Alibaba Cloud
Container Service for Kubernetes

Best Practices

Issue: 20181127

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

Legal disclaimer.....I
Generic conventions.....I
1 Cluster.....1
1.1 Update expired certificates of a Kubernetes cluster.....1
2 Application.....3
3 Network.....4
3.1 Deploy high-reliability Ingress Controller.....4
4 Release.....12
4.1 Implement Layer-4 canary release by using Alibaba Cloud Server Load Balancer in a Kubernetes cluster.....12
5 Istio.....17
5.1 Implement Istio distributed tracking in Kubernetes.....17
6 DevOps.....28

1 Cluster

1.1 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 ${base64CAKey}
```

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.

2 Application

3 Network

3.1 Deploy high-reliability Ingress Controller

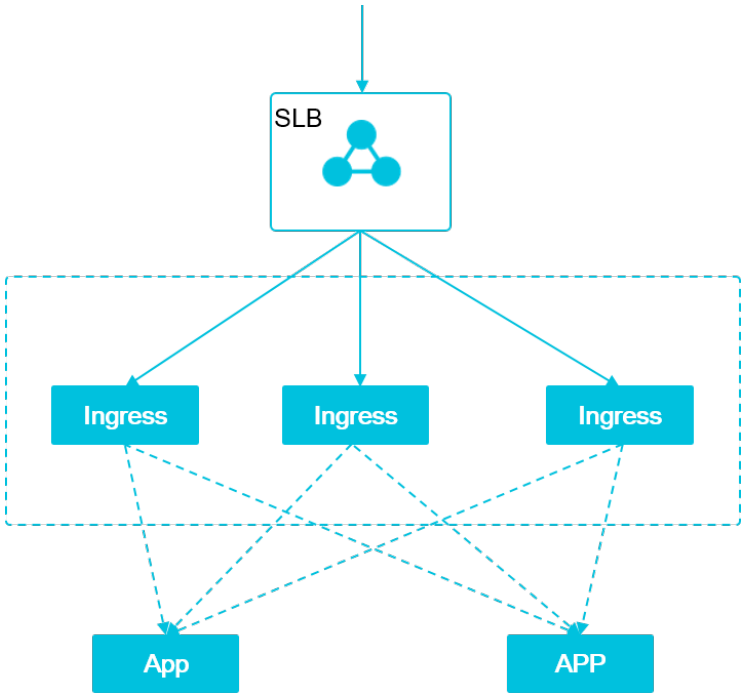
In Kubernetes clusters, Ingress is a collection of rules that authorize the inbound access to the cluster and provide you with Layer-7 Server Load Balancer capabilities. You can provide the externally accessible URL, Server Load Balancer, SSL, and name-based virtual host. As the access layer of the cluster traffic, the high reliability of Ingress is important. This document introduces how to deploy a set of high-reliability Ingress access layer with good performance.

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*.

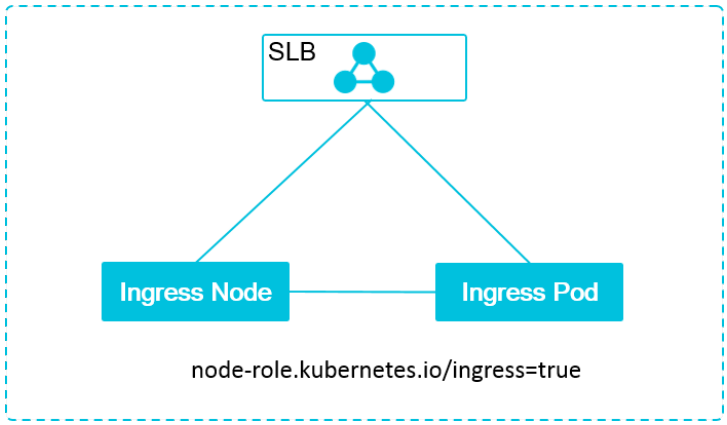
High-reliability deployment architecture

To implement high reliability, the single point of failure must be solved first. Generally, the single point of failure is solved by deployment with multiple copies. Similarly, use the multi-node deployment architecture to deploy the high-reliability Ingress access layer in Kubernetes clusters. As Ingress is the access point of the cluster traffic, we recommend that you have the Ingress node exclusive to you to avoid the business applications and Ingress services from competing for resources.



As mentioned in the preceding deployment architecture figure, multiple exclusive Ingress instances form a unified access layer to carry the traffic at the cluster entrance and expand or contract the Ingress nodes based on the backend business traffic. If your cluster scale is not large in the early stage, you can also deploy the Ingress services and business applications in the hybrid mode, but we recommend that you limit and isolate the resources.

Instructions on deploying high-reliability Ingress access layer



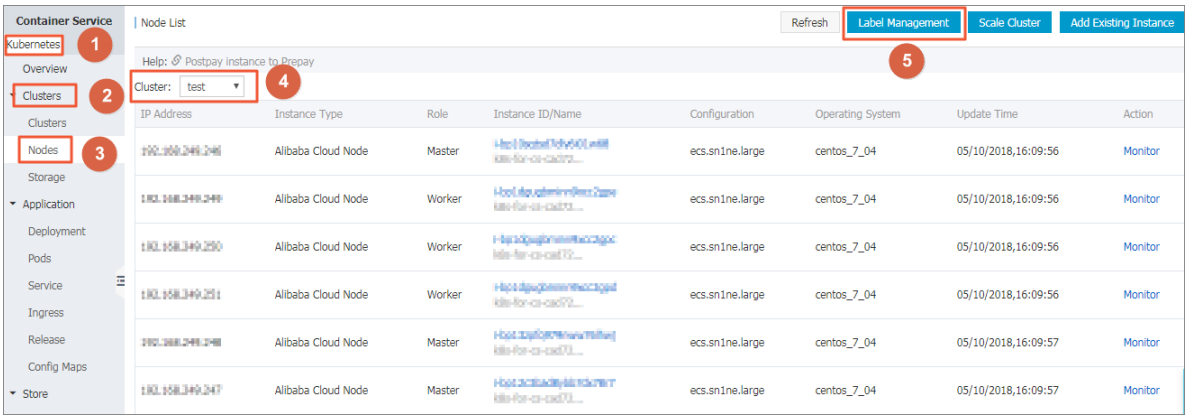
- Ingress Server Load Balancer: The frontend Server Load Balancer instance of the Ingress access layer.
- Ingress node: The cluster node in which the Ingress pod is deployed.
- Ingress pod: The Ingress service.

