Some configurations use the default or the simplest configuration.

Procedure

1. Log on to the *Container Service console*.
2. Under Kubernetes, click **Cluster** in the left-side navigation pane to enter the cluster list page. Click **Create Kubernetes Cluster** in the upper-right corner of the page.
3. Configure cluster settings.

   Most of the configurations in this example retain the default values. The specific configuration is shown in the following figure:





Configuration	Description
Cluster name	The name can be 1 to 63 characters long and contain numbers, Chinese characters, English letters, and hyphens (-),
Region and zone	The region and zone in which the cluster resides.
Virtual Private Cloud (VPC)	You can select **Auto Create** to create a Virtual Private Cloud (VPC) together with the Kubernetes cluster or **Use existing** to use an existing VPC.

Configuration	Description
	• With **Auto Create** selected, the system automatically creates a NAT gateway for your VPC when the cluster is created. • With **Use Existing** selected, if the selected VPC already has a NAT gateway, Container Service uses the existing NAT gateway. Otherwise, the system automatically creates a NAT gateway by default. If you do not want the system to automatically create a NAT gateway, clear the **Configure SNAT for VPC** check box. **Note:** If you select to not automatically create a NAT gateway, configure the NAT gateway on your own to implement the VPC public network environment with secure access, or manually configure the SNAT. Otherwise, instances in the VPC cannot access public network normally, which leads to cluster creation failure.
Node type	Pay-As-You-Go and Subscription are supported.
Master node	Select an instance type and system disk: • **Instance type**: see*Instance type families* • **System disk**: SSD and high-efficiency cloud disks are supported.
Worker node configuration	Select whether to create a worker node or add an existing ECS instance as the worker node. If you choose to add an instance, you can configure it as follows. • **Instance type**: see*Instance type families* • **System disk**: SSD and high-efficiency cloud disks are supported. • **Mount data disk**: SSD, high-efficiency, and basic cloud disks are supported.
Logon mode	Support setting keys and passwords. For information about using keys to log on,

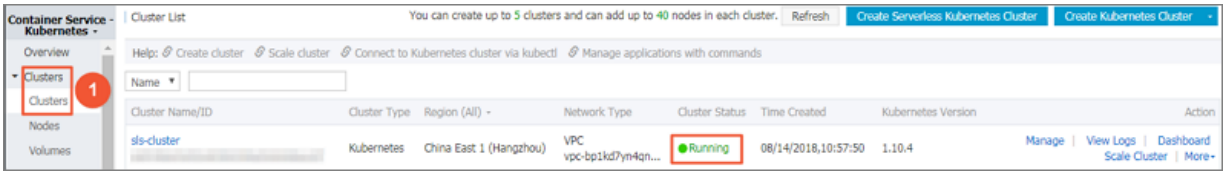
Configuration	Description
	see*Access Kubernetes clusters by using SSH key pairs*
Pod Network CIDR and Service CIDR (optional)	For more information about specific plan, see *Plan Kubernetes CIDR blocks under VPC*. **Note:** This option is available when you select to **use an existing** VPC.
SNAT	SNAT must be configured if you select **Auto Create** VPC. If you select **Use existing** VPC, you can select whether to automatically configure SNAT gateway. If you select not to configure SNAT automatically, configure the NAT gateway or configure SNAT manually.
SSH Logon	• If you enable SSH logon for Internet, you can access the cluster by using SSH. • If you do not enable SSH logon for Internet, you cannot access the cluster by using SSH or connect to the cluster by using kubectl. You can manually enable SSH access. For more information, see *Access Kubernetes clusters by using SSH*.
Cloud monitoring plug-in	Install the cloud monitoring plug-in on the ECS instance and then you can view the monitoring information of the created ECS instance in the CloudMonitor console.
RDS instance whitelist (optional)	Add the IP addresses of the ECS instances to the RDS instance whitelist. **Note:** This option is available if you are **using an existing** VPC.
Advanced configurations	• Network plug-ins, Flannel and Terway network plug-ins are supported. By default , the Flannel network plug-in is used. • Number of pods for a node: Maximum number of pods that can be run by a single node.

Configuration	Description
	• Custom image: Select whether or not to use the custom image. The ECS instance installs the default CentOS version if no custom image is selected. • Custom cluster CA: Select whether to use the custom cluster CA.

4. Click **Create Cluster** in the upper-right corner.

What's next

After the cluster is created, you can view the cluster in the Kubernetes cluster list of the Container Service console.



Now you have quickly created a Kubernetes cluster.

1.2 Create an application by using an image

Prerequisites

Create a Kubernetes cluster. For more information, see *#unique_11*.

Procedure

1. Log on to the *Container Service console*.
2. Under Kubernetes, click **Application** > **Deployment** in the left-side navigation pane. Enter the Deployment List page and click **Create by image** in the upper-right corner.
3. Enter the application **Name**, then select the **Cluster** and **Namespace**. Click **Next** to go to the Configuration step.

   By default, the system uses the default namespace if the **namespace** is not configured.

4. Configure the general settings for the application.
   - **Image name**: You can click **Select image** to select the image in the displayed dialog box and then click **OK**. In this example, the image name is nginx.

You can also enter the private registry in the format of `domainname/namespace/imagename:tag`.

- **Image version**: Click **Select image version** to select the version. If the image version is not specified, the system uses the latest version by default.
- **Scale**: Specify the number of containers. In this example, only one container is in the pod. If multiple containers are specified, the same number of pods will be started.

5. Configure the resource limit and resource reserve for the container.

   - **Resource Limit**: Specify the upper limit for the resources (CPU and memory) that can be used by this application to avoid occupying excessive resources.
   - **Resource Request**: Specify how many resources (CPU and memory) are reserved for the application, that is, these resources are exclusive to the container. Other services or processes will compete for resources when the resources are insufficient. By specifying the Resource Request, the application will not become unavailable because of insufficient resources.

   CPU is measured in millicores (one thousandth of one core). Memory is measured in bytes, which can be Gi, Mi, or Ki.

6. Configure the data volumes.

   Local storage and cloud storage can be configured.

   - **Local storage**: Supports hostPath, configmap, secret, and temporary directory. The local data volumes mount the corresponding mount source to the container path. For more information, see *volumes*.
   - **Cloud storage**: Supports three types of cloud storage: cloud disk, Network Attached Storage (NAS), and Object Storage Service (OSS).

   In this example, a data volume of cloud disk is configured. Mounting the cloud disk to the `/tmp` container path stores data generated in this path to the cloud disk.

7. Configure the environment variable.

You can configure the environment variable for the pod in the format of key-value pairs to add the environment label or pass the configurations for the pod. For more information, see *Pod variable*.

8. Configure the container.

   You can configure the Command, Args, and Container Config for the container running in the pod.

   - **Command** and **Args**: If not configured, the default settings of the image are used. If configured, the default settings of the image are overwritten. If only the Args is configured, the default command will run the new arguments when the container is started. Command and Args cannot be modified after the pod is created.
   - **Container Config**: Select the stdin check box to enable standard input for the container. Select the tty check box to assign an virtual terminal to send signals to the container. These two options are usually used together, which indicates to bind the terminal (tty) to the container standard input (stdin). For example, an interactive program obtains standard input from you and then displays the obtained standard input in the terminal.

9. Configure **health check**

   The health check function includes liveness probes and readiness probes. Liveness probes are used to detect when to restart the container. Readiness probes determine if the container is ready for receiving traffic. For more information about health checks, see *https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes*.

Request method	Configuration description
HTTP request	An HTTP GET request is sent to the container. The following are supported parameters: • Protocol: HTTP/HTTPS • Path: Path to access the HTTP server • Port: Number or name of the port exposed by the container. The port number must be in the range of 1 to 65535. • HTTP Header: Custom headers in the HTTP request. HTTP allows repeated headers. Supports the correct configurat ion of key values.

Request method	Configuration description
	• Initial Delay (in seconds): Namely, the **initialDelaySeconds**. Seconds for the first liveness or readiness probe has to wait after the container is started. • Period (in seconds): Namely, the **periodseconds**. Intervals at which the probe is performed. The default value is 10 seconds. The minimum value 1 second. • Timeout (in seconds): Namely, the **timeoutSeconds**. The time of probe timeout. The default value is 1 second and the minimum value is 1 second. • Success Threshold: The minimum number of consecutive successful probes that are considered as successful after a failed probe. The default value is 1. **This parameter must be 1 for liveness**. The minimum value is 1. • Failure Threshold: The minimum number of consecutive failed probes that are considered as failed after a successful probe. The default value is 3. The minimum value is 1.
TCP connection	A TCP socket is send to the container. The kubelet attempts to open a socket to your container on the specified port. If a connection can be established, the container is considered healthy. If not, it is considered as a failure. The following are supported parameters: • Port: Number or name of the port exposed by the container. The port number must be in the range of 1 to 65535. • Initial Delay (in seconds): Namely, the **initialDelaySeconds**. Seconds for the first liveness or readiness probe has to wait after the container is started. • Period (in seconds): Namely, the **periodseconds**. Intervals at which the probe is performed. The default value is 10 seconds. The minimum value 1 second.

Request method	Configuration description
	• Timeout (in seconds): Namely, the **timeoutSeconds**. The time of probe timeout. The default value is 1 second and the minimum value is 1 second. • Success Threshold: The minimum number of consecutive successful probes that are considered as successful after a failed probe. The default value is 1. **This parameter must be 1 for liveness**. The minimum value is 1. • Failure Threshold: The minimum number of consecutive failed probes that are considered as failed after a successful probe. The default value is 3. The minimum value is 1.
Command line	Detect the health of the container by executing probe detection commands in the container. The following are supported parameters: • Command: A probe command used to detect the health of the container. • Initial Delay (in seconds): Namely, the **initialDelaySeconds**. Seconds for the first liveness or readiness probe has to wait after the container is started. • Period (in seconds): Namely, the **periodseconds**. Intervals at which the probe is performed. The default value is 10 seconds. The minimum value 1 second. • Timeout (in seconds): Namely, the **timeoutSeconds**. The time of probe timeout. The default value is 1 second and the minimum value is 1 second. • Success Threshold: The minimum number of consecutive successful probes that are considered as successful after a failed probe. The default value is 1. **This parameter must be 1 for liveness**. The minimum value is 1.

Request method	Configuration description
	• Failure Threshold: The minimum number of consecutive failed probes that are considered as failed after a successful probe. The default value is 3. The minimum value is 1.

10. Select whether or not to enable the **Auto Scaling**.

    You can choose whether to enable **Auto Scaling**. To meet the demands of applications under different loads, Container Service supports the container auto scaling, which automatically adjusts the number of containers according to the container CPU and memory usage.

