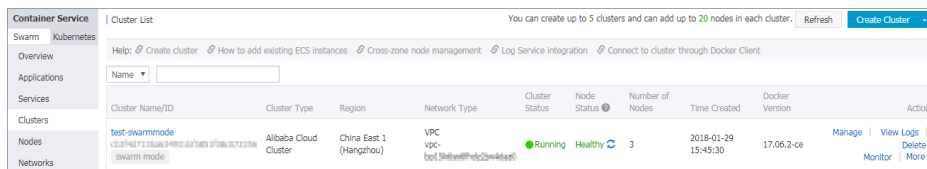


Container Service

Best Practices



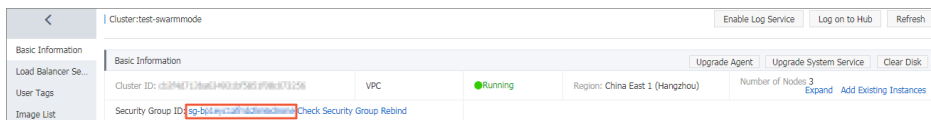
Configure security group rules

The Concourse component ATC monitors the port 8080 by default. Therefore, you must configure the inbound permissions of port 8080 for the cluster security group. For more information about the principle of Concourse, see [Concourse architecture](#).

In the Container Service console, click **Clusters** in the left-side navigation pane under **Swarm**.

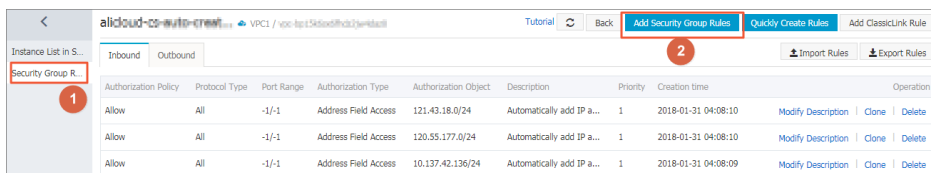
Click **Manage** at the right of the created cluster.

On the **Basic Information** page, click the security group ID.



Click **Security Group Rules** in the left-side navigation pane.

Click **Add Security Group Rules**.



Configure the inbound permissions of port 8080 for the security group and then click **OK**.

Add Security Group Rules

NIC: Intranet

Rule Direction: Inbound

Authorization Policy: Allow

Protocol Type: Custom TCP

* Port Range: 8080/8080

Priority: 1

Authorization Type: Address Field Acc

* Authorization Object: 0.0.0.0/0

Description:

It must contain 2-256 characters and it cannot begin with http:// or https://

OK Cancel

Create keys in the ECS instance

You must generate three private keys for running Concourse safely. For the specific functions for these keys, see **Generating Keys** in the Standalone Binary.

Log on to the Elastic Compute Service (ECS) instance. In the root directory, create the directories keys/web and keys/worker. You can run the following command to create these two directories rapidly.

```
mkdir -p keys/web keys/worker
```

Run the following command to generate three private keys.

```
ssh-keygen -t rsa -f tsa_host_key -N ""
ssh-keygen -t rsa -f worker_key -N ""
ssh-keygen -t rsa -f session_signing_key -N ""
```

Copy the certificate to the corresponding directory.

```
cp ./keys/worker/worker_key.pub ./keys/web/authorized_worker_keys
cp ./keys/web/tsa_host_key.pub ./keys/worker
```

Deploy Concourse CI

Log on to the Container Service console.

Click **Configurations** in the left-side navigation pane under **Swarm**.

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

Enter **CONCOURSE_EXTERNAL_URL** as the **Variable Name** and **http://your-ecs-public-ip:8080** as the **Variable Value**.

* File Name:
The configuration file name should contain 1 to 32 characters.

Description:
The description can contain up to 128 characters.

Configuration: [Edit JSON File](#)

Variable Name	Variable Value	Action
CONCOURSE_EXTERNAL_URL	http://your-ecs-public-ip:8080	Edit Delete

The variable key should contain 1 to 32 characters; the variable value should contain 1 to 128 characters. The variable value must be unique. The variable name and variable value cannot be empty.

Click **Applications** in the left-side navigation pane.

Select the cluster used in this example from the **Cluster** list.

Click **Create Application** in the upper-right corner.

Enter the basic information for the application you are about to create.

Select **Create with Orchestration Template**.

Use the following template:

```
version: '2'
services:
  concourse-db:
    image: postgres:9.5
    privileged: true
    environment:
      POSTGRES_DB: concourse
      POSTGRES_USER: concourse
      POSTGRES_PASSWORD: changeme
      PGDATA: /database
  concourse-web:
    image: concourse/concourse
    links: [concourse-db]
    command: web
    privileged: true
    depends_on: [concourse-db]
    ports: ["8080:8080"]
    volumes: ["/root/keys/web:/concourse-keys"]
    restart: unless-stopped # required so that it retries until concourse-db comes up
    environment:
      CONCOURSE_BASIC_AUTH_USERNAME: concourse
      CONCOURSE_BASIC_AUTH_PASSWORD: changeme
      CONCOURSE_EXTERNAL_URL: "${CONCOURSE_EXTERNAL_URL}"
      CONCOURSE_POSTGRES_HOST: concourse-db
      CONCOURSE_POSTGRES_USER: concourse
      CONCOURSE_POSTGRES_PASSWORD: changeme
      CONCOURSE_POSTGRES_DATABASE: concourse
  concourse-worker:
    image: concourse/concourse
    privileged: true
    links: [concourse-web]
    depends_on: [concourse-web]
    command: worker
    volumes: ["/keys/worker:/concourse-keys"]
    environment:
      CONCOURSE_TSA_HOST: concourse-web
    dns: 8.8.8.8
```

Click **Create and Deploy**. The **Template Parameter** dialog box appears.

Template Parameter

Associated Configuration File:

Parameter	Value	Contrast
size		Miss

Description:

Same

 The selected configuration file contains this variable and the variable values are the same.

Diff

 The selected configuration file contains this variable but the variable values are different.

Miss

 The selected configuration file does not contain this variable.

Replace Variable

OK

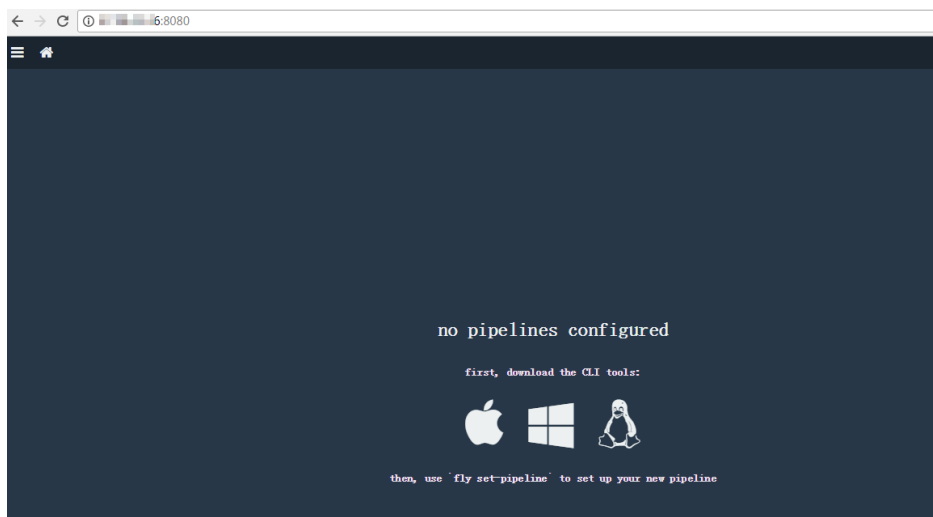
Cancel

Select the configuration file to be associated with from the **Associated Configuration File** list.