The Ingress Server Load Balancer, Ingress node, and Ingress pod are associated based on the tag `node-role.kubernetes.io/ingress=true`:

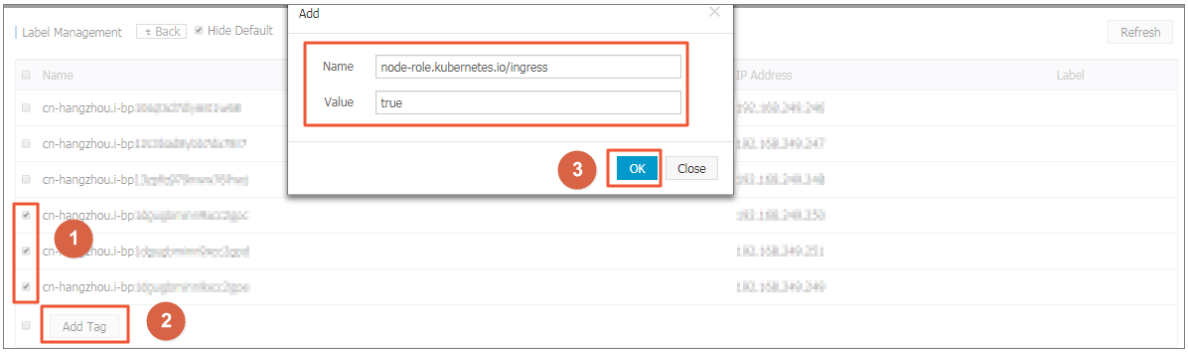
1. The Ingress Server Load Balancer backend only mounts the cluster nodes with the tag `node-role.kubernetes.io/ingress=true`.
2. The Ingress pod is only deployed to the cluster nodes with the tag `node-role.kubernetes.io/ingress=true`.

Step 1 Add a label for Ingress nodes

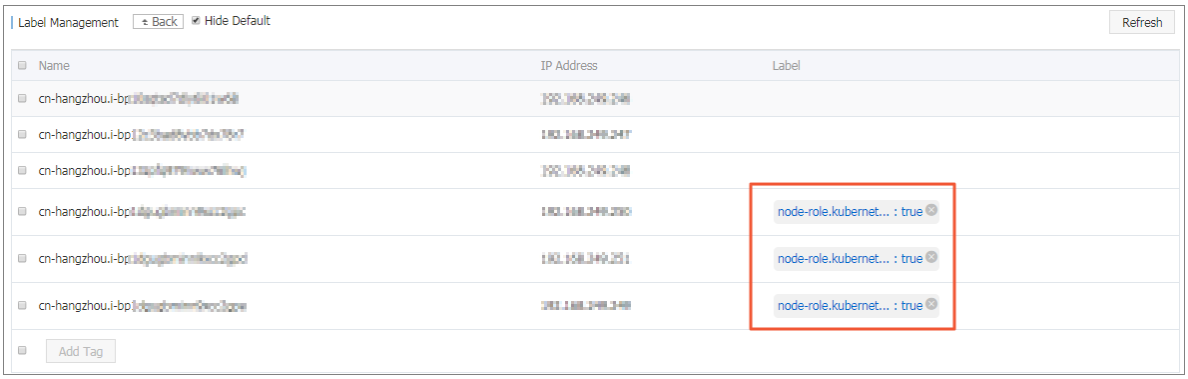
1. Log on to the *Container Service console*.
2. Under Kubernetes, click **Clusters** > **Nodes** in the left-side navigation pane.
3. Select the cluster from the Cluster drop-down list. View the instance IDs of the worker nodes and then click **Label Management** in the upper-right corner.



4. The Label Management page appears. Select the worker nodes and then click **Add Tag**. Add the label `node-role.kubernetes.io/ingress : true` to the worker nodes and then click **OK**.



On the Label Management page, you can see the label is added to the worker nodes.



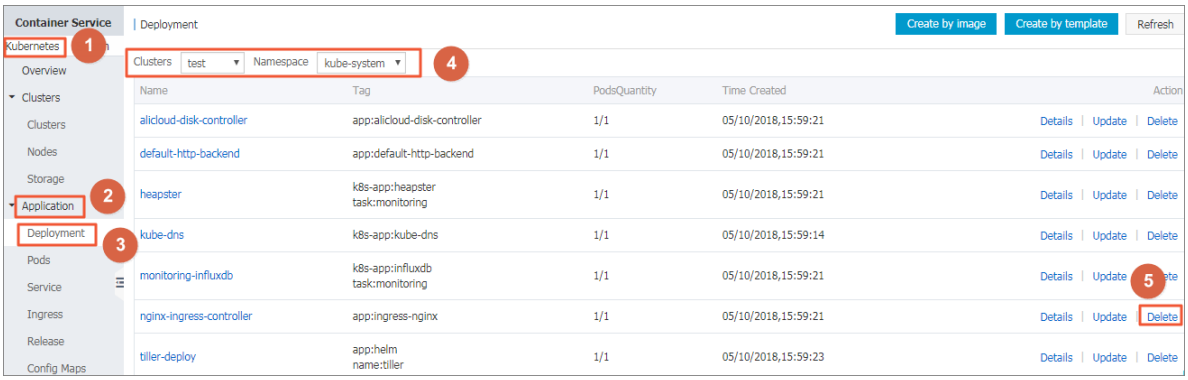
You can also log on to the master node and run the command `kubectl label no nodeID node-role.kubernetes.io/ingress=true` to add the label to the worker nodes quickly.

Step 2 Create an Ingress service

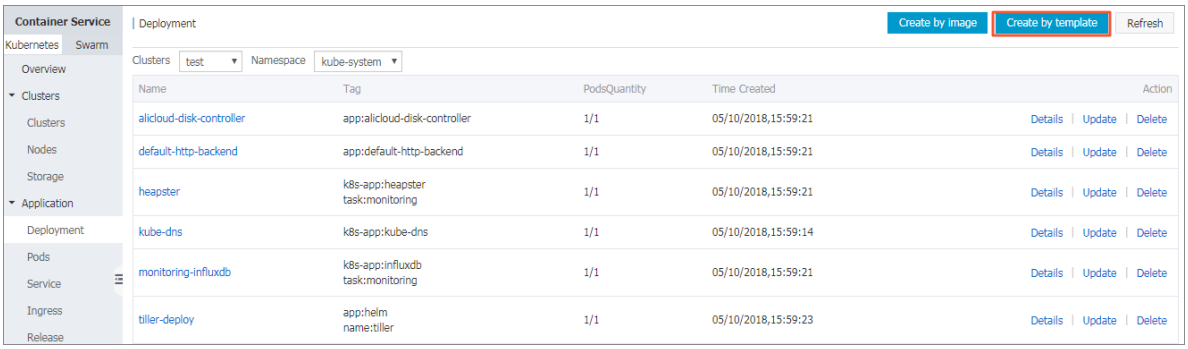
1. Log on to the *Container Service console*.
2. Under Kubernetes, click**Application** > **Deployment** in the left-side navigation pane.
3. Select the cluster from the Clusters drop-down list and kube-system from the Namespace drop-down list. Click **Delete** at the right of nginx-ingress-controller and then click OK in the displayed dialog box.