    Note:
    To enable auto scaling, you must configure required resources for the deployment. Otherwise, the container auto scaling cannot take effect.

    - **Metric**: CPU and memory. Configure a resource type as needed.
    - **Condition**: The percentage value of resource usage. The container begins to expand when the resource usage exceeds this value.
    - **Maximum number of containers**: The maximum number of containers that the deployment can expand to.
    - **Minimum number of containers**: The minimum number of containers that the deployment can shrink to.

11. Click **Next** after completing the configurations.
12. In the Access Control step, configure a service to bind with the backend pods. Click **Create** after the access control configurations.

    - **Service**: Select None to not create a service, or select a service type as follows:
      - ClusterIP: Exposes the service by using the internal IP address of your cluster. With this type selected, the service is accessible only within the cluster.
      - NodePort: Exposes the service by using the IP address and static port (NodePort) on each node. A ClusterIP service, to which the NodePort service is routed, is automatically created. You can access the NodePort service from outside the cluster by requesting `<NodeIP>:<NodePort>`.

- Server Load Balancer: Exposes the service by using Server Load Balancer, which is provided by Alibaba Cloud. Select public or inner to access the service by using the Internet or intranet. Server Load Balancer can route to the NodePort and ClusterIP services.

- **Name**: By default, a service name composed of the application name and the suffix svc is generated. In this example, the generated service name is nginx-default-svc. You can modify the service name as needed.
- **Port Mapping**: You must add the service port and the container port. If NodePort is selected as the service type, you must configure the node port to avoid the port conflict. Select TCP or UDP as the Protocol.

13. The Done step indicating the successful creation appears. The objects contained in the application are displayed. You can click **View** to view the deployment list.

14. The newly created deployment nginx-default-deployment is displayed on the Deployment page.

15. Click **Application** > **Service** in the left-side navigation pane. The newly created service nginx-default-svc is displayed on the Service List page.

16. Access the external endpoint in the browser to access the Nginx welcome page.

1.3 Use Yaml to create a statefull tomcat application

Prerequisites

- Create a Kubernetes cluster. For more information, see *Create a cluster quickly*.
- You have created the resource objects involved in this example, such as storage volumes, config maps, secrets, node labels, and other resource objects.

Context

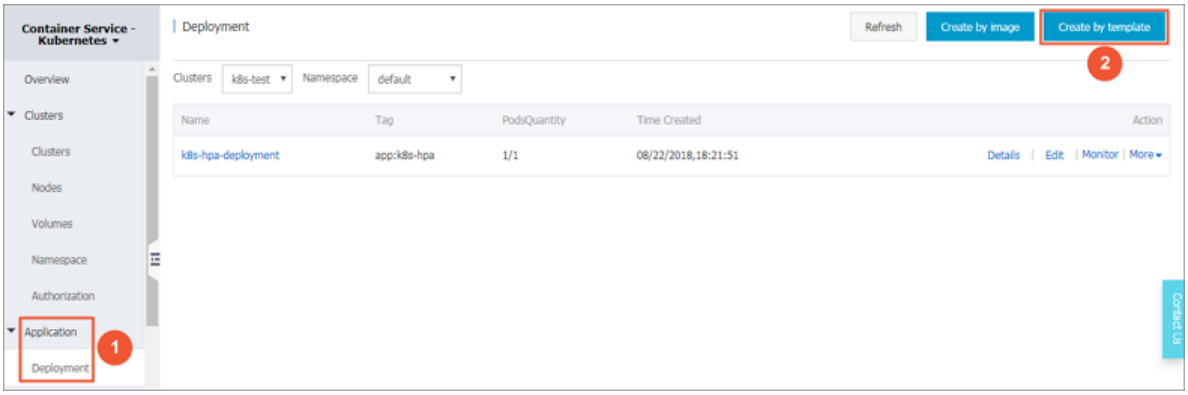
In a Container Service Kubernetes orchestration template, you must define resource objects required for running an application, and combine the resource objects into a complete application by using label selector.

This example shows how to create a tomcat application by customizing a template in an orchestrat ion template. The resource objects involved are as follows:

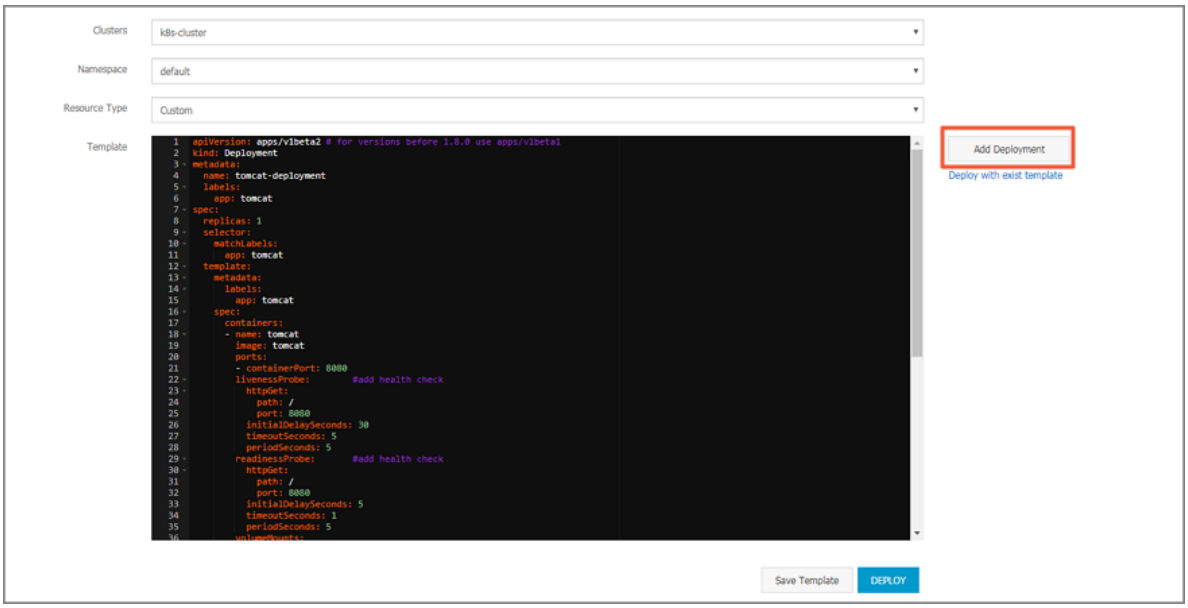
1. Storage volumes
2. Config maps
3. Secrets
4. Nodes specified by labels
5. Health check
6. Server/Load Balancer

Procedure

1. Log on to the *Container Service console*.
2. Under Kubernetes, click **Application** > **Deployment** in the left-side navigation pane.
3. On the Deployment page, click **Create by template** in the upper-right corner.



4. Configure the template. **Customize the template** to create a tomcat application.
   - **Clusters**: Select a cluster. Resource objects are to be deployed in this cluster.
   - **Namespace**: Select a namespace to which resource objects belong. The default namespace is default. Except for the underlying computing resources such as nodes and persistent storage volumes, most of the resource objects must act on a namespace.
   - **Resource Type**: Alibaba Cloud Container Service provides multiple resource types of Kubernetes yaml sample templates, enabling you to quickly deploy resource objects. You can write a template based on the format requirements of Kubernetes yaml orchestration to describe the resource type you want to define.
   - **Add Deployment**: If you are not familiar with Kubernetes yaml orchestration, click **Add Deployment** to configure through the web interface.



a) First create a basic tomcat template on which this example shows how to configure resource objects in a yaml file.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly &lt;
image_name:tags>
        ports:
        - containerPort: 8080
```

b) Add a storage volume based on the basic template

Before adding a data volume, apply for a storage volume and create the storage volume claim. You can apply for a storage volume by using one of the following methods: *Use Alibaba Cloud cloud disks*, *Use Alibaba Cloud NAS*, and *Use Alibaba Cloud OSS*.

Create the storage volume claim after applying for a storage volume. For more information, see *Create a persistent storage volume claim*. In this example, use Alibaba Cloud cloud

disks as the storage volumes and use the cloud disk static storage volumes by using PV/PVC. The PVC name is pvc-yunpan-test.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly &lt;
image_name:tags>
        ports:
        - containerPort: 8080
        volumeMounts:                  #add volume
        - name: pvc-yunpan-test
          mountPath: /data
      volumes:                         #add volume
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
```

c) Add a config map

Before using a config map, create a config map. For information about creating and using a config map, see *Use a config map in a pod*.

In this example, use the config map name and content in the following sample. The config map name is special-config. The config maps are `SPECIAL_LEVEL:very` and`SPECIAL_TYPE:charm`. Use config maps by means of environment variables.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
```

```
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly &lt;
image_name:tags>
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
        env:
        - name: SPECIAL_LEVEL_KEY #add configmap
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY #add configmap
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
```

d) Add a secret

Create a secret first. For more information, see *Create a secret*.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly &lt;
image_name:tags>
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
        env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
```

```
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
        - name: SECRET_USERNAME #add secret
          valueFrom:
            secretKeyRef:
              name: account
              key: username
        - name: SECRET_PASSWORD #add secret
          valueFrom:
            secretKeyRef:
              name: account
              key: password
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
```

e) Add a node

When you deploy an application, you can deploy the application on a node with the specific label. For instructions, see *Schedule a pod to a specified node*.

In this example, label a node with group:worker. When the application deployment succeeds , the application is deployed on the labeled node.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
        env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
```

```
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: account
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: account
              key: password
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
      nodeSelector:       #add node selector
        group: worker
```

f) Add health check

On Container Service platform, you can add health check for the application to check the health status of the application. Use liveness probes and readiness probes to detect the health status of a container in the application.

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat
        ports:
        - containerPort: 8080
        livenessProbe:        #add health check
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 30
          timeoutSeconds: 5
          periodSeconds: 5
        readinessProbe:       #add health check
```

```
          httpGet:
            path: /
            - port: 8080
          initialDelaySeconds: 5
          timeoutSeconds: 1
          periodSeconds: 5
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
        env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: account
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: account
              key: password
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
      nodeSelector:
        group: worker
```

g) Creates a LoadBalancer type service for the tomcat deployment.

To access applications deployed on Container Service from external networks such as the public network, you can expose the application by creating a LoadBalancer type service. A LoadBalancer type service creates Load Balancer on Alibaba Cloud. You can access the application through the Load Balancer IP address.