Click **Replace Variable** and click **OK**.

After the application is created, the following three services are started: concourse-worker, concourse-db, and concourse-web.

Then, the Concourse CI deployment is finished. Open <http://your-ecs-public-ip:8080> in the browser to access the Concourse CI.



Run a CI task (Hello world)

In the browser opened in the last section, download the CLI corresponding to your operating system and install the CLI client. Use ECS (Ubuntu16.04) as an example.

For Linux and Mac OS X systems, you must add the execution permissions to the downloaded FLY CLI file first. Then, install the CLI to the system and add it to \$PATH.

```
chmod +x fly
install fly /usr/local/bin/fly
```

After the installation, you can check the version.

```
$fly -v
3.4.0
```

Connect to the target. The username and password are concourse and changeme by default.

```
$ fly -t lite login -c http://your-ecs-public-ip:8080
in to team 'main'
username: concourse
password:
saved
```

Save the following configuration template as hello.yml.

```
jobs:
- name: hello-world
plan:
- task: say-hello
config:
platform: linux
image_resource:
type: docker-image
source: {repository: ubuntu}
run:
path: echo
args: ["Hello, world!"]
```

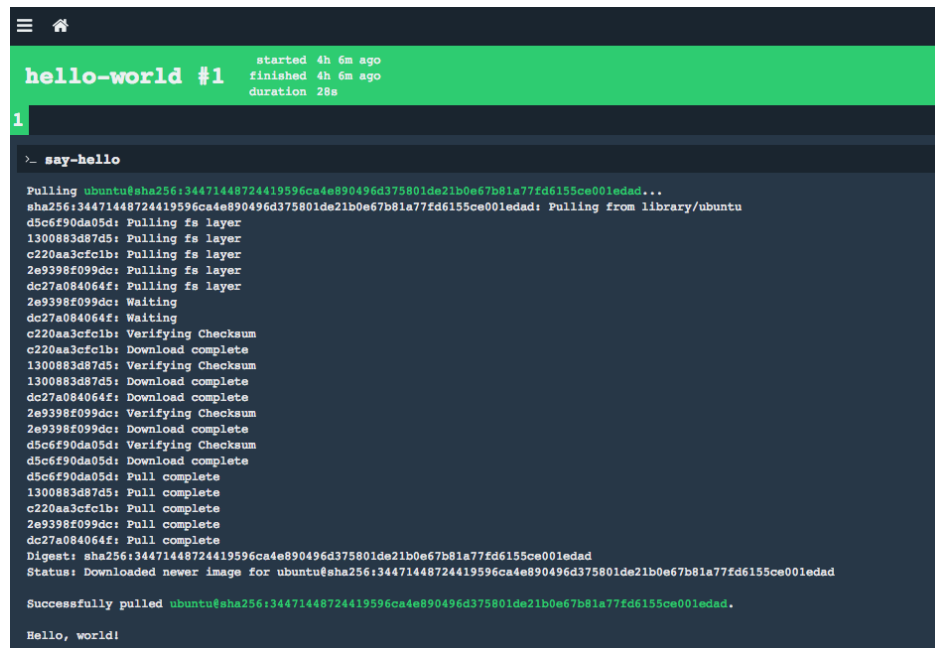
Register the task.


```
fly -t lite set-pipeline -p hello-world -c hello.yml
```

Start the task.

```
fly -t lite unpause-pipeline -p hello-world
```

The page indicating the successful execution is as follows:

The screenshot shows a terminal window with a green header bar. The header bar contains the text 'hello-world #1' on the left and 'started 4h 6m ago', 'finished 4h 6m ago', and 'duration 28s' on the right. Below the header bar, the terminal shows the command 'say-hello' being executed. The output of the command is a long string of text indicating the pulling of an Ubuntu image from a library, followed by a series of 'Pulling fs layer' and 'Pull complete' messages for various layers. The final output is 'Hello, world!'.

```
started 4h 6m ago
finished 4h 6m ago
duration 28s

1

> say-hello

Pulling ubuntu@sha256:34471448724419596ca4e890496d375801de21b0e67b81a77fd6155ce001edad...
sha256:34471448724419596ca4e890496d375801de21b0e67b81a77fd6155ce001edad: Pulling from library/ubuntu
d5c6f90da05d: Pulling fs layer
1300883d87d5: Pulling fs layer
c220aa3cfc1b: Pulling fs layer
2e9398f099dc: Pulling fs layer
dc27a084064f: Pulling fs layer
2e9398f099dc: Waiting
dc27a084064f: Waiting
c220aa3cfc1b: Verifying Checksum
c220aa3cfc1b: Download complete
1300883d87d5: Verifying Checksum
1300883d87d5: Download complete
dc27a084064f: Download complete
2e9398f099dc: Verifying Checksum
2e9398f099dc: Download complete
d5c6f90da05d: Verifying Checksum
d5c6f90da05d: Download complete
d5c6f90da05d: Pull complete
1300883d87d5: Pull complete
c220aa3cfc1b: Pull complete
2e9398f099dc: Pull complete
dc27a084064f: Pull complete
Digest: sha256:34471448724419596ca4e890496d375801de21b0e67b81a77fd6155ce001edad
Status: Downloaded newer image for ubuntu@sha256:34471448724419596ca4e890496d375801de21b0e67b81a77fd6155ce001edad.

Successfully pulled ubuntu@sha256:34471448724419596ca4e890496d375801de21b0e67b81a77fd6155ce001edad.

Hello, world!
```

For more information about the characteristics of Concourse CI, see [Concourse CI project](#).

Implement Istio distributed tracking in Kubernetes

Background

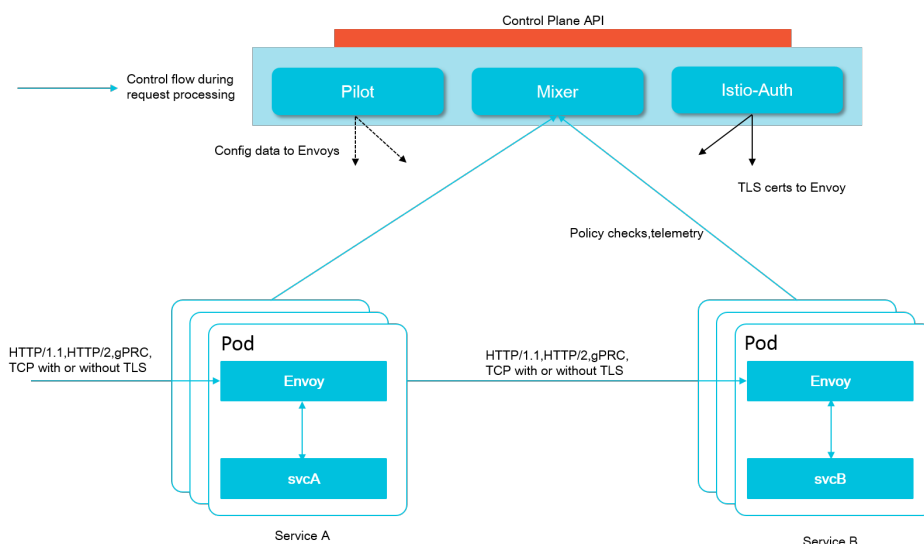
Microservice is a focus in the current era. More and more IT enterprises begin to embrace the microservices. The microservice architecture splits a complex system into several small services and each service can be developed, deployed, and scaled independently. As a heaven-made match, the microservice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

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

In May 2017, Google, IBM, and Lyft published the open-source service network architecture Istio, which provides the connection, management, monitoring, and security protection of microservices. Istio provides an infrastructure layer for services to communicate with each other, decouples the issues such as version management, security protection, failover, monitoring, and telemetry in 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 Kubernetes clusters

Log on to the Container Service console.