   An Ingress Controller is deployed by default when the cluster is initialized. For more information, see *ingress-nginx*. You must delete the Ingress Controller deployed by default first and then deploy a new set of high-reliability Ingress Controller access layer.

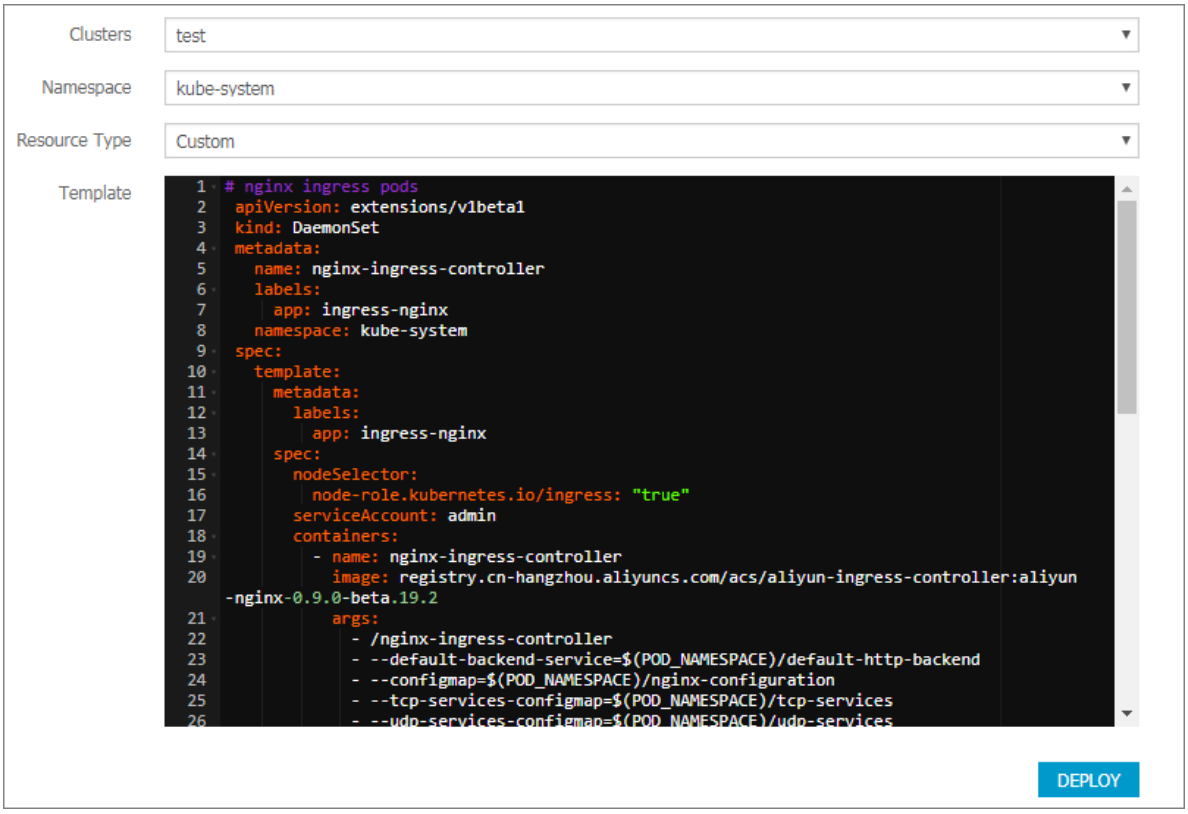
   Note:
   The Ingress Controller deployed by default is associated with the nginx-ingress-lb service. Do not delete the associated service when deleting the deployment. The nginx-ingress-lb service is about to be updated later.



4. Click **Create by template** in the upper-right corner.



5. Select the cluster from the Clusters drop-down list and kube-system from the Namespace drop-down list. Select a sample template or Custom from the Resource Type drop-down list. Click **DEPLOY**.




```
# nginx ingress pods
 apiVersion: extensions/v1beta1
 kind: DaemonSet
 metadata:
   name: nginx-ingress-controller
   labels:
     app: ingress-nginx
   namespace: kube-system
 spec:
   template:
     metadata:
       labels:
         app: ingress-nginx
     spec:
       nodeSelector:
         node-role.kubernetes.io/ingress: "true"
       serviceAccount: admin
       containers:
         - name: nginx-ingress-controller
           image: registry.cn-hangzhou.aliyuncs.com/acs/aliyun-ingress-controller:aliyun
-nginx-0.9.0-beta.19.2
           args:
             - /nginx-ingress-controller
             - --default-backend-service=$(POD_NAMESPACE)/default-http-backend
             - --configmap=$(POD_NAMESPACE)/nginx-configuration
             - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-services
             - --udp-services-configmap=$(POD_NAMESPACE)/udp-services
```

In this example, redeploy the Ingress Controller to the target Ingress node in the DaemonSet method. You can also deploy the Ingress Controller by using deployment together with the affinity.

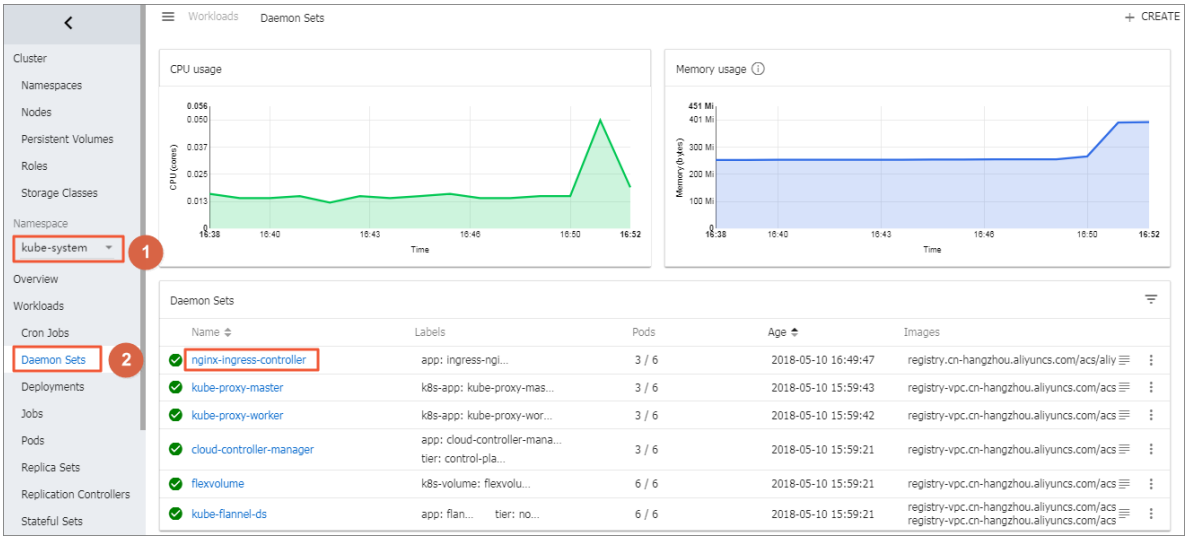
```
# nginx ingress pods
 apiVersion: extensions/v1beta1
 kind: DaemonSet
 metadata:
   name: nginx-ingress-controller
   labels:
     app: ingress-nginx
   namespace: kube-system
 spec:
```

```
   template:
     metadata:
       labels:
         app: ingress-nginx
     spec:
       nodeSelector:
         node-role.kubernetes.io/ingress: "true" ##Deploy the pod to
 the corresponding node by using the label selector.