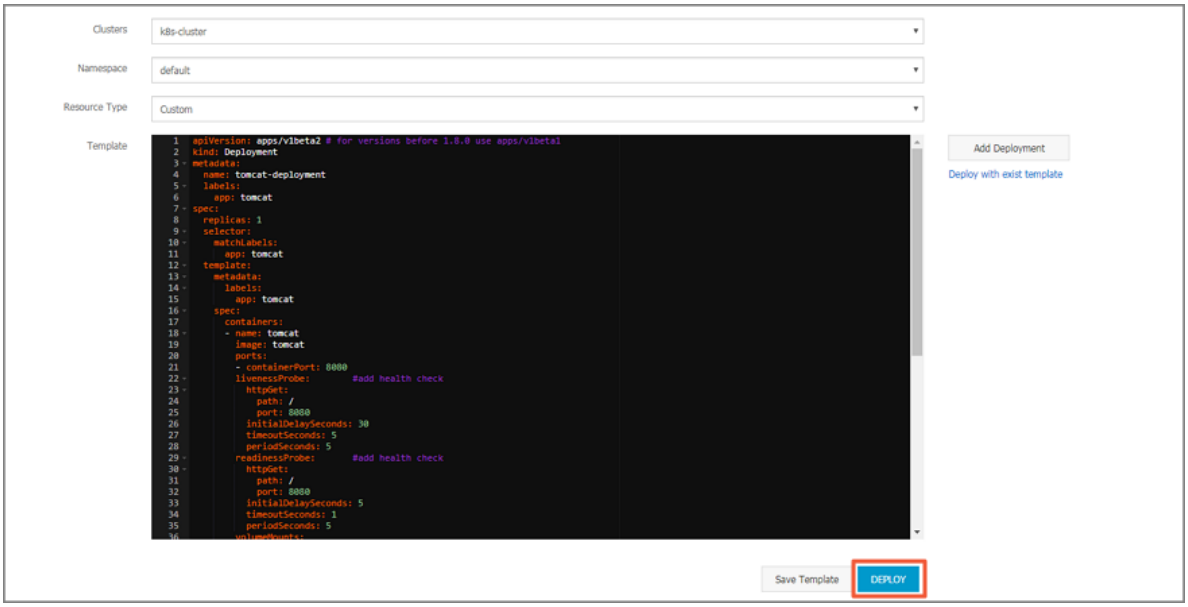
For information about creating a service, see *Create a service*.

In this example, the orchestration template is as follows:

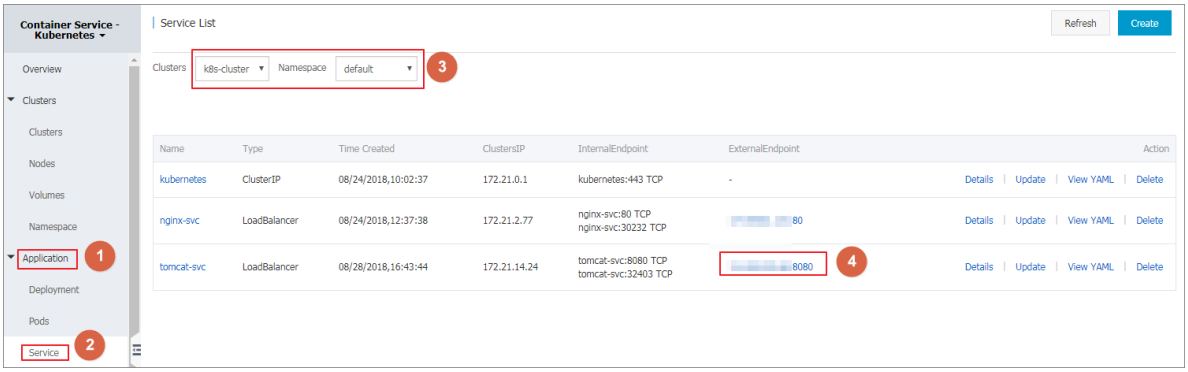
```
apiVersion: v1
kind: Service
metadata:
  name: tomcat-svc
  labels:
    app: tomcat-svc
spec:
  selector:
    app: tomcat
  ports:
```

```
  - protocol: TCP
    port: 8080
    targetPort: 8080
type: LoadBalancer
```

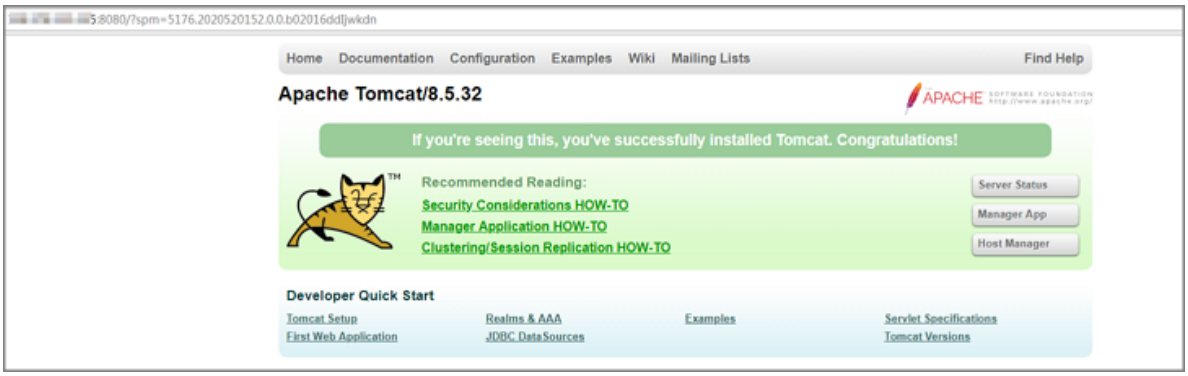
5. After the configuration is completed according to the application requirements, click **Create**.



6. After the deployment succeeds, click **Service** in the left navigation pane, and select the tomcat-svc service to view its external endpoint.



7. Entering the external endpoint in the browser address bar, you can access the tomcat app welcome page .



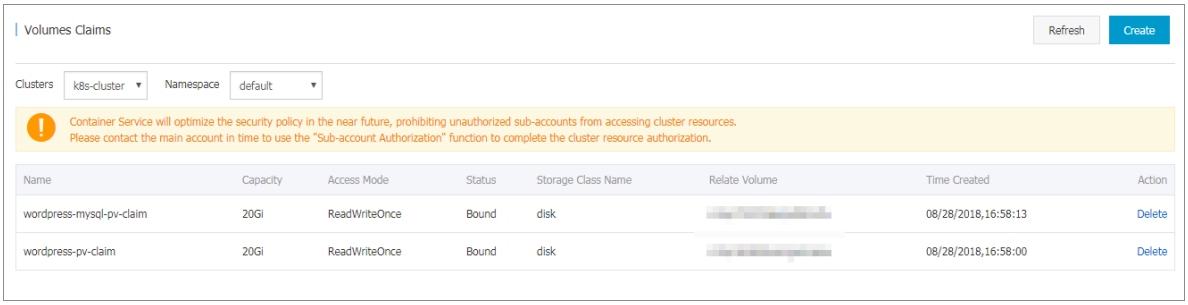
What's next

According to your orchestration template, you can explore features of the tomcat application in storage volumes, secrets, config maps, node scheduling, and health check.

1.4 Deploy dependency-based WordPress applications

Prerequisites

- Create a Kubernetes cluster. For more information, see *Create a cluster quickly*.
- Create storage volumes and storage volume claims. For how to create a storage volume, see *Use Alibaba Cloud cloud disks*, *Use Alibaba Cloud NAS*, and *Use Alibaba Cloud OSS*. For how to create a storage volume claim, see *Create a persistent storage volume claim*. Use Alibaba Cloud disks as storage volumes. In the example, choose PV/PVC for the storage volume mount. Create two storage volume claims: wordpress-pv-claim and wordpress-mysql-pv-claim which are used in the wordpress yaml file and the wordpress-mysql yaml file respectively, to mount corresponding storage volumes.



Context

This example shows how to create dependency-based applications by customizing a template in a orchestration template.

The main components are:

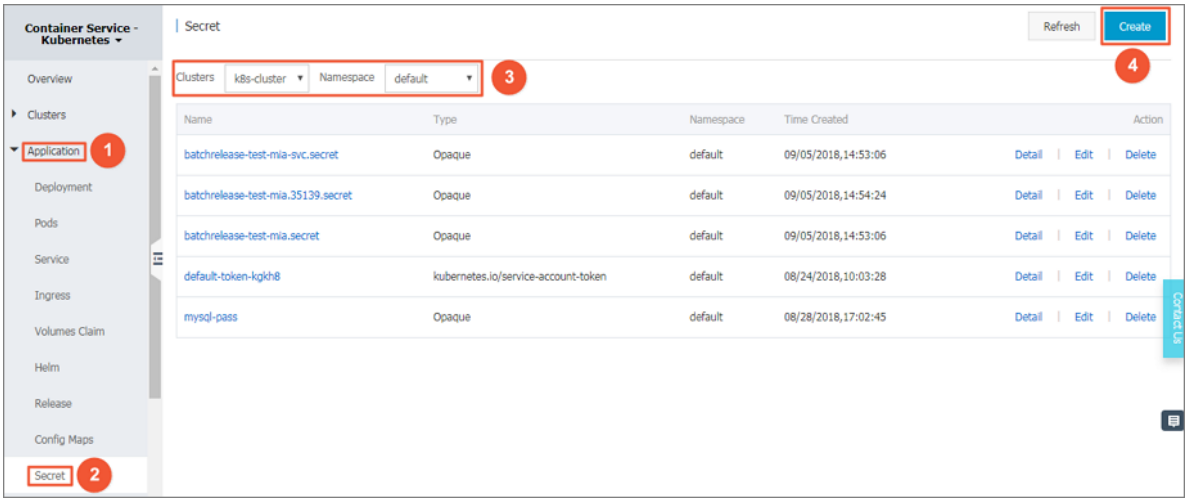
- wordpress
- mysql

Resources involved:

- Storage volume
- Secret
- Service

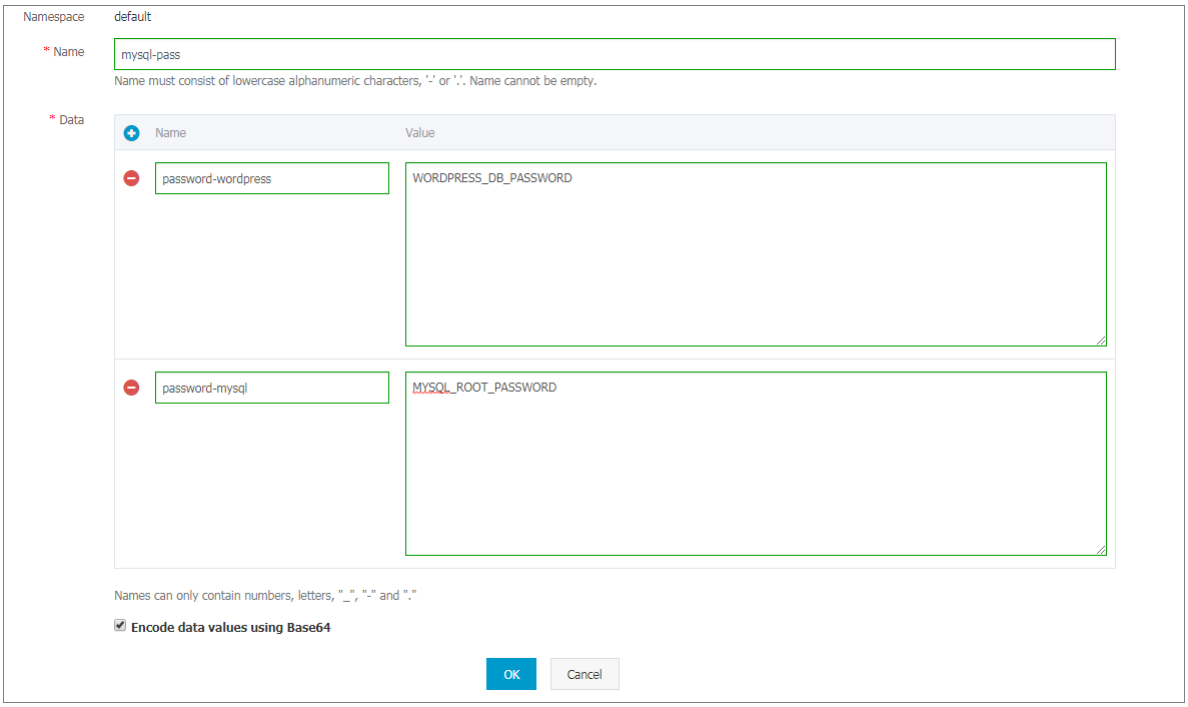
Procedure

1. Log on to the *Container Service console*.
2. Use the prepared storage volume claims. Create two storage volume claims: wordpress-pv-claim and wordpress-mysql-pv-claim which are used in the wordpress yaml file and the wordpress-mysql yaml file respectively, to mount corresponding storage volumes.
3. Click **Application** > **Secret** in the left-side navigation pane, select a cluster and namespace, and click **Create** in the upper-right corner. For the creation process, see *Create a secret*.

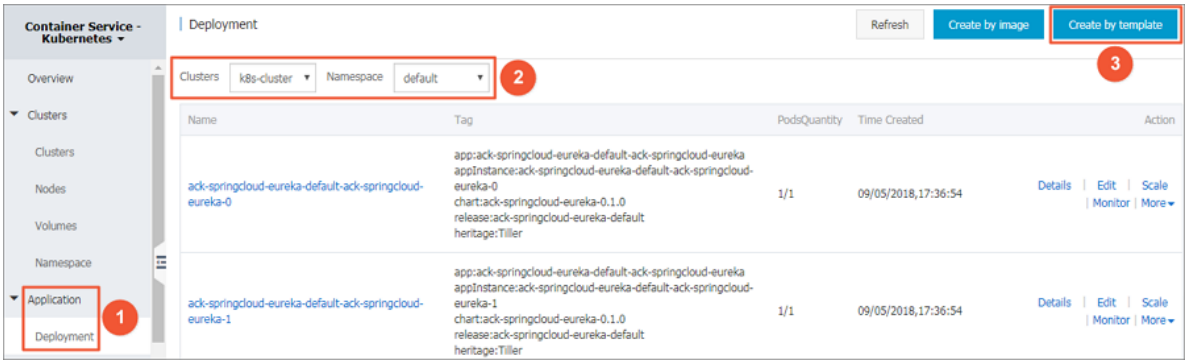


Since a user name and password is required to create and access the MySQL database, create a secret to manage the user name and password.

Before using a secret, create a secret that needs to be encrypted. In this example, the MySQL root password is created as the secret and the secret name is mysql-pass. This secret is used in the WordPress yaml file and wordpress-mysql yaml file.



4. Click **Application** > **Deployment** in the left-side navigation pane, and click **Create by template** in the upper-right corner.



Select a cluster and namespace. The yaml file for creating WordPress deployment is as follows :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
```

```
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql  #Use the name to point to the
mysql to be accessed. The name corresponds to the mysql service name
.
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-wordpress
        ports:
        - containerPort: 80
          name: wordpress
        volumeMounts:
        - name: wordpress-pvc
          mountPath: /var/www/html
      volumes:
      - name: wordpress-pvc
        persistentVolumeClaim:
          claimName: wordpress-pv-claim
```

The yaml file for creating mysql deployment is as follows:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-mysql
        ports:
        - containerPort: 3306
          name: mysql
```

```
        volumeMounts:
        - name: wordpress-mysql-pvc
          mountPath: /var/lib/mysql
      volumes:
      - name: wordpress-mysql-pvc
        persistentVolumeClaim:
          claimName: wordpress-mysql-pv-claim
```

5. To enable external access for the WordPress, you need to create the access method exposed by the WordPress service. In this example, create the WordPress service of the LoadBalancer type so that Container Service automatically creates Alibaba Cloud Server Load Balancer to provide external access.

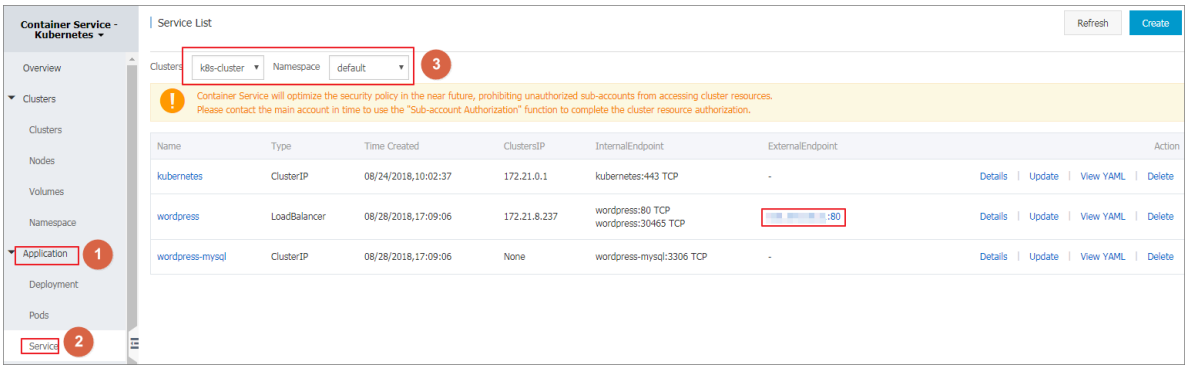
   Create a service named WordPress-mysql for the WordPress mysql so that the WordPress deploymet created on the WordPress mysql can be accessed. As the mysql is called only internally for the WordPress, you do not need to create a LoadBalancer type of service for it.

   For how to create a service, see *Create a service*.

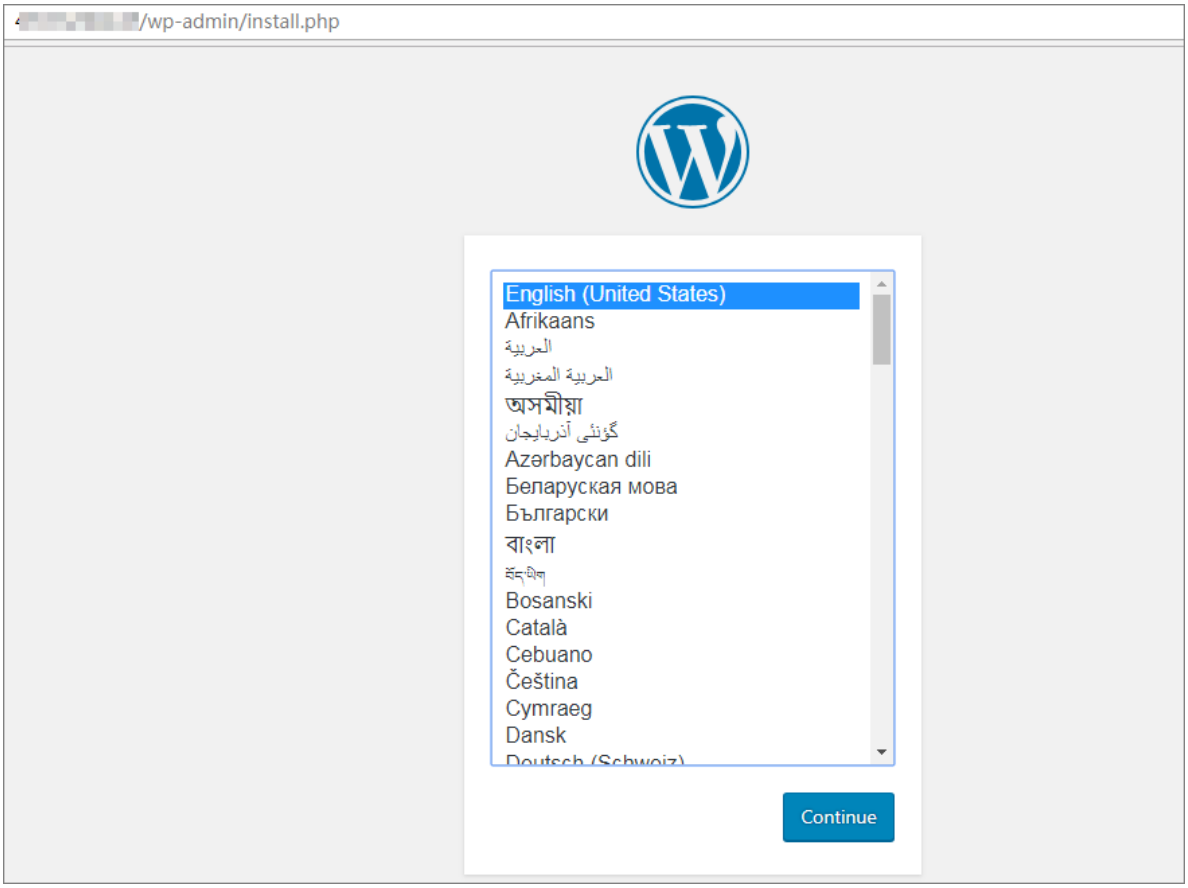
   The yaml file used to create WordPress and mysql service is as follows:

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    -port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
     clusterIP: None
```

6. When the deployment is completed, click **Application** > **Service** in the left-side navigation pane. Locate the WordPress service and view its external endpoint.



7. Access the external endpoint of the WordPress service in a browser and you can access the WordPress application through the IP address provided by Server Load Balancer.



What's next

During the configuration of the WordPress application, you can log on to the application by using the password configured in the secret. In addition, the data generated by the container to which the WordPress application belongs is saved in the data storage volume.

2 Advanced operations

2.1 Use Helm to deploy a microservice application

This document describes how to deploy a complex application to Alibaba Cloud Kubernetes Container Service. You can use different methods to deploy a SpringCloud application based on different combinations of infrastructure deployment and application deployment.

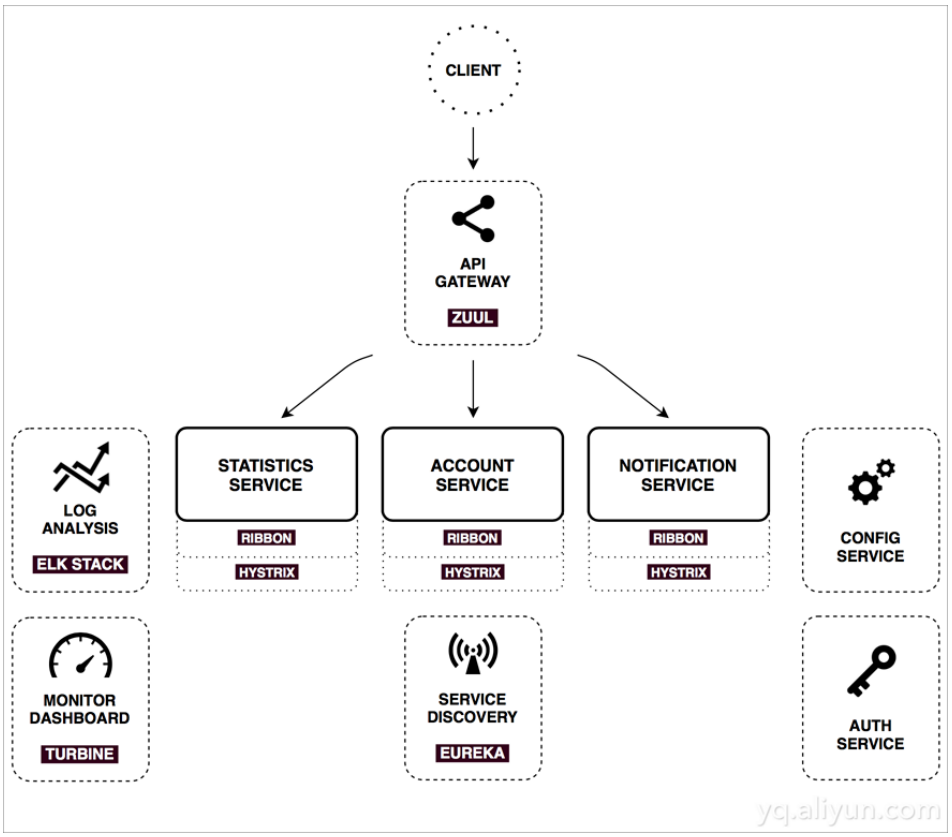
Deployment methods

1. Deploy infrastructures such as Eureka and ConfigServer together with the application.
2. Deploy the application after building infrastructures on Container Service

Sample application PiggyMetrics

PiggyMetrics is a SpringCloud application project on GitHub with more than 3400 Stars. The project main body is deployed by using DockerCompose and contains complete source codes and well-built container images. It is a very good SpringCloud containerization example.

This project contains three business microservices: statistical service, account service, and notification service. Each service corresponds to a separate MongoDB. The microservice architecture diagram(using the author's original diagram) is as follows:



SpringCloud basic components include the registry service (Eureka service registration), config service (configuration management), gateway (the API gateway, also the JavaScript Web Interface), monitor service (Hystrix Dashboard/Turbine) and more.

The deployment description file used in this article is on GitHub. If you are interested in the files, see the link: *https://github.com/binblee/PiggyMetrics/tree/master/charts*.

Scenario 1 Deploy all services with one-click deployment of helm

PiggyMetrics is deployed to a standalone device in the docker-compose YAML. To deploy PiggyMetrics to the Kubernetes environment, convert the docker-compose YAML to Kubernetes deployment YAML. The Yunqi Community has tool named *kompose* that can convert the compose file to the Kubernetes deployment file in one click.

Note:

The *docker compose* template in PiggyMetrics is in version 2.1 that is not supported by kompose. Therefore, change the version of the docker compose file to version 2.

Additionally, remove the syntax that kompose does not support:

```
depends_on:
      config:
```

```
        condition: service_healthy  #condition is not supported
```

Add Kubernetes server type annotation：

```
labels:
      kompose.service.type: loadbalancer
```

For the changed compose file, see *https://github.com/binblee/PiggyMetrics/blob/master/charts/docker-compose.yml*.

Set the environmental variables required for PiggyMetrics deployment before executing kompose.

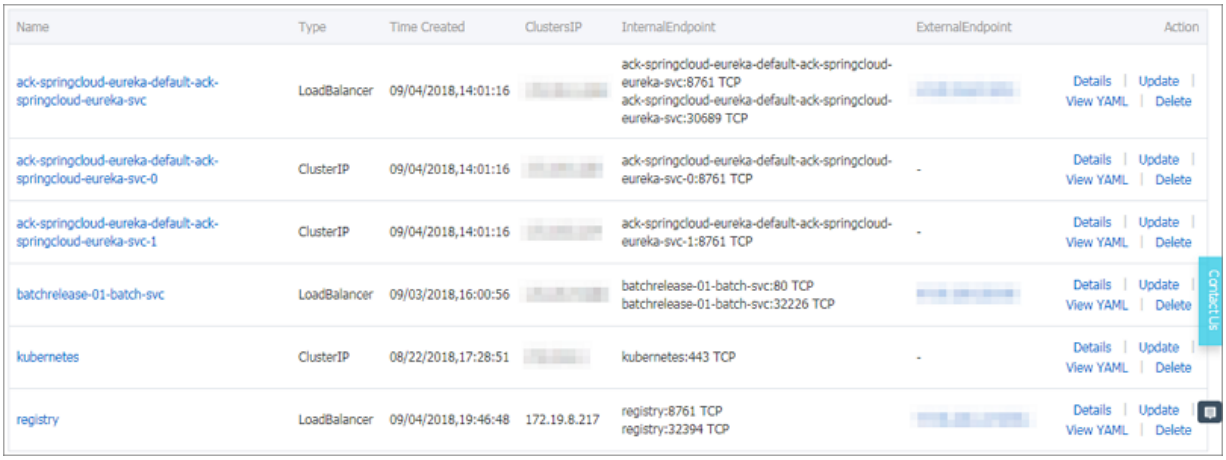
```
$ export NOTIFICATION_SERVICE_PASSWORD=passw0rd
$ export CONFIG_SERVICE_PASSWORD=passw0rd
$ export STATISTICS_SERVICE_PASSWORD=passw0rd
$ export ACCOUNT_SERVICE_PASSWORD=passw0rd
$ export MONGODB_PASSWORD=passw0rd
$ kompose convert -f docker-compose.yml -o piggymetrics -c
```

The **-c** option of kompose can generate to the *helm chart* format directory result. Use *helm* command line to deploy all services.

```
charts $ helm install -n piggymetrics piggymetrics/
```

You can see that the success message is output after deployment.

Try this configuration on your own Minikube or Alibaba Cloud Container Service for Kubernetes: *https://www.aliyun.com/product/kubernetes*. After the deployment is completed, go to the service list page, you can see all services, and the access addresses and port numbers exposed by the corresponding LoadBalancer services.

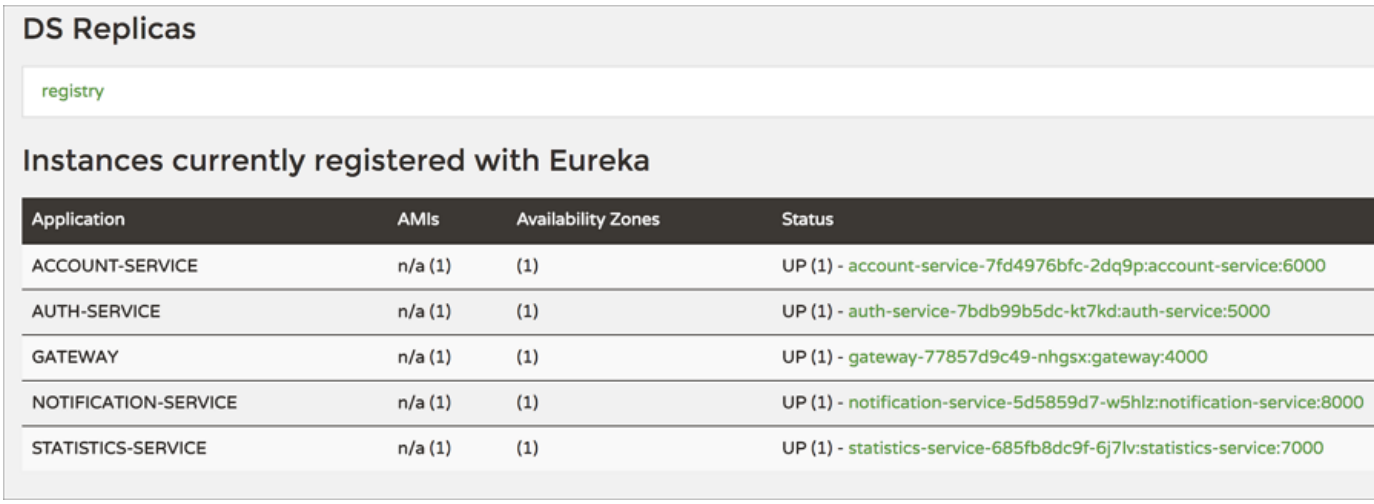


Name	Type	Time Created	ClustersIP	InternalEndpoint	ExternalEndpoint	Action
ack-springcloud-eureka-default-ack-springcloud-eureka-svc	LoadBalancer	09/04/2018,14:01:16		ack-springcloud-eureka-default-ack-springcloud-eureka-svc:8761 TCP ack-springcloud-eureka-default-ack-springcloud-eureka-svc:30689 TCP		Details | Update | View YAML | Delete
ack-springcloud-eureka-default-ack-springcloud-eureka-svc-0	ClusterIP	09/04/2018,14:01:16		ack-springcloud-eureka-default-ack-springcloud-eureka-svc-0:8761 TCP	-	Details | Update | View YAML | Delete
ack-springcloud-eureka-default-ack-springcloud-eureka-svc-1	ClusterIP	09/04/2018,14:01:16		ack-springcloud-eureka-default-ack-springcloud-eureka-svc-1:8761 TCP	-	Details | Update | View YAML | Delete
batchrelease-01-batch-svc	LoadBalancer	09/03/2018,16:00:56		batchrelease-01-batch-svc:80 TCP batchrelease-01-batch-svc:32226 TCP		Details | Update | View YAML | Delete
kubernetes	ClusterIP	08/22/2018,17:28:51		kubernetes:443 TCP	-	Details | Update | View YAML | Delete
registry	LoadBalancer	09/04/2018,19:46:48	172.19.8.217	registry:8761 TCP registry:32394 TCP		Details | Update | View YAML | Delete

You can access the PiggyMetrics interface by clicking registry service.

Piggymetrics is a personal financial service that allows you to express beautiful reports after entering your income and expenditure.

Visit the registry service to see all the services registered to the Eureka server.



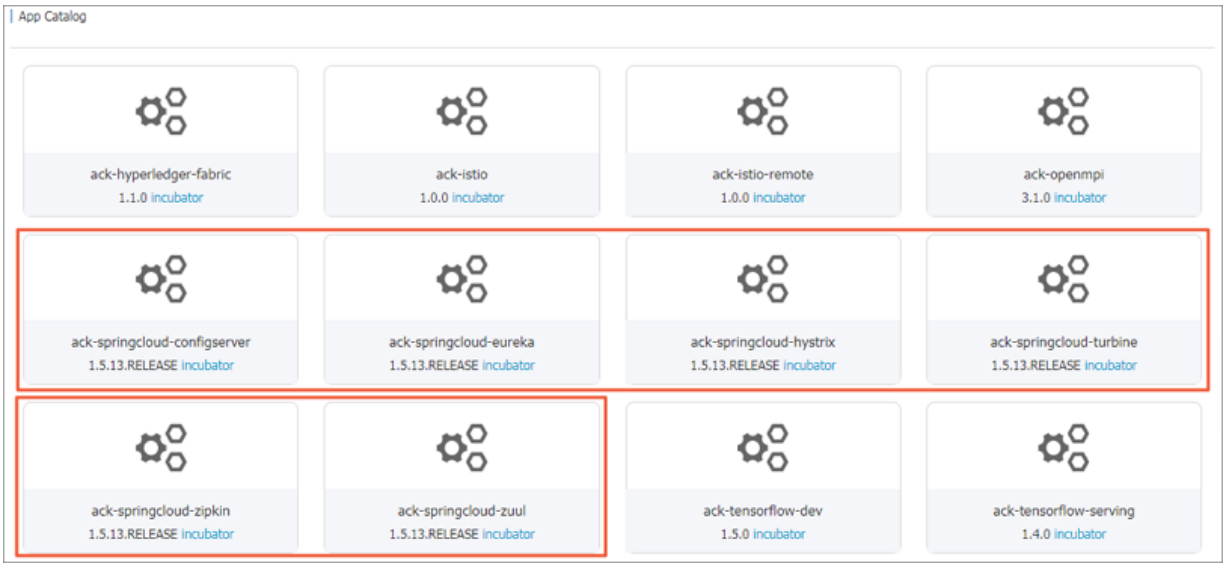
Remove PiggyMetrics to prepare for the next experiment:

```
charts $ helm delete --purge piggymetrics
release "piggymetrics" deleted
```

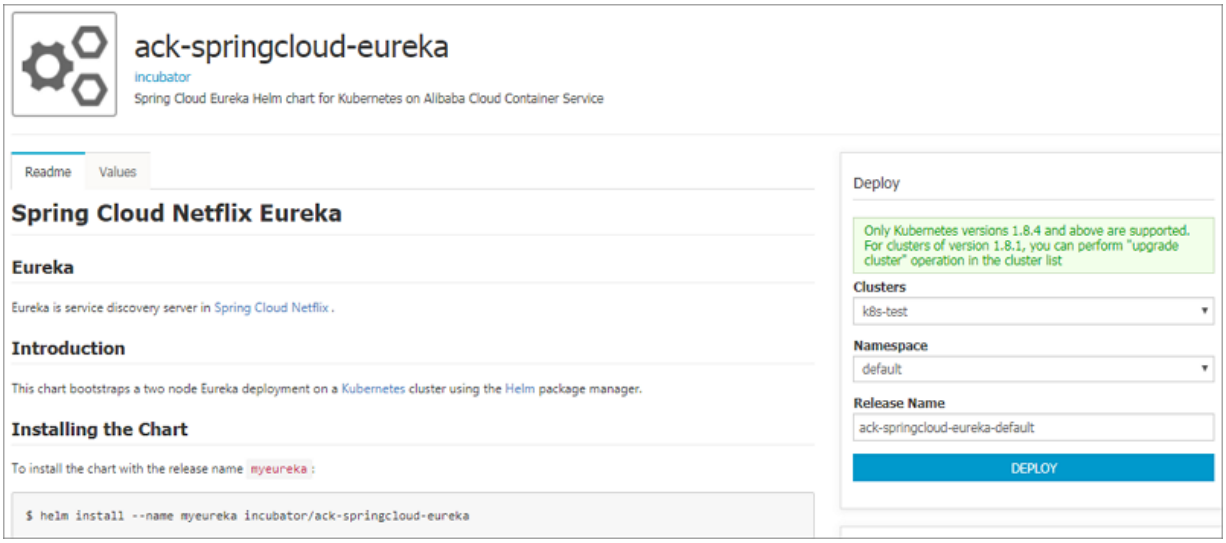
Scenario 2 Deploy the application to an existing SpringCloud basic component environment

The preceding Scenario 1 shows how to deploy all basic components (Eureka, Zuul, ConfigServer, and Hystrix Dashboard) and service applications (gateway, notification, and statistics) with one helm chart. In practice, the more common situation is that basic components such as Eureka already exist in the cluster. You only need to deploy, upgrade, and maintain your business applications.

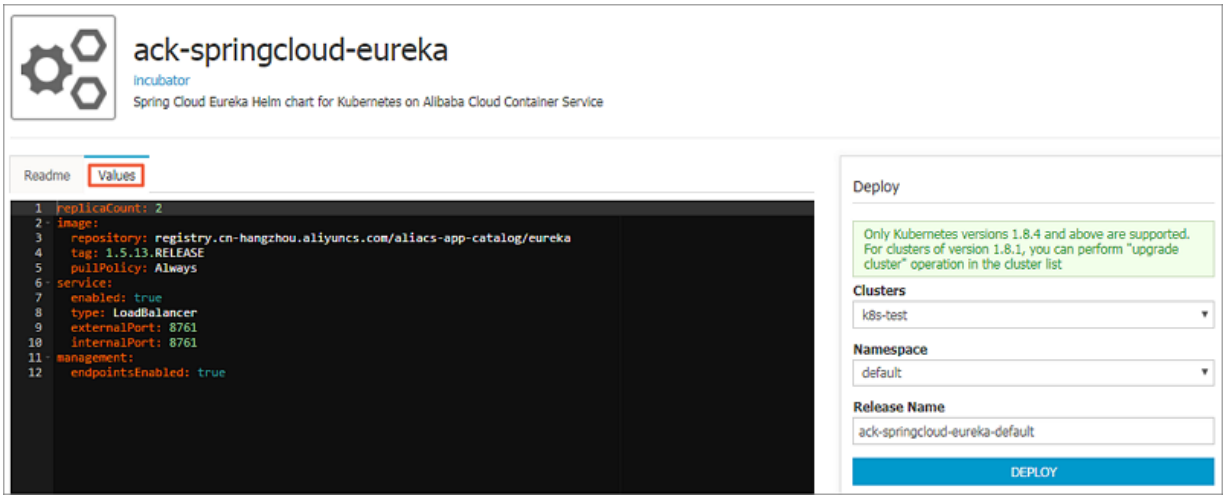
In Alibaba Cloud Container Service for Kubernetes, the App Catalog contains SpringCloud basic components.



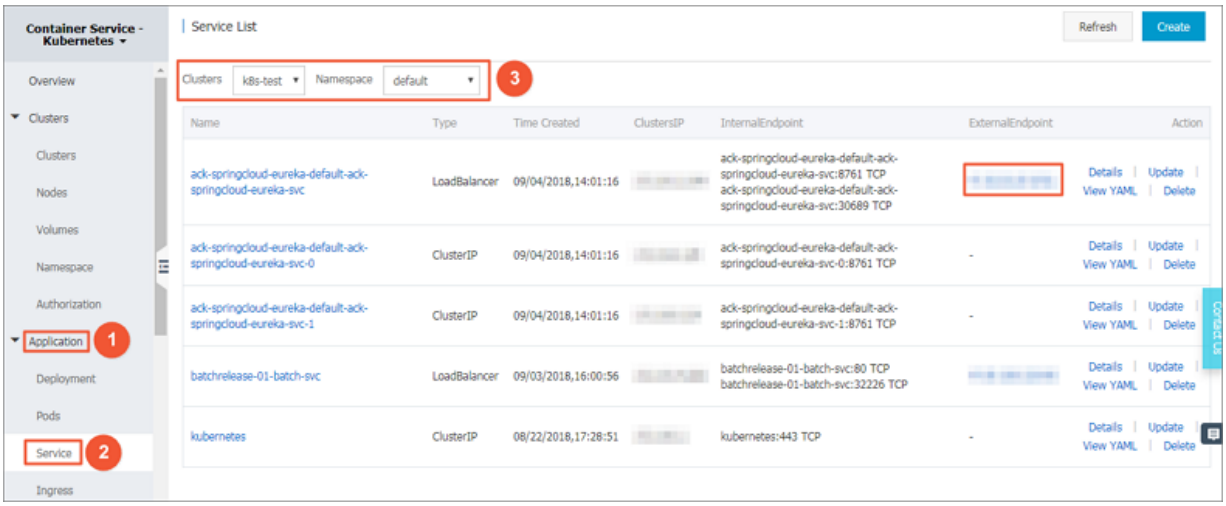
Deploy Eureka services on the App Catalog. Click the ack-springcloud-eureka component:



To view or change the configuration, click the Values tab:



Deploy directly without changing any parameters. After deployment, enter the Service List page, you can see that EurekaServer has two examples. The exposed service address is `ack-springcloud-eureka-default-ack-springcloud-eureka-svc`.



In PiggyMetrics, the EUREKA service that all containers automatically access on startup is called `registry`. In general, the EUREKA service name can be passed as a parameter in the image. In this experiment, do not change any codes or images. Therefore, take another measure, namely, expose Eureka to another service called **registry**.

Use Container Service to deploy the following yaml files.

Note:

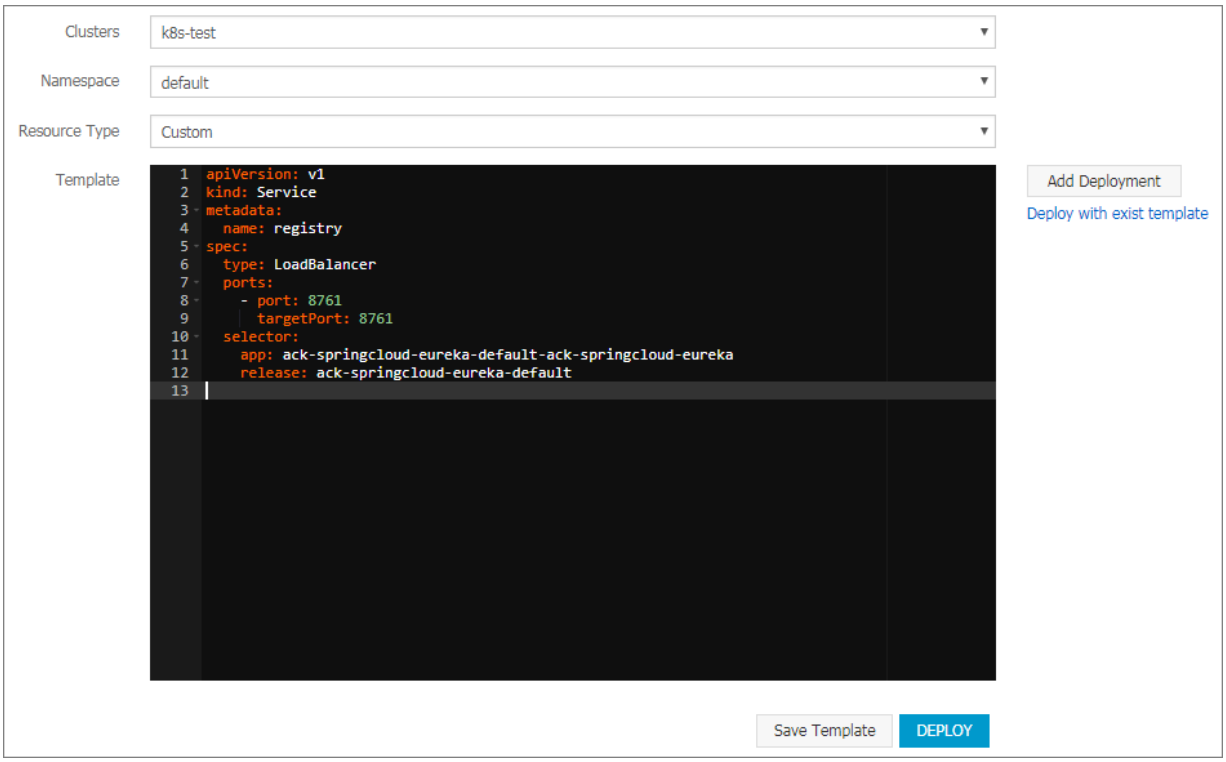
If you change the release name during App Catalog deployment, make the same changes to the followings.