Click **Kubernetes** in the left-side navigation pane.

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

Configure the parameters to create a cluster. For how to create a Kubernetes cluster, see [Create a cluster](#).

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

Name	Cluster Name/ID	Cluster Type	Region	Network Type	Cluster Status	Time Created	Kubernetes Version	Action
k8s-test	cs-28027948b47a45028e09a76f9020473d	Kubernetes	China East 1 (Hangzhou)	VPC vpc-4p49a8b4uy7f6k3aw8t	Running	2018-01-23 11:30:50	1.8.4	Cluster Optimization Manage View Logs Delete Dashboard More

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 **kubectl** or **SSH**.

<	Cluster: k8s-test
Basic Information	Basic Information Cluster ID: cs-28027948b47a45028e09a76f9020473d VPC vpc-4p49a8b4uy7f6k3aw8t Running Region: China East 1 (Hangzhou) Scale Cluster
	Connection Information API Server Internet endpoint https://114.93.78.179:6443 API Server Intranet endpoint https://172.16.153.130:6443 Master node SSH IP address 104.188.78.178 Service Access Domain https://cs-28027948b47a45028e09a76f9020473d.cn-hangzhou.aliyuncs.com
	Cluster resource Internet SLB lb-bp15wng3vg9hnhua20p VPC vpc-4p49a8b4uy7f6k3aw8t NAT Gateway n-gw-bp1tali7ya5v8n83oxu3g
	Connect to Kubernetes cluster via kubectl 1. Download the latest kubectl client from the Kubernetes Edition page . 2. Install and set up the kubectl client. For more information, see Installing and Setting Up kubectl 3. Configure the cluster credentials: <pre>mkdir \$HOME/.kube scp root@104.188.78.178:/etc/kubernetes/kube.conf \$HOME/.kube/config</pre> Once configured, you can use kubectl to access Kubernetes clusters from your local machine.

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-system
```

```

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
svc/istio-ingress LoadBalancer 172.21.10.18 101.37.113.231 80:30511/TCP,443:31945/TCP 1m
svc/istio-mixer ClusterIP 172.21.14.221 <none>
9091/TCP,15004/TCP,9093/TCP,9094/TCP,9102/TCP,9125/UDP,42422/TCP 1m
svc/istio-pilot ClusterIP 172.21.4.20 <none> 15003/TCP,443/TCP 1m

```

```

NAME READY STATUS RESTARTS AGE
po/istio-ca-55b954ff7-crsjq 1/1 Running 0 1m
po/istio-ingress-948b746cb-4t24c 1/1 Running 0 1m
po/istio-initializer-6c84859cd-8mvfj 1/1 Running 0 1m
po/istio-mixer-59cc756b48-tkx6c 3/3 Running 0 1m
po/istio-pilot-55bb7f5d9d-wc5xh 2/2 Running 0 1m

```

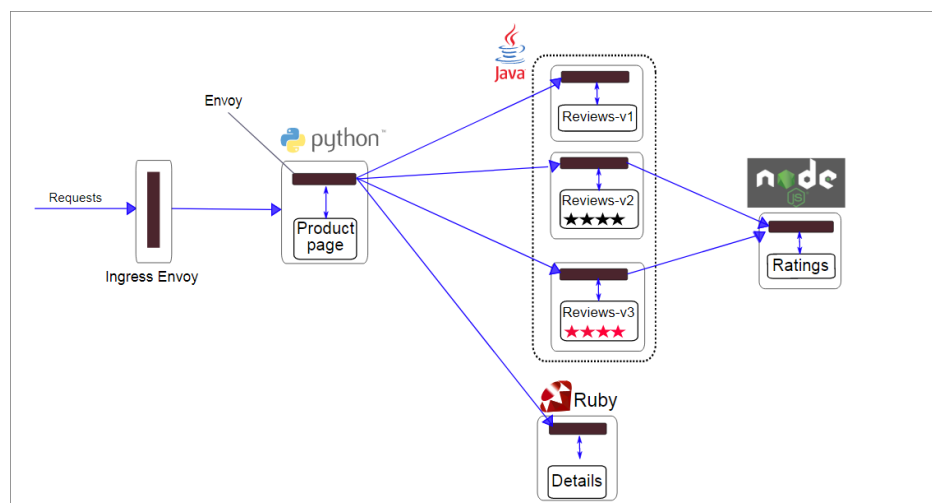
After all the pods are in the running status, the Istio deployment is finished.

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
```

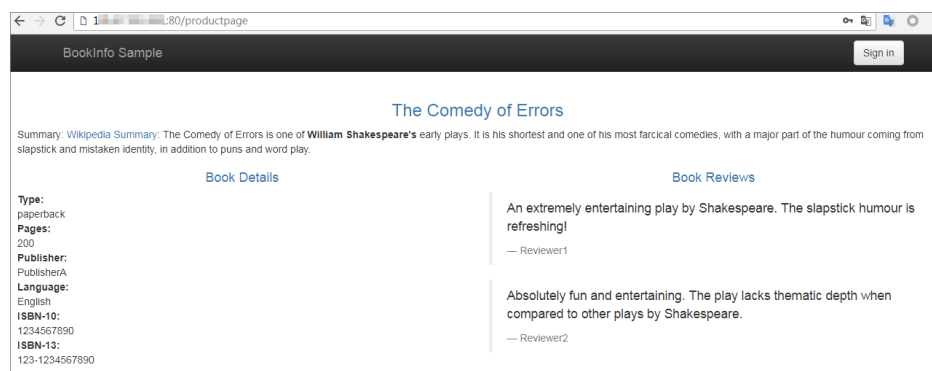
If the preceding command cannot obtain the external IP address, run the following command to obtain the corresponding address.

```
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.

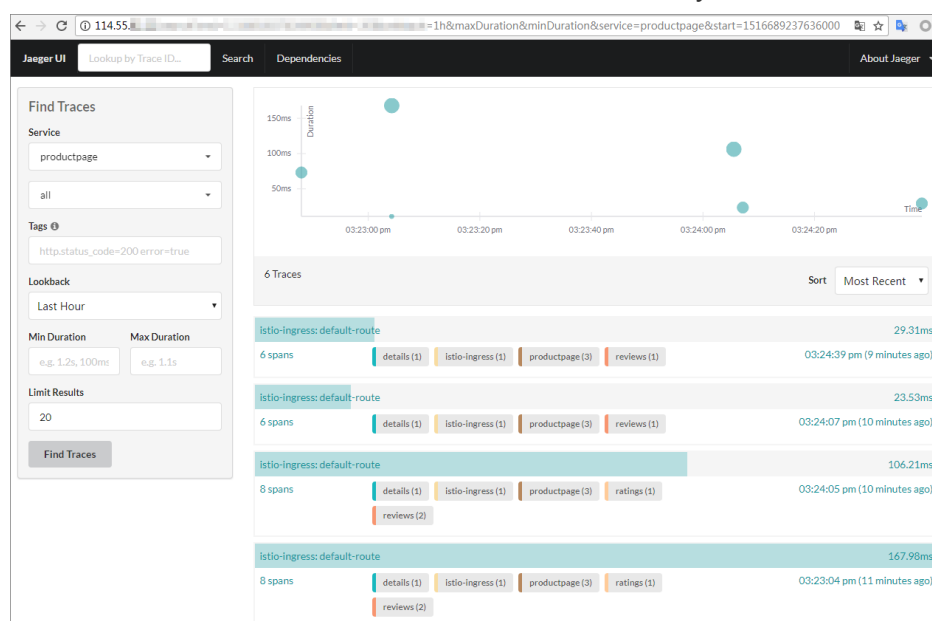
```
kubectl port-forward -n istio-system $(kubectl get pod -n istio-system -l app=jaeger -o jsonpath='{.items[0].metadata.name}') 16686:16686 &
```

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.

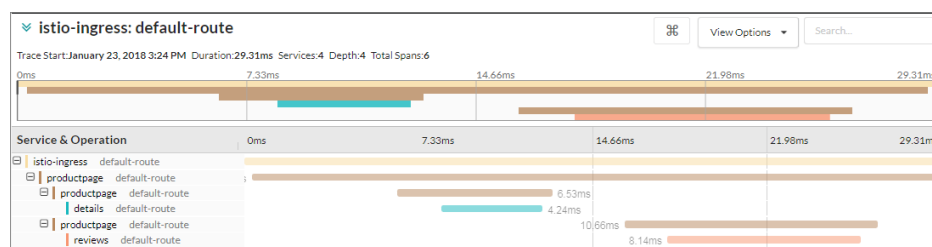
```
$ kubectl get svc -n istio-system
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
jaeger-agent ClusterIP None <none> 5775/UDP,6831/UDP,6832/UDP 1h
jaeger-collector ClusterIP 172.21.10.187 <none> 14267/TCP,14268/TCP,9411/TCP 1h
jaeger-query LoadBalancer 172.21.10.197 114.55.82.11 80:31960/TCP ##The external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