       serviceAccount: admin
       containers:
         - name: nginx-ingress-controller
           image: registry.cn-hangzhou.aliyuncs.com/acs/aliyun-
ingress-controller:aliyun-nginx-0.9.0-beta. 19.2
           args:
             - /nginx-ingress-controller
             - --default-backend-service=$(POD_NAMESPACE)/default-
http-backend
             - --configmap=$(POD_NAMESPACE)/nginx-configuration
             - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-
services
             - --udp-services-configmap=$(POD_NAMESPACE)/udp-
services
             - --annotations-prefix=nginx.ingress.kubernetes.io
             - --publish-service=$(POD_NAMESPACE)/nginx-ingress-lb
             - --v=2
           env:
             - name: POD_NAME
               valueFrom:
                 fieldRef:
                   fieldPath: metadata.name
             - name: POD_NAMESPACE
               valueFrom:
                 fieldRef:
                   fieldPath: metadata.namespace
           ports:
           - name: http
             containerPort: 80
           - name: https
             containerPort: 443
           livenessProbe:
             failureThreshold: 3
             httpGet:
               path: /healthz
               port: 10254
               scheme: HTTP
             initialDelaySeconds: 10
             periodSeconds: 10
             successThreshold: 1
             timeoutSeconds: 1
           readinessProbe:
             failureThreshold: 3
             httpGet:
               path: /healthz
               port: 10254
               scheme: HTTP
             periodSeconds: 10
             successThreshold: 1
             timeoutSeconds: 1
```

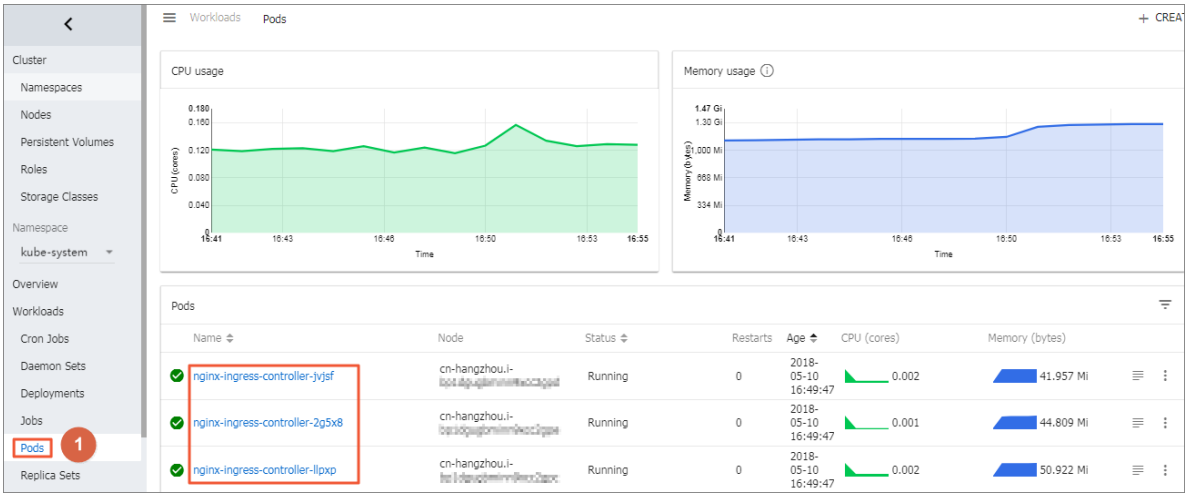
6. A message indicating the deployment status is displayed on the page after you click DEPLOY.

   After the successful deployment, click **Kubernetes Dashboard** in the message to go to

the dashboard. Select kube-system as the namespace. Click **Daemon Sets** in the left-side navigation pane and view the nginx-ingress-controller.