```
apiVersion: v1
kind: Service
metadata:
  name: registry
spec:
  type: LoadBalancer
  ports:
    - port: 8761
      targetPort: 8761
  selector:
    app: ack-springcloud-eureka-default-ack-springcloud-eureka
    release: ack-springcloud-eureka-default
```

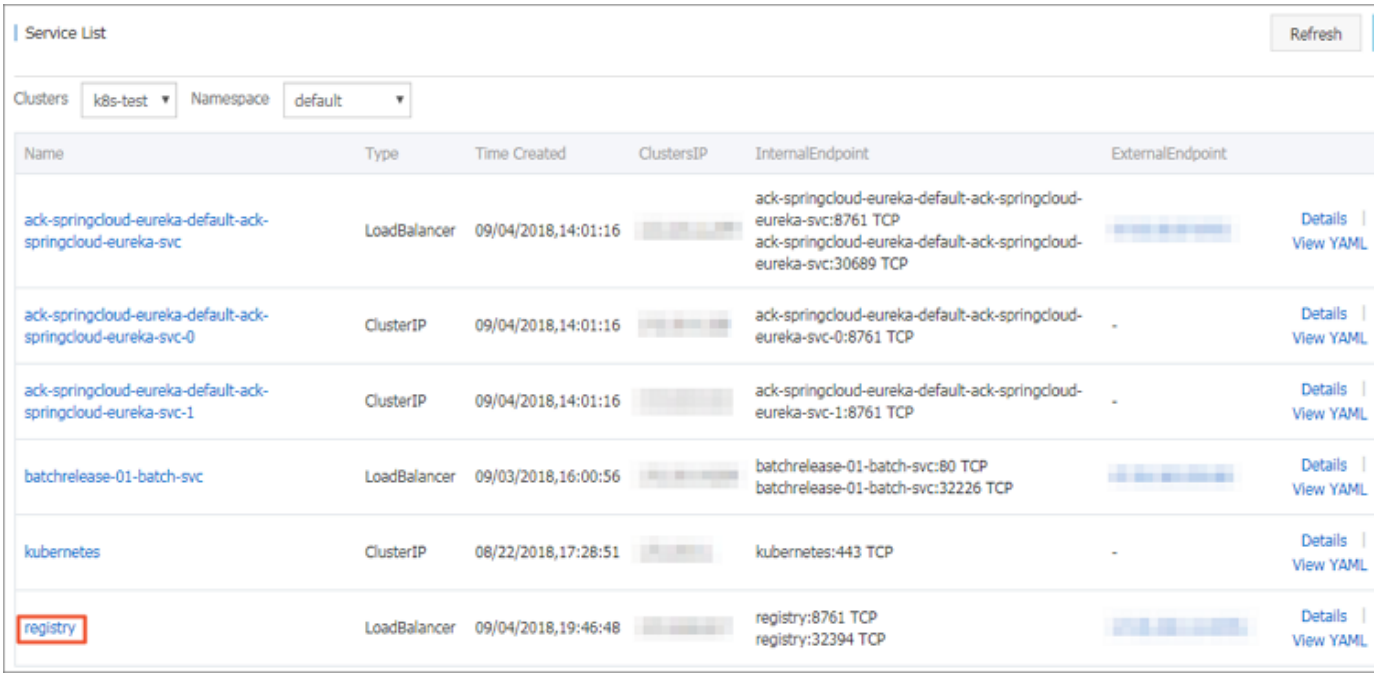
You can use the kubectl command line to create the service:

```
$ kubectl apply -f  registry-svc.yml
```

You can also do this through the console interface:



After deployment, enter the Service List page again, you can see the registry is created:



Copy the helm chart directory of PiggyMetrics to a new directory, piggymetrics-no-eureka. Delete the following two files:

```
templates/registry-deployment.yaml
```

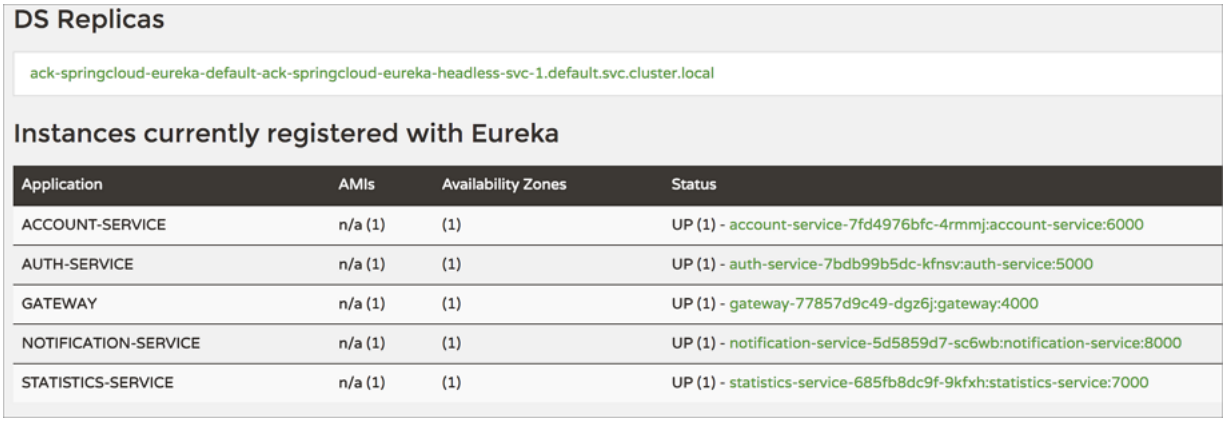
```
templates/registry-service.yaml
```

These two files are the yaml files used to deploy Eureka deployment and svc, respectively. As you have used the App Catalog to successfully deploy a new registry service as the basic SpringCloud component, you do not have to repeat the deployment.

Execute the helm command to deploy PiggyMetrics again.