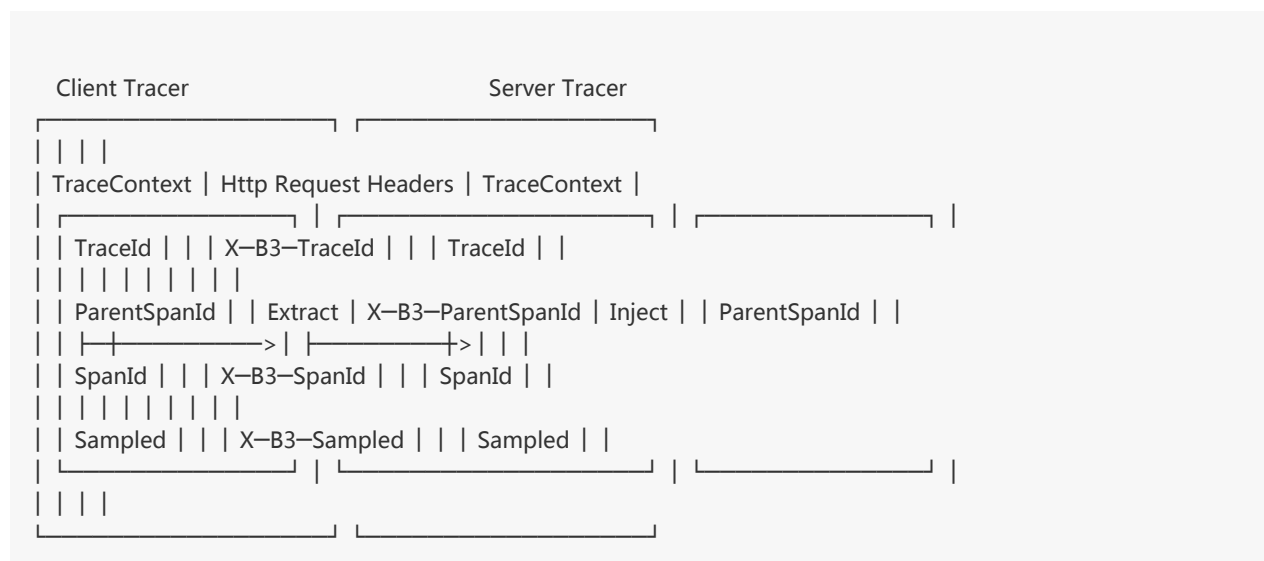
Whenever a microservice initiates an external call, the client Envoy will create a new span. A span represents the complete interaction process between a collection of microservices, starting from a caller (client) sending a request to receiving the response from the server.

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 might 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

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



For specific codes, see the Istio document.

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.

Swarm

Run TensorFlow-based AlexNet in Alibaba Cloud Container Service

AlexNet is a CNN network developed in 2012 by Alex Krizhevsky using five-layer convolution and three-layer ReLU layer, and won the ImageNet competition (ILSVRC). AlexNet proves the effectiveness in classification (15.3% error rate) of CNN, against the 25% error rate by previous image recognition tools. The emergence of this network marks a milestone for deep learning applications in the computer vision field.

AlexNet is also a common performance indicator tool for deep learning framework. TensorFlow provides the `alexnet_benchmark.py` tool to test GPU and CPU performance. This document uses AlexNet as an example to illustrate how to run a GPU application in Alibaba Cloud Container Service easily and quickly.

Prerequisite

Create a GN5 GPU cluster in Container Service console.

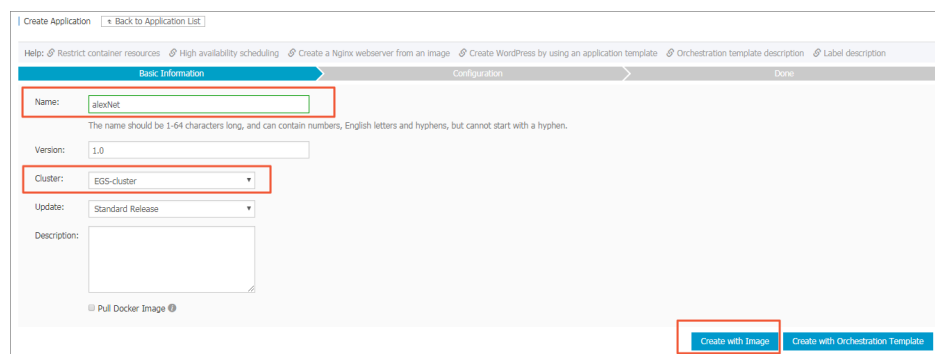
Procedure

Log on to the Container Service console.

Click **Applications** in the left-side navigation pane.

Click **Create Application** in the upper-right corner.

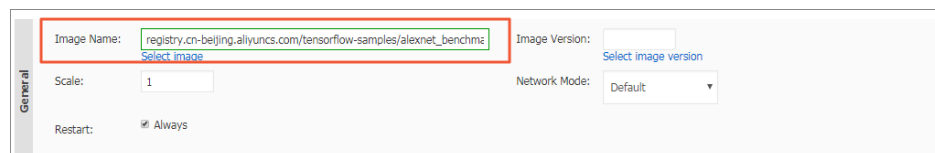
Complete the configurations. Enter the application name (**alexNet** in this example) in the **Name** field and then select the created GN5 GPU cluster from the **Cluster** list.



The screenshot shows the 'Create Application' interface in the Container Service console. The 'Basic Information' tab is selected. The 'Name' field contains 'alexNet'. The 'Cluster' dropdown menu is set to 'EGS-cluster'. The 'Create with Image' button is highlighted with a red box.

Click **Create with Image**.

Enter `registry.cn-beijing.aliyuncs.com/tensorflow-samples/alexnet_benchmark:1.0.0-devel-gpu` in the **Image Name** field.



The screenshot shows the 'General' configuration section. The 'Image Name' field is set to 'registry.cn-beijing.aliyuncs.com/tensorflow-samples/alexnet_benchmark:1.0.0-devel-gpu'. The 'Scale' is set to 1. The 'Restart' policy is set to 'Always'.

In the **Container** section, enter the command in the **Command** field. For example, enter `python /alexnet_benchmark.py --batch_size 128 --num_batches 100`.

Container

Command:

Entrypoint:

CPU Limit: Memory Limit: MB

Capabilities:

Container Config: ☐ stdin ☐ tty



Click the  button in the **Label** section. Enter the Alibaba Cloud gpu extension label. Enter **aliyun.gpu** in the **Tag Name** field, and the number of scheduling GPUs (**1** in this example) in the **Tag Value** field.