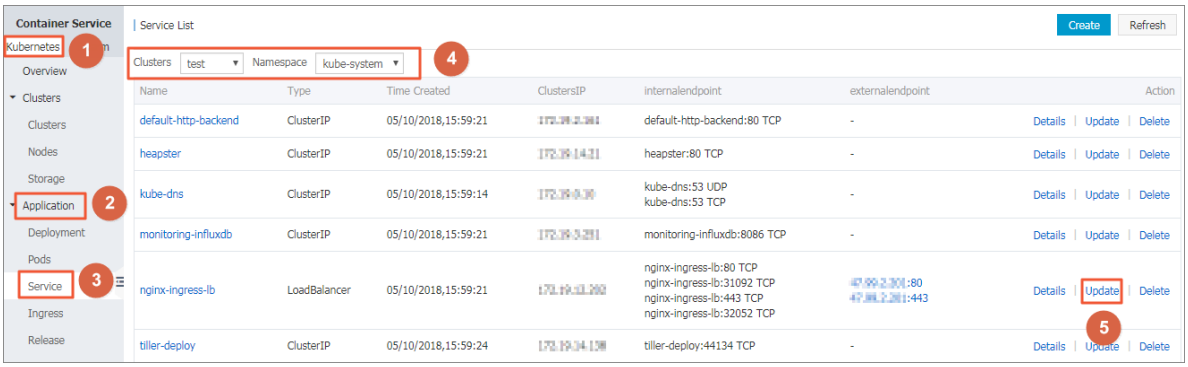
7. Click **Pods** in the left-side navigation pane to view the pods of nginx-ingress-controller.



Step 3 Update Ingress Server Load Balancer service

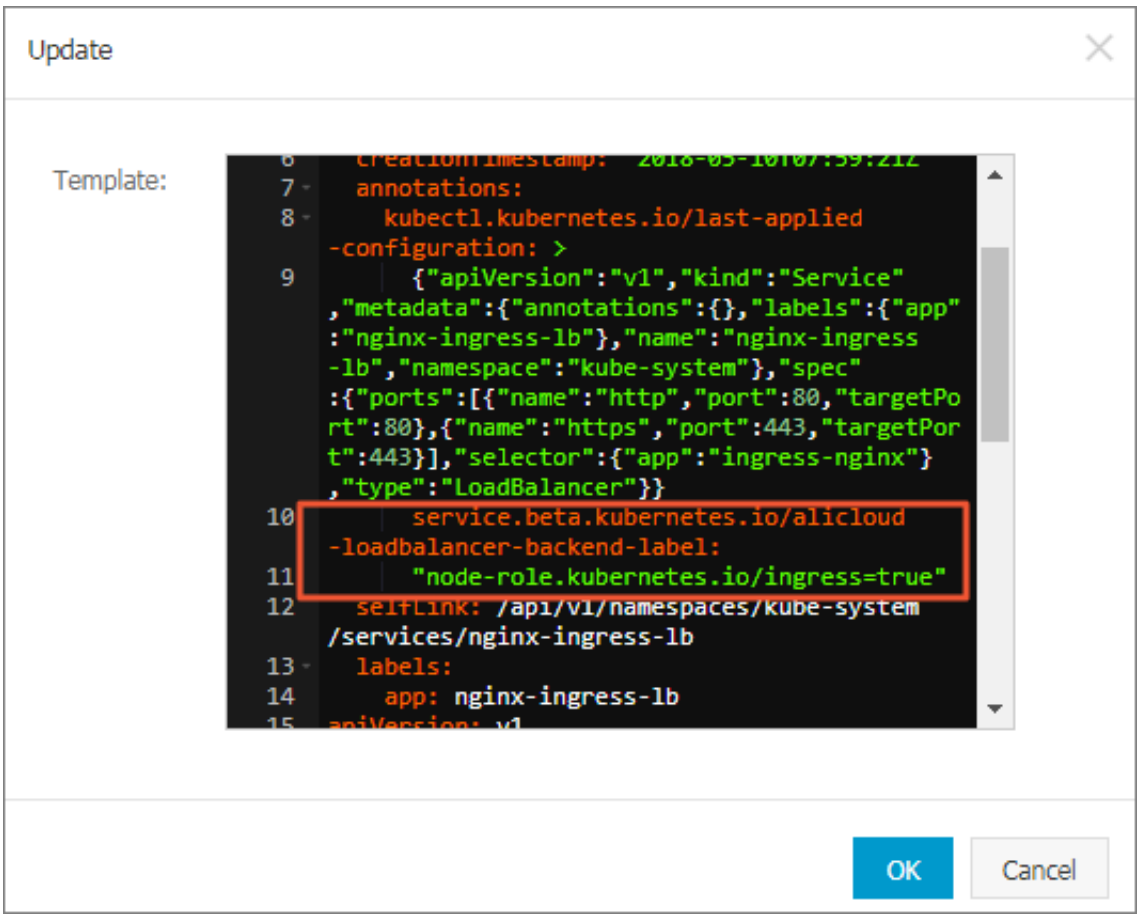
1. Log on to the *Container Service console*.
2. Under Kubernetes, click**Application** > **Service in the left-side navigation pane.** in the left-side navigation pane.
3. Select the cluster from the Clusters drop-down list and kube-system from the Namespace drop-down list. Click **Update**.

   An Ingress Server Load Balancer service is deployed by default when the cluster is initialized. For more information, see *ingress-nginx*. You must update the Ingress Server Loadbalancer service to automatically identify the ingress node that is mounted for marking.



4. In the displayed dialog box, add the annotation `service.beta.kubernetes.io/alicloud-loadbalancer-backend-label "node-role.kubernetes.io/ingress=true"`, and then click **OK**.

   You can also log on to the master node of the cluster and run the command `kubectl apply -f https://acs-k8s-ingress.oss-cn-hangzhou.aliyuncs.com/nginx-ingress-slb-service.yml` to update the nginx-ingress-lb service.



Then, you have deployed the high-reliability access layer of Ingress, which allows you to effectively deal with the challenges of single point of failure and business traffic, and quickly expand the Ingress access layer by adding tags.

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_12*.
- You have connected to the master node by using SSH. For more information, see *#unique_13*.

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:
     matchLabels:
       run: old-nginx
   template:
     metadata:
       labels:
         run: old-nginx
         app: nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
         imagePullPolicy: Always
         name: old-nginx
         ports:
```

```
          - containerPort: 80
            protocol: TCP
       restartPolicy: Always
 ---
 apiVersion: v1
 kind: Service
 metadata:
   labels:
     run: nginx
   name: nginx
 spec:
   ports:
   - port: 80
     protocol: TCP
     targetPort: 80
   selector:
     app: nginx
   sessionAffinity: None
   type: LoadBalancer ##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:
     matchLabels:
       run: new-nginx
   template:
     metadata:
       labels:
         run: new-nginx
         app: nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
         imagePullPolicy: Always
         name: new-nginx
         ports:
         - containerPort: 80
           protocol: TCP
       restartPolicy: 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.

5 Istio

5.1 Implement Istio distributed tracking in Kubernetes

Background

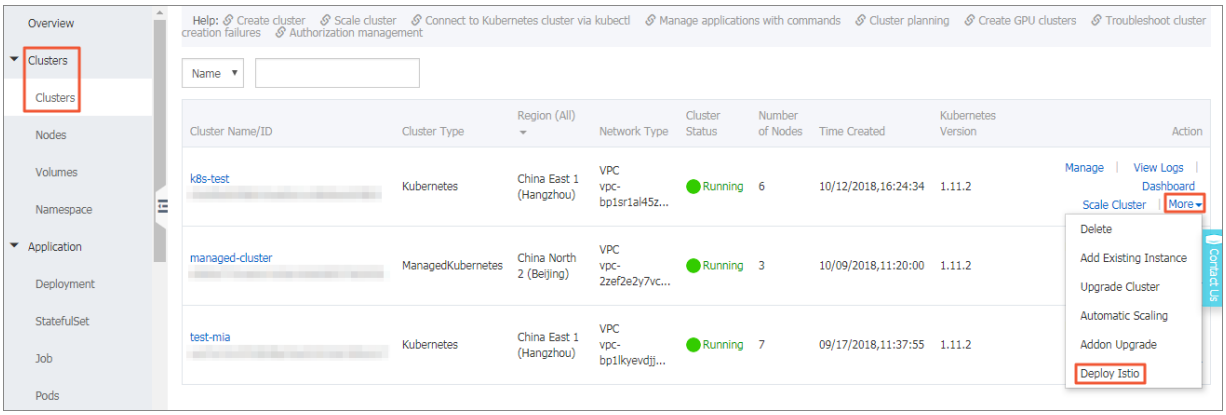
Microservice is a focus in the current era. More and more IT enterprises begin to embrace the microservices. The microservice architecture splits a complex system into several small services and each service can be developed, deployed, and scaled independently. As a heaven-made match, the microservice 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 microservi ces. 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 application 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.