```
$ helm install -n piggymetrics piggymetrics-no-eureka/
```

After all services start, access the registry service and you can see that all PiggyMetrics services are properly registered with EurekaServer.



DS Replicas

ack-springcloud-eureka-default-ack-springcloud-eureka-headless-svc-1.default.svc.cluster.local

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ACCOUNT-SERVICE	n/a (1)	(1)	UP (1) - account-service-7fd4976bfc-4rmmj:account-service:6000
AUTH-SERVICE	n/a (1)	(1)	UP (1) - auth-service-7bdb99b5dc-kfnsv:auth-service:5000
GATEWAY	n/a (1)	(1)	UP (1) - gateway-77857d9c49-dgz6j:gateway:4000
NOTIFICATION-SERVICE	n/a (1)	(1)	UP (1) - notification-service-5d5859d7-sc6wb:notification-service:8000
STATISTICS-SERVICE	n/a (1)	(1)	UP (1) - statistics-service-685fb8dc9f-9kfxh:statistics-service:7000

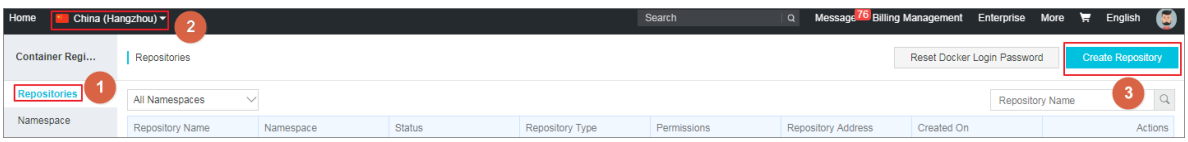
The PiggyMetrics application has been deployed to the environment with EurekaServer. Access `GATEWAY` to see the familiar login interface.

2.2 Use a private image repository to create an application

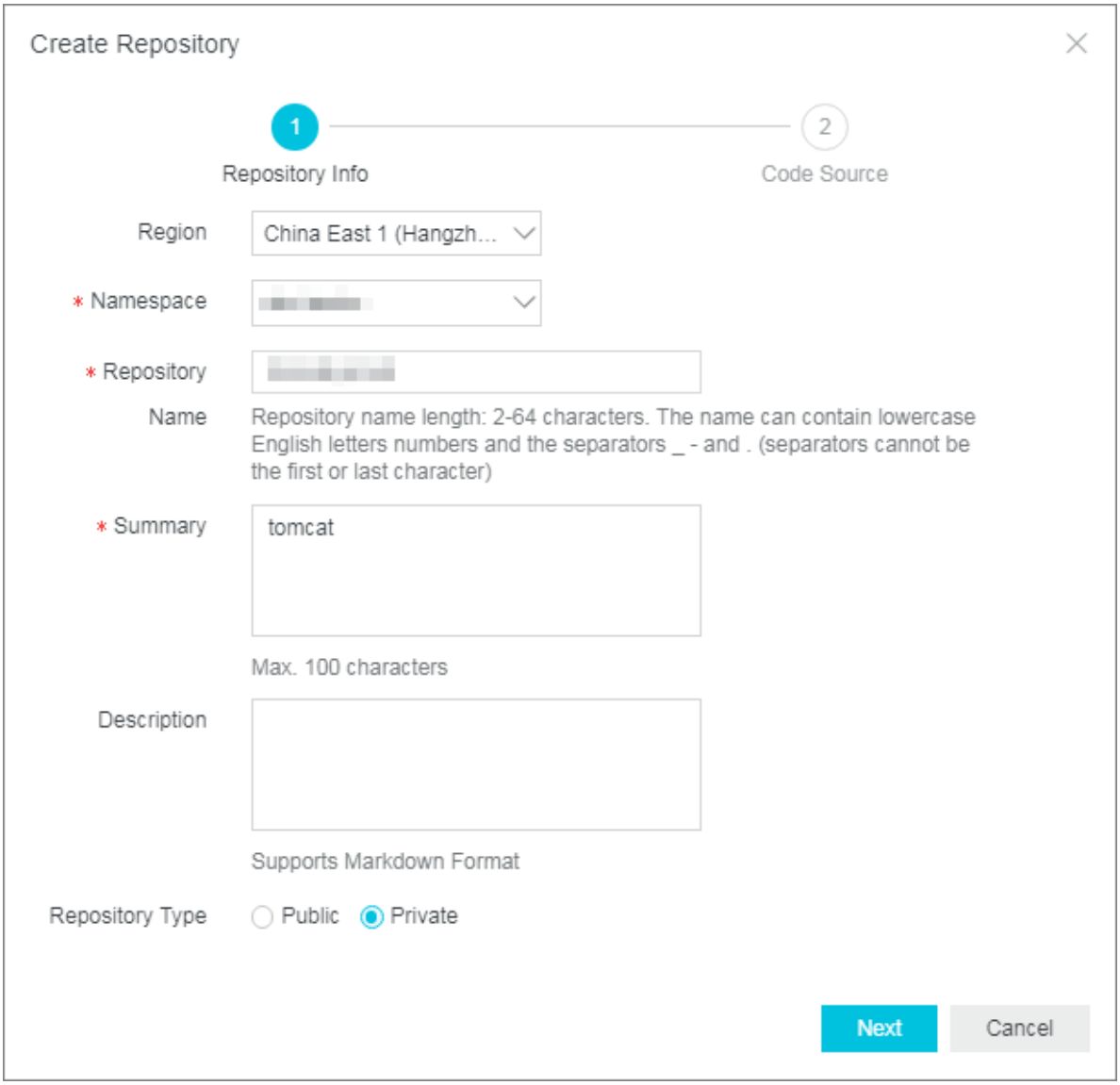
In many scenarios, an image in a private image repository is used for deploying an application. In this document, use Alibaba Cloud image repository service to create a private image repository, and create an application that uses this private image repository.

Step 1 Create a private image repository

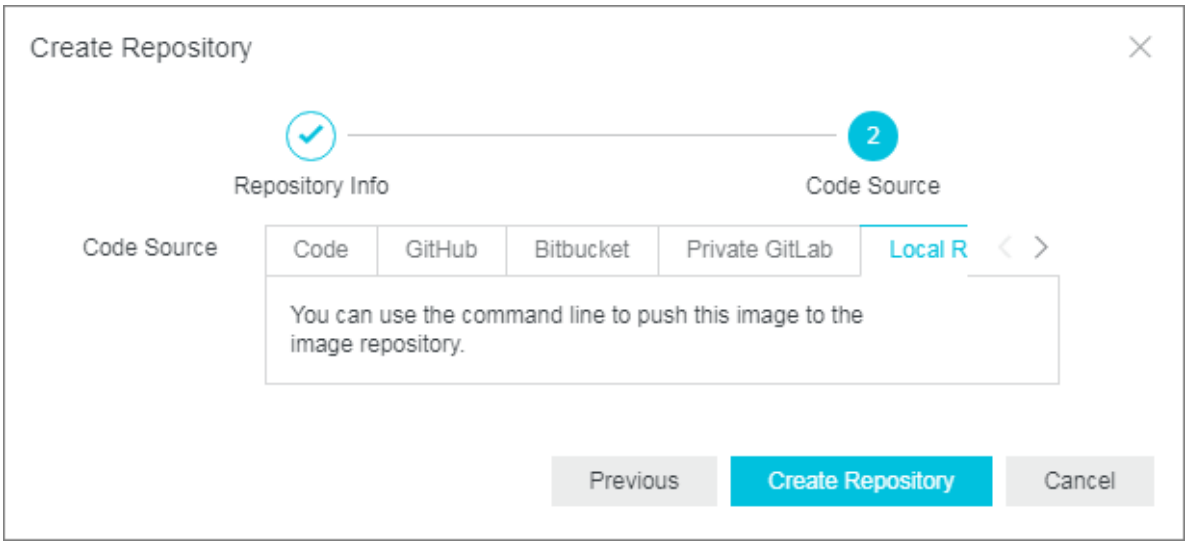
1. Log on to the *Container Registry console*.
2. Click **Repositories** in the left-side navigation pane, select the target region, and lick **Create Repository**.
3. Configure the image repository in the dialog box, and then click **Create Repository**. In this example, select the private image repository type and set the code source as a local repository.



4. On the repositories page, select the target region, and you can see that the created image repository. Click **Manage**on the right.



5. On the repository management page, click **Details**, and you can follow the guide to use the private image repository.

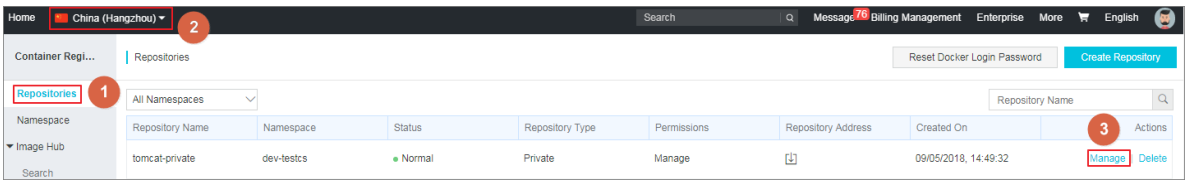


6. Log on to the image repository in the Linux environment and upload the local image to the private image repository.

```
$ sudo docker login --username=abc@aliyun.com
 Password
             ## Image repository independent login password:
 Login Succe ed

$ dockeagesr im
            #This example is tomcat ages
REPOSITORY
        TAG                 IMAGE ID            CREATED
SIZE
tomcat
        latest              2d43521f2b1a        6 days ago
463MB

 $ sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/
kubernetes-java/tomcat-private:[Image version number]            #V1
 in this example
 $ sudo docker push registry.cn-hangzhou.aliyuncs.com/kubernetes-
java/tomcat-private:[Inage version number]                       #V1
 in this example

 The push refers to a repository [registry.cn-hangzhou.aliyuncs.com/
kubernetes-java/tomcat-private]
9072c7b03a1b: Pushed
f9701cf47c58: Pushed
365c8156ff79: Pushed
2de08d97c2ed: Pushed
6b09c39b2b33: Pushed
4172ffa172a6: Pushed
1dccf0da88f3: Pushed
d2070b14033b: Pushed
63dcf81c7ca7: Pushed
ce6466f43b11: Pushed
719d45669b35: Pushed
3b10514a95be: Pushed
```

```
V1: digest: sha256:cded14cf64697961078aedfdf870e704a5227018
8c8194b6f70c778a8289d87e size: 2836
```

7. Return to the image repository detail page, and click **Image version** in the left navigation pane, you can see that the image has been uploaded successfully, and you can view the image version information.



Step 2 Create a docker-registry secret

When using Kubernetes to create an application by pulling a private image, pass the identity authentication information of the private image repository to Kubernetes through a docker-registry secret.

Create a docker-registry secret as follows:

```
kubectl create secret docker-registry regsecret --docker-server=
registry-internal.cn-hangzhou.aliyuncs.com --docker-username=abc@
aliyun.com --docker-password=xxxxxx --docker-email=abc@aliyun.com
```

where:

- --regsecret: Specifies the secret key name and the name is customizable.
- --docker-server: Specifies the Docker repository address.
- --docker-username: Specifies the user name of the Docker repository.
- --docker-password: Specifies the Docker repository login password, namely, the independent login password of the container image registry.
- --docker-email: Specifies the email address.

Note:

You cannot use the secrets on the Container Service console to create secrets.

To pull an image successfully, add the secret parameter to the yml file.

```
containers:
    - name: foo
      image: registry-internal.cn-hangzhou.aliyuncs.com/abc/test:1.0
imagePullSecrets:
    - name: regsecret
```

where:

- imagePullSecrets declares that a secret key must be specified when you pull the image.
- regsecret must be the same as the preceding secret key name.
- The docker repository name in the image must be the same as that in the -- docker-server.

Step 3 Use a private image repository to create an application

The orchestration is as follows:

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: private-image
  nameSpace: default
  labels:
    app: private-image
spec:
  replicas: 1
  selector:
    matchLabels:
      app: private-image
  template:
    metadata:
      labels:
        app: private-image
    spec:
      containers:
      - name: private-image
        image: registry.cn-hangzhou.aliyuncs.com/xxx/tomcat-private:
latest
        ports:
        - containerPort: 8080
      imagePullSecrets:
      - name: regsecret
```

For more information, see the official Kubernetes documentation *Use a Private Registry*.