Label

Labels:

Tag Name:

Tag Value:

Click **Create** after configuring the application.

You can view the created **alexNet** application on the **Application List** page.

Application List

Refresh

Create Application

Help: [Create an application](#) [Change application configurations](#) [Simple route blue-green release policy](#) [Container auto scaling](#)

Cluster:

EGS-cluster

Hide System Applications

Hide Offline Applications

Hide Online Applications

Name

Name	Description	Status	Container Status	Time Created	Time Updated	Action
alexNet		Ready	Ready:1 Stop:0	2017-11-20 10:16:06	2017-11-20 10:16:06	Stop Update Delete Redeploy Events

Click the application name **alexNet**.

Click the **Logs** tab.

Services

Containers

Logs

Events

Routes

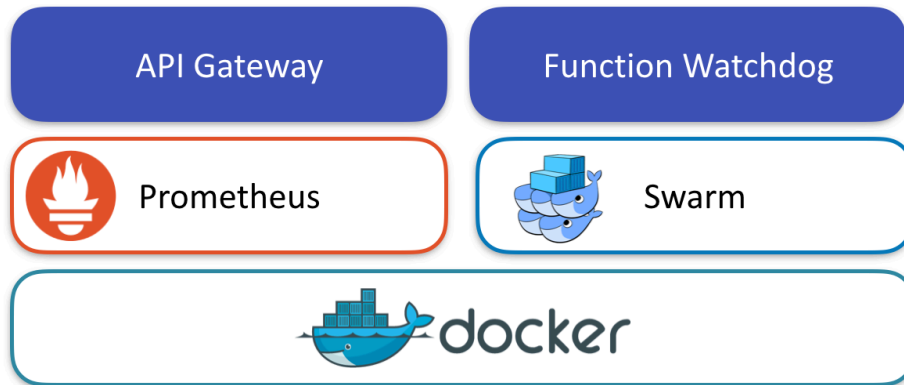
Entries Per Container: 100Items

Filter by Start Time:

alexNet_alexNet-1	2017-11-20T02:30:16.327234862Z	I tensorflow/stream_executor/dso_loader.cc:135] successfully opened CUDA library libcudart.so.8.0 locally
alexNet_alexNet-1	2017-11-20T02:30:17.060750558Z	WARNING:tensorflow:From /alexNet_benchmark.py:204: initialize_all_variables (from tensorflow.python.ops.variables) is deprecated and will be removed after 2017-03-02. Instructions for updating:
alexNet_alexNet-1	2017-11-20T02:30:17.06075480Z	Use tf.global_variables_initializer() instead.
alexNet_alexNet-1	2017-11-20T02:30:17.060687838Z	W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE3 instructions, but these are available on your machine and could speed up CPU computation.
alexNet_alexNet-1	2017-11-20T02:30:17.060622032Z	W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instructions, but these are available on your machine and could speed up CPU computation.
alexNet_alexNet-1	2017-11-20T02:30:17.060623185Z	W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instructions, but these are available on your machine and could speed up CPU computation.
alexNet_alexNet-1	2017-11-20T02:30:17.060638738Z	W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instructions, but these are available on your machine and could speed up CPU computation.
alexNet_alexNet-1	2017-11-20T02:30:17.060634216Z	W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructions, but these are available on your machine and could speed up CPU computation.
alexNet_alexNet-1	2017-11-20T02:30:17.060648662Z	W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instructions, but these are available on your machine and could speed up CPU computation.
alexNet_alexNet-1	2017-11-20T02:30:17.100807280Z	I tensorflow/stream_executor/cuda_gpu_executor.cc:918] successful Nvml device read from Sysfs had negative value (-1), but there must be at least one Nvml device, so returning Nvml device 0 with properties:
alexNet_alexNet-1	2017-11-20T02:30:17.151334743Z	name: Tesla P100-PCIE-16GB
alexNet_alexNet-1	2017-11-20T02:30:17.151337266Z	major: 4 minor: 0 memoryClockRate (GHz) 1.3285
alexNet_alexNet-1	2017-11-20T02:30:17.151339880Z	pciBusID 0000:00:08:0
alexNet_alexNet-1	2017-11-20T02:30:17.151342382Z	Total Memory: 15.89618
alexNet_alexNet-1	2017-11-20T02:30:17.151344078Z	Free memory: 15.43618
alexNet_alexNet-1	2017-11-20T02:30:17.151346980Z	I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] DMA: 0
alexNet_alexNet-1	2017-11-20T02:30:17.151349277Z	I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0: Y
alexNet_alexNet-1	2017-11-20T02:30:17.151351568Z	I tensorflow/core/common_runtime/gpu/gpu_device.cc:973] Creating TensorFlow device (/gpu:0) -> (device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:08:0)
alexNet_alexNet-1	2017-11-20T02:30:25.470520880Z	conv1 [128, 56, 56, 64]
alexNet_alexNet-1	2017-11-20T02:30:25.470530752Z	pool1 [128, 27, 27, 64]
alexNet_alexNet-1	2017-11-20T02:30:25.470537682Z	conv2 [128, 27, 27, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470543782Z	pool2 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470550462Z	conv3 [128, 13, 13, 192]
alexNet_alexNet-1	2017-11-20T02:30:25.470551582Z	conv4 [128, 13, 13, 192]
alexNet_alexNet-1	2017-11-20T02:30:25.470553032Z	conv5 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470554032Z	conv6 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470555032Z	conv7 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470556032Z	conv8 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470557032Z	conv9 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470558032Z	conv10 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470559032Z	conv11 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470560032Z	conv12 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470561032Z	conv13 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470562032Z	conv14 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470563032Z	conv15 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470564032Z	conv16 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470565032Z	conv17 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470566032Z	conv18 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470567032Z	conv19 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470568032Z	conv20 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470569032Z	conv21 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470570032Z	conv22 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470571032Z	conv23 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470572032Z	conv24 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470573032Z	conv25 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470574032Z	conv26 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470575032Z	conv27 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470576032Z	conv28 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470577032Z	conv29 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470578032Z	conv30 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470579032Z	conv31 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470580032Z	conv32 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470581032Z	conv33 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470582032Z	conv34 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470583032Z	conv35 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470584032Z	conv36 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470585032Z	conv37 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470586032Z	conv38 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470587032Z	conv39 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470588032Z	conv40 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470589032Z	conv41 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470590032Z	conv42 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470591032Z	conv43 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470592032Z	conv44 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470593032Z	conv45 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470594032Z	conv46 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470595032Z	conv47 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470596032Z	conv48 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470597032Z	conv49 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470598032Z	conv50 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470599032Z	conv51 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470600032Z	conv52 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470601032Z	conv53 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470602032Z	conv54 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470603032Z	conv55 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470604032Z	conv56 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470605032Z	conv57 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470606032Z	conv58 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470607032Z	conv59 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470608032Z	conv60 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470609032Z	conv61 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470610032Z	conv62 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470611032Z	conv63 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470612032Z	conv64 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470613032Z	conv65 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470614032Z	conv66 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470615032Z	conv67 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470616032Z	conv68 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470617032Z	conv69 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470618032Z	conv70 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470619032Z	conv71 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470620032Z	conv72 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470621032Z	conv73 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470622032Z	conv74 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470623032Z	conv75 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470624032Z	conv76 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470625032Z	conv77 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470626032Z	conv78 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470627032Z	conv79 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470628032Z	conv80 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470629032Z	conv81 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470630032Z	conv82 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470631032Z	conv83 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470632032Z	conv84 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470633032Z	conv85 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470634032Z	conv86 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470635032Z	conv87 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470636032Z	conv88 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470637032Z	conv89 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470638032Z	conv90 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470639032Z	conv91 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470640032Z	conv92 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470641032Z	conv93 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470642032Z	conv94 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470643032Z	conv95 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470644032Z	conv96 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470645032Z	conv97 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470646032Z	conv98 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470647032Z	conv99 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470648032Z	conv100 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470649032Z	conv101 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470650032Z	conv102 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470651032Z	conv103 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470652032Z	conv104 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470653032Z	conv105 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470654032Z	conv106 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470655032Z	conv107 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470656032Z	conv108 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470657032Z	conv109 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470658032Z	conv110 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470659032Z	conv111 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470660032Z	conv112 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470661032Z	conv113 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470662032Z	conv114 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470663032Z	conv115 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470664032Z	conv116 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470665032Z	conv117 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470666032Z	conv118 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470667032Z	conv119 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470668032Z	conv120 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470669032Z	conv121 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470670032Z	conv122 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470671032Z	conv123 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470672032Z	conv124 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470673032Z	conv125 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470674032Z	conv126 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470675032Z	conv127 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470676032Z	conv128 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470677032Z	conv129 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470678032Z	conv130 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470679032Z	conv131 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470680032Z	conv132 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470681032Z	conv133 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470682032Z	conv134 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470683032Z	conv135 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470684032Z	conv136 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470685032Z	conv137 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470686032Z	conv138 [128, 13, 13, 128]
alexNet_alexNet-1	2017-11-20T02:30:25.470687032Z	conv139 [128, 13, 13, 128]
alexNet_alexNet-1		