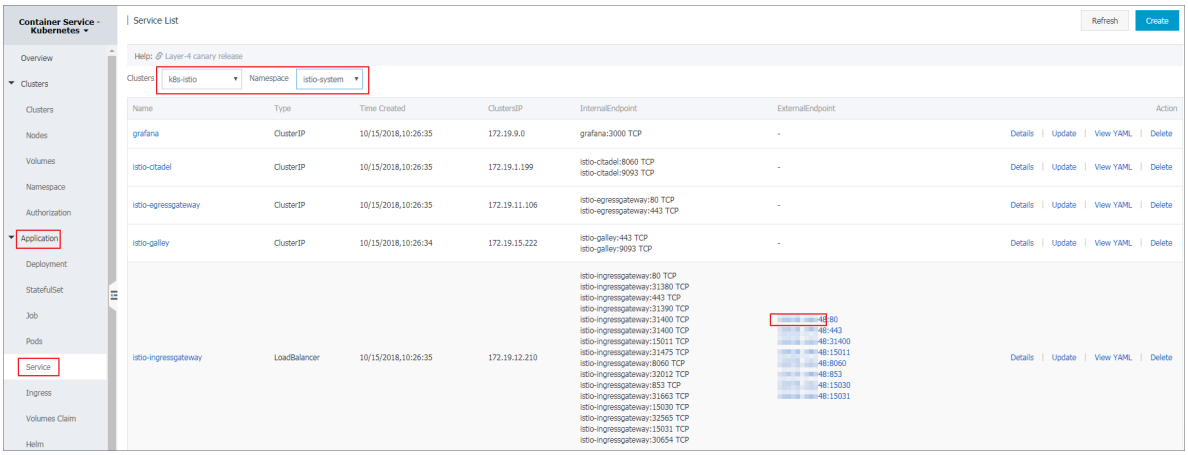
Alibaba Cloud Container Service has enabled the Initializers plug-in by default for Kubernetes clusters if the cluster version is later than 1.8. No other configurations are needed.

Note:

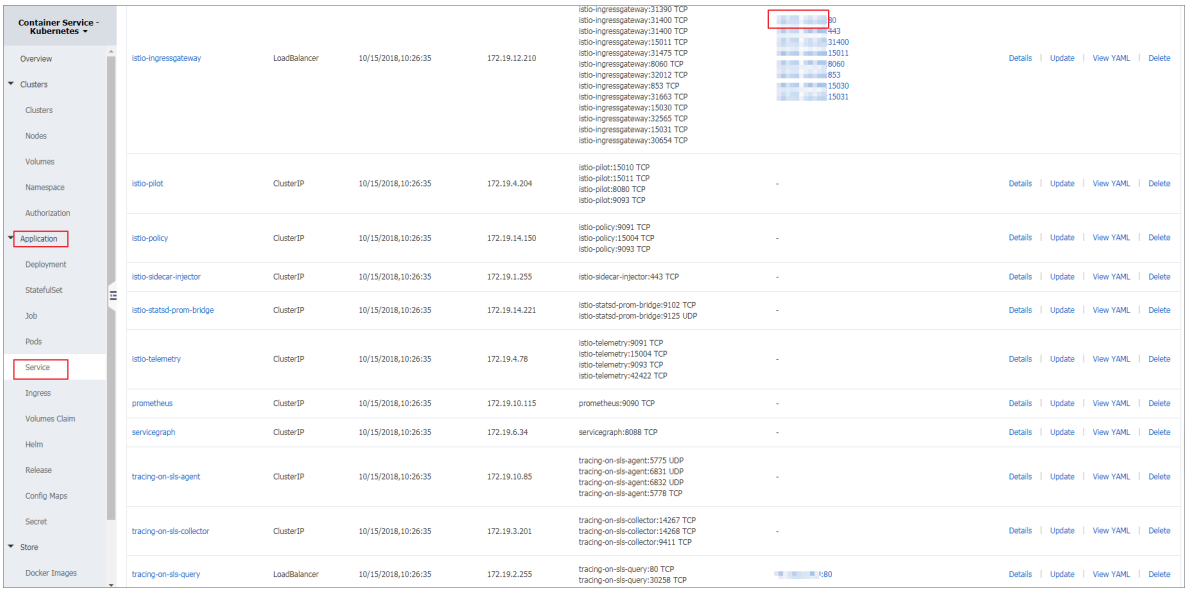
After you deploy the Istio, a sidecar is injected to each pod to take over the service communication. Therefore, we recommend that you verify this in the independent test environment.

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/getLatestIstio | 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 environment 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-initializer.yaml     ##
Deploy Istio initializer 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 LoadBalancer 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
```

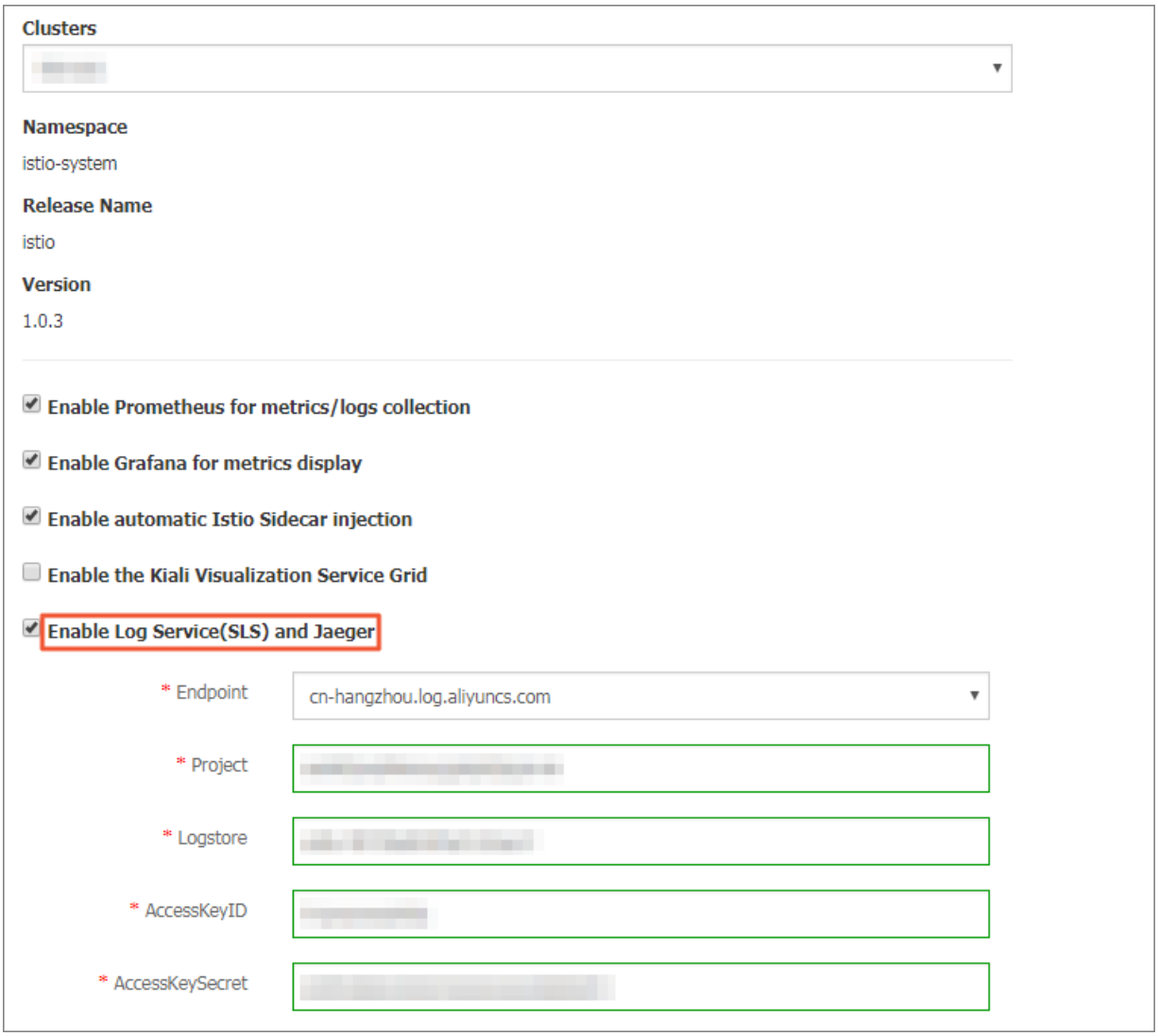
After all the pods are in the running status, the Istio deployment is finished.