2. Implement the scheduling capability of functions in a simple way by using the resource scheduling of Docker swarm mode clusters and the Server Load Balancer capability of routing mesh. Each function corresponds to one service in the Docker cluster.

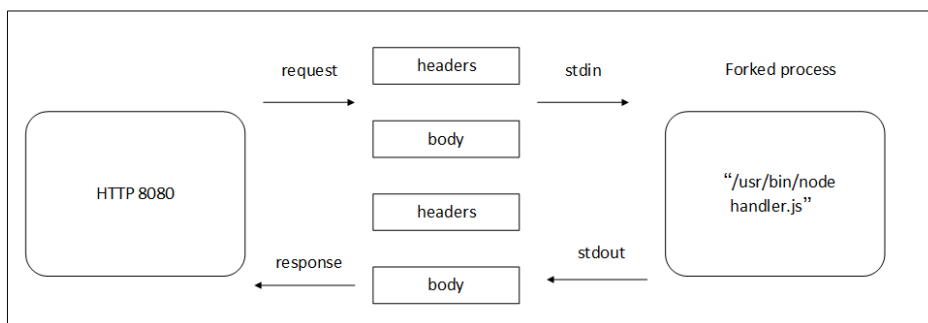
Function call monitoring and auto scaling are implemented by using Prometheus.



The design architecture is simple.

- The API Gateway is in charge of receiving service calls and routing requests to backend functions for implementation. It also collects service call indicators and sends the indicators to Prometheus. Prometheus calls back the API Gateway based on the number of service calls within a period of time to automatically scale the number of instances in the service container.

Function Watchdog forwards HTTP requests as process calls, passes the requested data to the process by STDIN, and then returns the process STDOUT to the caller as the HTTP response result. Package the function process and Function Watchdog into a container image for deployment. The call process is as follows.



Install FaaS locally

Prepare a local Docker swarm mode cluster first. If no Docker swarm mode cluster is in place, you can install the latest Docker Engine and run the following command:

docker swarm init

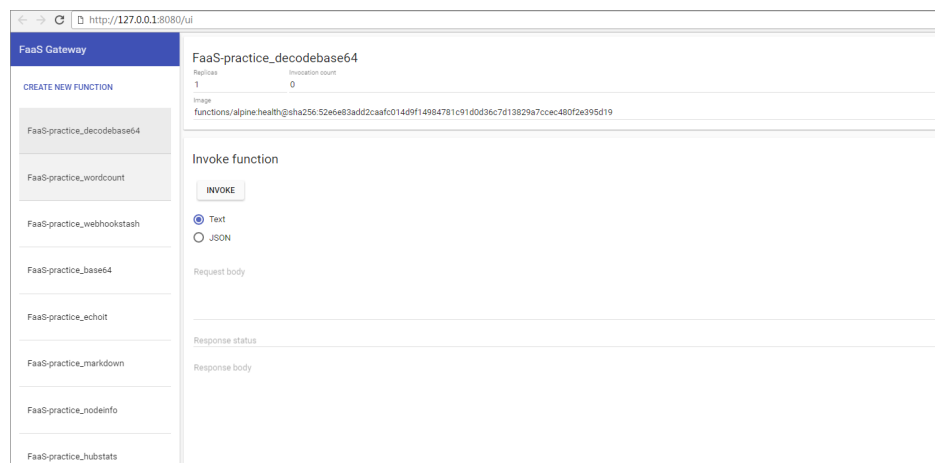
Run the following command to deploy FaaS:

```
git clone https://github.com/alexellis/faas
cd faas
./deploy_stack.sh
```

After the deployment, you can run the following command to check the FaaS status:

```
$ docker stack services func
ID NAME MODE REPLICAS IMAGE
1a8b2tb19ulk func_gateway replicated 1/1 functions/gateway:0.5.6
4jdexem6kppg func_webhookstash replicated 1/1 functions/webhookstash:latest
9ju4er5jur9l func_wordcount replicated 1/1 functions/alpine:health
e190suippx7i func_markdown replicated 1/1 alexellis2/faas-markdownrender:latest
l70j4c7kf99t func_alertmanager replicated 1/1 functions/alertmanager:latest
mgujgoa2u8f3 func_decodebase64 replicated 1/1 functions/alpine:health
o44asbnhqbda func_hubstats replicated 1/1 alexellis2/faas-dockerhubstats:latest
q8rx49ow3may func_echoit replicated 1/1 functions/alpine:health
t1ao5psnsj0s func_base64 replicated 1/1 functions/alpine:health
vj5z7rpdlo48 func_prometheus replicated 1/1 functions/prometheus:latest
xmwzd4z7l4dv func_nodeinfo replicated 1/1 functions/nodeinfo:latest
```

Then, access <http://127.0.0.1:8080/ui> FaaS in the browser.



Test FaaS in Alibaba Cloud

Limits

Make sure you have the conditions for creating a swarm mode cluster.

- By default, you can create at most 5 clusters in all regions, and add at most 20 nodes to each cluster. To create more clusters or add more nodes to a cluster, open a ticket.
- The Server Load Balancer instance created with the cluster only supports the Pay-As-You-Go billing method.

Procedure

Create a swarm mode cluster.

FaaS is deployed based on the Docker swarm mode cluster. Create a swarm mode cluster in Alibaba Cloud Container Service first.

The screenshot shows the 'Set Basic Information' step in the Alibaba Cloud Container Service console. The 'Cluster Name' is 'test'. The 'Region' is 'China East 1 (Hangzhou)'. The 'Zone' is 'China East 1 Zone F'. The 'Cluster Mode' is 'Swarm Mode (Beta)'. The 'Network Type' is 'VPC'. The 'VPC' is 'vpc-bp1w92wrgz01fm6...'. The 'Create Cluster' button is visible.

Create an application by using an orchestration template. For more information, see [Create an application by using an orchestration template](#).