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
```

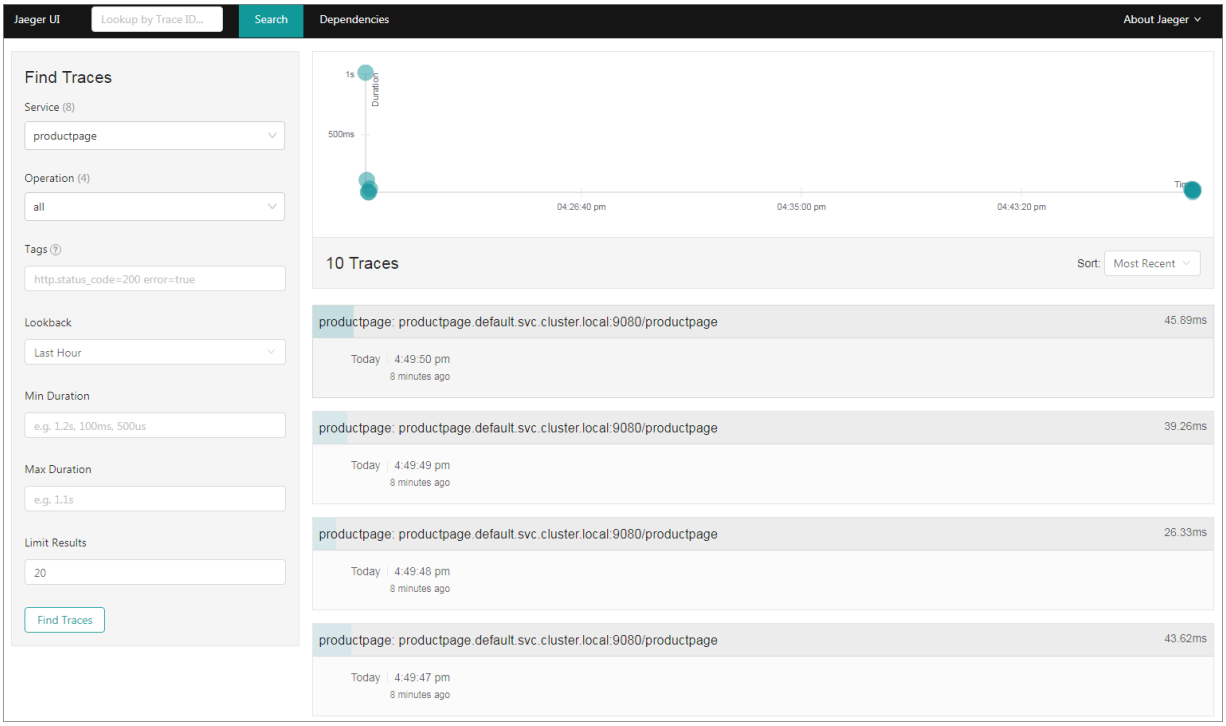
If the preceding command cannot obtain the external IP address, run the following command to obtain the corresponding address.

```
export GATEWAY_URL=$(kubectl get ingress -o wide -o jsonpath={.items[0
].status.loadBalancer.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_URL}/
productpage
```

You can open `http://${GATEWAY_URL}/productpage` 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.

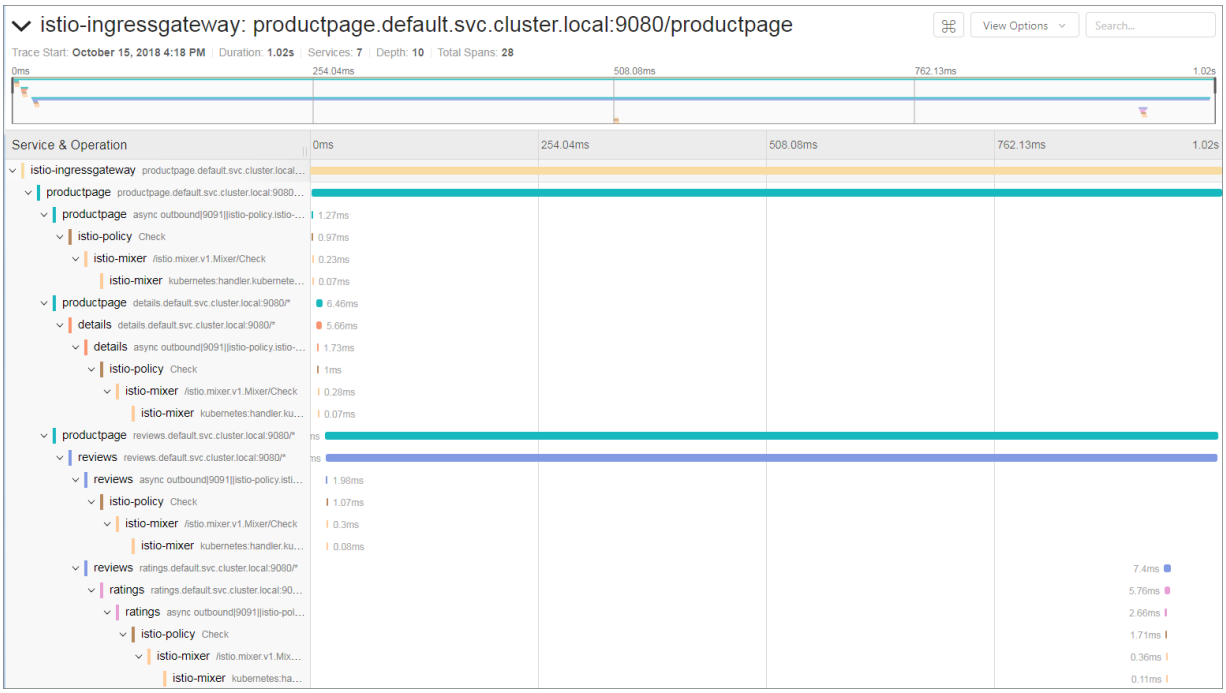
```
kubectl port-forward -n istio-system $(kubectl get pod -n istio-system
 -l app=jaeger -o jsonpath='{.items[0].metadata.name}') 16686:16686 &
```

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.