The orchestration sample is as follows:

```
version: "3"
services:

# Core API services are pinned, HA is provided for functions.
gateway:
volumes:
- "/var/run/docker.sock:/var/run/docker.sock"
ports:
- 8080:8080
labels:
aliyun.routing.port_8080: faas
image: functions/gateway:0.5.6
networks:
- functions
environment:
dnsrr: "true" # Temporarily use dnsrr in place of VIP while issue persists on PWD
deploy:
placement:
constraints: [node.role == manager]

prometheus:
image: functions/prometheus:latest # autobuild from Dockerfile in repo.
```

```

command: "-config.file=/etc/prometheus/prometheus.yml -storage.local.path=/prometheus -
storage.local.memory-chunks=10000 --alertmanager.url=http://alertmanager:9093"
ports:
- 9090:9090
depends_on:
- gateway
- alertmanager
labels:
aliyun.routing.port_9090: prometheus
environment:
no_proxy: "gateway"
networks:
- functions
deploy:
placement:
constraints: [node.role == manager]

alertmanager:
image: functions/alertmanager:latest # autobuild from Dockerfile in repo.
environment:
no_proxy: "gateway"
command:
- '-config.file=/alertmanager.yml'
networks:
- functions
ports:
- 9093:9093
deploy:
placement:
constraints: [node.role == manager]

# Sample functions go here.

# Service label of "function" allows functions to show up in UI on http://gateway:8080/
webhookstash:
image: functions/webhookstash:latest
labels:
function: "true"
depends_on:
- gateway
networks:
- functions
environment:
no_proxy: "gateway"
https_proxy: $https_proxy

# Pass a username as an argument to find how many images user has pushed to Docker Hub.
hubstats:
image: alexellis2/faas-dockerhubstats:latest
labels:
function: "true"
depends_on:
- gateway
networks:
- functions
environment:

```



```
no_proxy: "gateway"
https_proxy: $https_proxy

# Node.js gives OS info about the node (Host)
nodeinfo:
image: functions/nodeinfo:latest
labels:
function: "true"
depends_on:
- gateway
networks:
- functions
environment:
no_proxy: "gateway"
https_proxy: $https_proxy

# Uses `cat` to echo back response, fastest function to execute.
echoit:
image: functions/alpine:health
labels:
function: "true"
depends_on:
- gateway
networks:
- functions
environment:
fprocess: "cat"
no_proxy: "gateway"
https_proxy: $https_proxy

# Counts words in request with `wc` utility
wordcount:
image: functions/alpine:health
labels:
function: "true"
com.faaS.max_replicas: "10"
depends_on:
- gateway
networks:
- functions
environment:
fprocess: "wc"
no_proxy: "gateway"
https_proxy: $https_proxy

# Calculates base64 representation of request body.
base64:
image: functions/alpine:health
labels:
function: "true"
depends_on:
- gateway
networks:
- functions
environment:
fprocess: "base64"
```

```
no_proxy: "gateway"
https_proxy: $https_proxy

# Decodes base64 representation of request body.
decodebase64:
image: functions/alpine:health
labels:
function: "true"
depends_on:
- gateway
networks:
- functions
environment:
fprocess: "base64 -d"
no_proxy: "gateway"
https_proxy: $https_proxy

# Converts body in (markdown format) -> (html)
markdown:
image: alexellis2/faas-markdownrender:latest
labels:
function: "true"
depends_on:
- gateway
networks:
- functions
environment:
no_proxy: "gateway"
https_proxy: $https_proxy

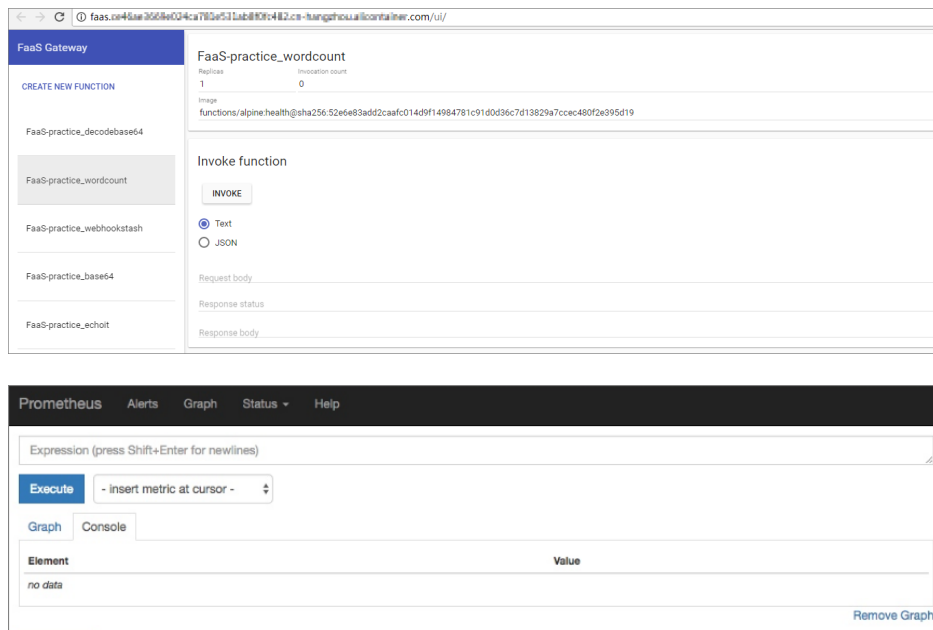
networks:
functions:
driver: overlay
```

Compared with the local deployment, only two labels are added, defining the routes of API Gateway and Prometheus.

- aliyun.routing.port_8080: Faas: Virtual domain name of the API Gateway.
- aliyun.routing.port_9090: Prometheus: Virtual domain name of the Prometheus service.

Click the application name on the **Application List** page and then click the **Routes** tab.

Click the route address to access the API Gateway and Prometheus service interfaces of Faas.



Subsequent operations

You can test the service scalability based on this method. For more information, see [Open-source GitHub project address](#).

Best practices for restarting nodes

Restarting nodes directly might cause an exception in clusters. For example, for the Manager nodes in swarm mode clusters, if the number of healthy nodes is less than 2, the cluster might be incapable of self-cure and then become unavailable. In the context of Alibaba Cloud use cases, this document introduces the best practices for restarting nodes in the situations such as Container Service is actively operated and maintained.

Check the high availability configurations of business

Before restarting Container Service nodes, we recommend checking or modifying the following business configurations. In this way, restarting nodes cannot cause the exception of a single node and the business availability cannot be impaired.

Data persistence strategy of configurations

We recommend the data persistence for external volumes of important data configurations

such as configurations of logs and business. In this way, after the container is restructured, deleting the former container cannot cause the data loss.

For how to use the Container Service data volumes, see [Data volume management](#).

Restart strategy of configurations

We recommend configuring the restart: always restart strategy for the corresponding business services so that containers can be automatically pulled up after the nodes are restarted.

High availability strategy of configurations

We recommend integrating with the product architecture to configure the affinity and mutual exclusion strategies, such as high availability scheduling (`availability:az` property), specified node scheduling (affinity and constraint properties), and specified nodes scheduling (constraint property), for the corresponding businesses. In this way, restarting nodes cannot cause the exception of a single node. For example, for the database business, we recommend the active-standby or multi-instance deployment, and integrating with the preceding characteristics to ensure the different instances are on different nodes and related nodes are not being restarted at the same time.

Best practices

We recommend checking the high availability configurations of business by reading the preceding introductions. Then, complete the following steps in sequence on each node.

Note: Do not perform on multiple nodes at the same time.

Back up snapshots

We recommend creating the latest snapshots for all the related disks of the nodes and then backing up the snapshots. In this way, when starting the shut-down nodes, the exception does not occur because the server is not restarted for a long time and the business availability is not impaired.

Verify the container configuration availability of business (ignore this step if the cluster is a swarm mode cluster)

For a non-swarm mode cluster, restarting the corresponding business containers on nodes ensures the containers can be pulled up again normally.

Note: The minimum control operation unit of swarm mode clusters is service. Therefore, you cannot directly process the business containers by starting or stopping Docker on swarm mode cluster nodes. Otherwise, an error occurs. The correct way is to perform automatic adjustment for business by readjusting the application replicas in Container Service console.

Modify node role (apply to swarm mode clusters)

If the corresponding node is a Manager node in the swarm mode cluster, set the node to a Worker node first.