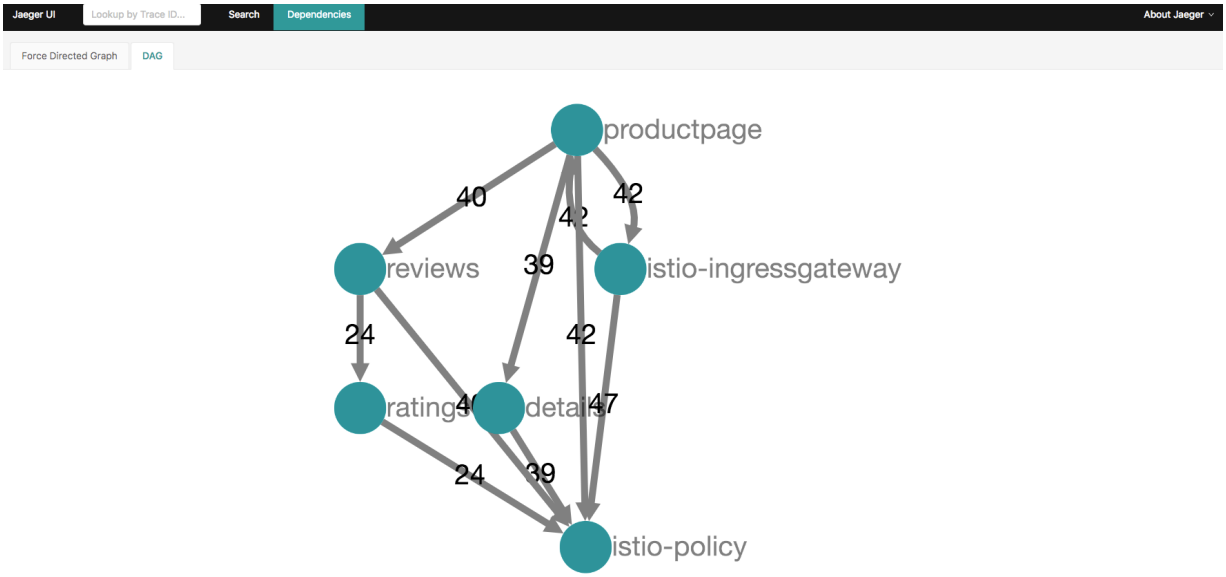
```
$ kubectl get svc -n istio-system
NAME               TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                                                        AGE
jaeger-agent       ClusterIP      None            <none>
5775/UDP,6831/UDP,6832/UDP                                     1h
jaeger-collector   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.

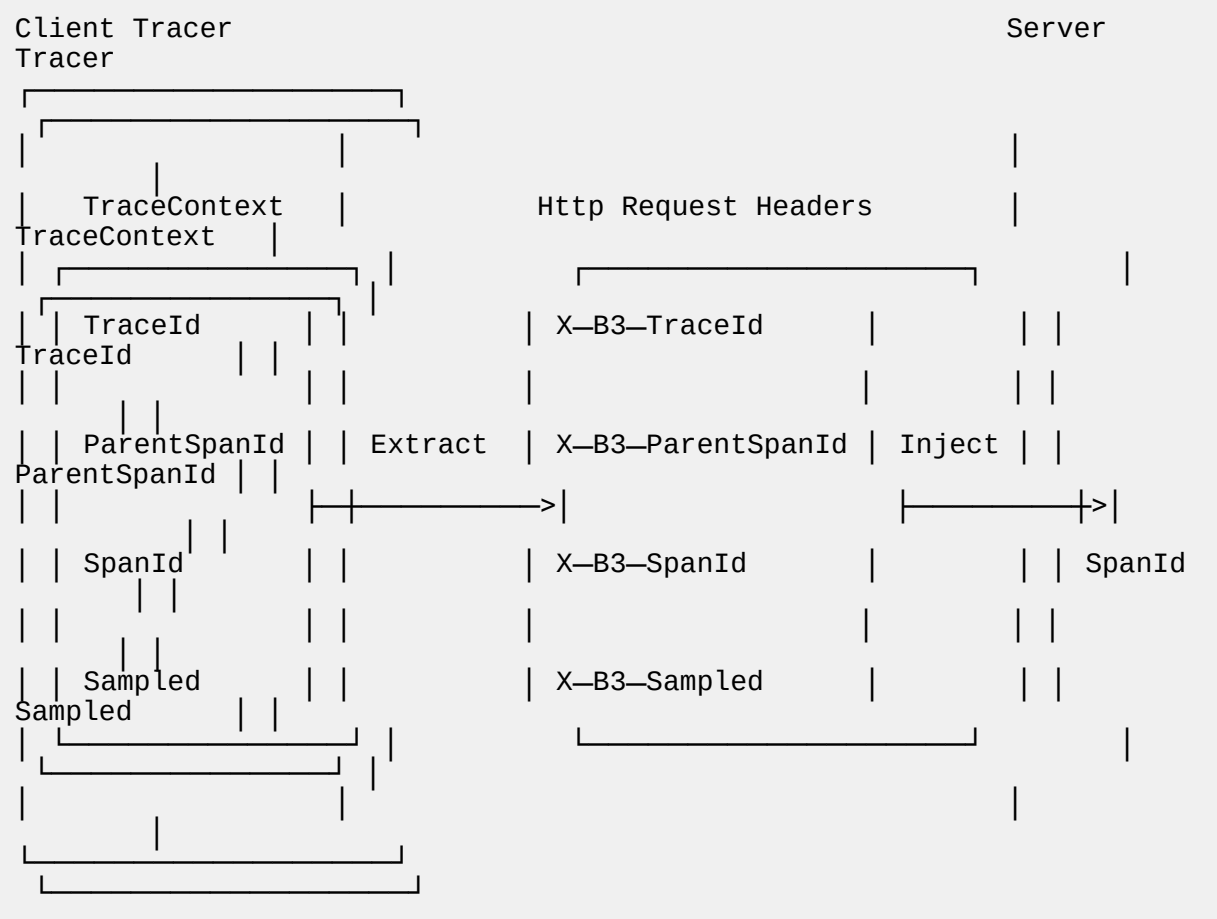
Whenever a microservice initiates an external call, the client Envoy will create a new span. A span represents the complete interaction process between a collection of microservices, starting from a caller (client) sending a request to receiving the response from the server.

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-parentspanid`
- `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.

6 DevOps