Verify the running availability of Docker Engine

Try to restart Docker daemon and ensure the Docker Engine can be restarted normally.

Perform related operations and maintenance

Perform the related operations and maintenance in the plan, such as updating business codes, installing system patches, and adjusting system configurations.

Restarting nodes

Restart nodes normally in the console or system.

Check the status after the restart

Check the health status of the nodes and the running status of the business containers in Container Service console after restarting the nodes.

Call back node role (apply to swarm mode clusters)

If the corresponding node is a Manager node in the swarm mode cluster, set the node to a Manager node again.

Use OSSFS data volumes to share WordPress attachments

This document introduces how to share WordPress attachments across different containers by creating OSSFS data volumes in Alibaba Cloud Container Service.

Scenarios

Docker containers simplify WordPress deployment. With Alibaba Cloud Container Service, you can use an orchestration template to deploy WordPress with one click.

Note: For more information, see [Create WordPress with an orchestration template](#).

In this example, the following orchestration template is used to create an application named **wordpress**.

```
web:
  image: registry.aliyuncs.com/acs-sample/wordpress:4.3
  ports:
    - '80'
  environment:
    WORDPRESS_AUTH_KEY: changeme
    WORDPRESS_SECURE_AUTH_KEY: changeme
    WORDPRESS_LOGGED_IN_KEY: changeme
    WORDPRESS_NONCE_KEY: changeme
    WORDPRESS_AUTH_SALT: changeme
    WORDPRESS_SECURE_AUTH_SALT: changeme
    WORDPRESS_LOGGED_IN_SALT: changeme
    WORDPRESS_NONCE_SALT: changeme
    WORDPRESS_NONCE_AA: changeme
  restart: always
  links:
    - 'db:mysql'
  labels:
    aliyun.logs: /var/log
    aliyun.probe.url: http://container/license.txt
    aliyun.probe.initial_delay_seconds: '10'
    aliyun.routing.port_80: http://wordpress
    aliyun.scale: '3'
  db:
    image: registry.aliyuncs.com/acs-sample/mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: password
    restart: always
    labels:
      aliyun.logs: /var/log/mysql
```

This application contains a MySQL container and three WordPress containers (aliyun.scale: '3' is the extension label of Alibaba Cloud Container Service, and specifies the number of containers. For more information about the labels supported by Alibaba Cloud Container Service, see [Label description](#)).

The WordPress containers access MySQL by using a link. The `aliyun.routing.port_80: http://wordpress` label defines the load balancing among the three WordPress containers (for more information, see [Simple routing - supports HTTP and HTTPS](#)).

In this example, the application deployment is simple and the deployed application is of complete features. However, the attachments uploaded by WordPress are stored in the local disk, which means they cannot be shared across different containers or opened when requests are routed to other containers.

Solutions

This document introduces how to use OSSFS data volumes of Alibaba Cloud Container Service to share WordPress attachments across different containers, without any code modifications.

OSSFS data volume, a third-party data volume provided by Alibaba Cloud Container Service, packages various cloud storages (such as Object Storage Service (OSS)) as data volumes and then directly mounts them to the containers. This means the data volumes can be shared across different containers and automatically re-mounted to the containers when the containers are restarted or migrated.

Procedure

Create OSSFS data volumes.

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

Click **Data Volumes** in the left-side navigation pane.

Select the cluster in which you want to create data volumes from the **Cluster** list.

Click **Create** in the upper-right corner to create the OSSFS data volumes.

For how to create OSSFS data volumes, see [Create an OSSFS data volume](#).

In this example, the created OSSFS data volumes are named **wp_upload**. Container Service uses the same name to create data volumes on each node of a cluster.

Data Volume List							Refresh	Create
Help: Data volume guide								
Cluster: test-link								
Node	Volume Name	Driver	Mount Point	Container	Volume Parameters	Action		
test-link-1	fd23b18020644603b0e5d2c...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_web_1		Delete All Volumes with the Same Name		
test-link-2	8c1517c3b3414d605c839649...	Ephemeral Disk	/var/lib/docker/volumes/...	test-cluster-link_redis_...		Delete All Volumes with the Same Name		
test-link-3	f91423c7345bc3cd7c09c78...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_web_1		Delete All Volumes with the Same Name		
test-link-4	wp_upload	OSS File System	/mnt/acs_mnt/ossfs/cjbe...		View	Delete All Volumes with the Same Name		
test-link-5	775c1d0987160e6512ad54c...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_web_3		Delete All Volumes with the Same Name		
test-link-6	a03bbe91cd847704654cc65...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_web_3		Delete All Volumes with the Same Name		
test-link-7	wp_upload	OSS File System	/mnt/acs_mnt/ossfs/cjbe...		View	Delete All Volumes with the Same Name		
test-link-8	0dac5db2abc0c71b8c8eb8f4...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_db_1		Delete All Volumes with the Same Name		
test-link-9	b741328d5f69f781d5cabd7...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_db_1		Delete All Volumes with the Same Name		
test-link-10	76fcb0b0767d72753d52...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_web_2		Delete All Volumes with the Same Name		
test-link-11	44aa4d32f723834b800d7790...	Ephemeral Disk	/var/lib/docker/volumes/...	wordpress_web_2		Delete All Volumes with the Same Name		
test-link-12	wp_upload	OSS File System	/mnt/acs_mnt/ossfs/cjbe...		View	Delete All Volumes with the Same Name		

Use the OSSFS data volumes.

The WordPress attachments are stored in the `/var/www/html/wp-content/uploads` directory by default. In this example, map OSSFS data volumes to this directory and then an OSS bucket can be shared across different WordPress containers.

Log on to the Container Service console.

Click **Applications** in the left-side navigation pane.

Select the cluster used in this example from the **Cluster** list.

Click **Update** at the right of the application **wordpress** created in this example.

Container Service							Refresh	Create Application
Help: Create an application Change application configurations Simple route blue-green release policy Container auto scaling								
Cluster: test-link								
Overview								
Applications								
Services								
Clusters								
Nodes								
Networks								
Data Volumes								
Name	Description	Status	Container Status	Time Created	Time Updated	Action		
test-cluster-link		Running	Running:4 Stop:0	2018-01-22 13:22:49	2018-01-22 17:03:15	Stop	Update Redeploy	Delete Events
wordpress		Running	Running:4 Stop:0	2018-01-22 16:35:15	2018-01-22 16:35:53	Stop	Update Redeploy	Delete Events

In the **Template** field, add the mapping from OSSFS data volumes to the WordPress directory.

Note: You must modify the **Version**. Otherwise, the application cannot be redeployed.

Change Configuration

Name: wordpress

*Version: 1.1

Note: The version of the application must be changed; otherwise, the "OK" button is not available.

Description:

Use Latest Image: ☒ Force Reschedule: ☐

Release Mode: Standard Release

Template:

```
1 web:
2   image: registry.aliyuncs.com/acs-sample/wordpress:4
3   ports:
4     - '80'
5   volumes:
6     - 'wp_upload:/var/www/html/wp-content/uploads'
7   environment:
8     WORDPRESS_AUTH_KEY: changeme
9     WORDPRESS_SECURE_AUTH_KEY: changeme
10    WORDPRESS_LOGGED_IN_KEY: changeme
11    WORDPRESS_NONCE_KEY: changeme
12    WORDPRESS_AUTH_SALT: changeme
13    WORDPRESS_SECURE_AUTH_SALT: changeme
14    WORDPRESS_LOGGED_IN_SALT: changeme
15    WORDPRESS_NONCE_SALT: changeme
```

Use Existing Orchestration Template Label description

OK Cancel

Click OK to redeploy the application.

Open WordPress and upload attachments. Then, you can see the uploaded attachments in the OSS bucket.

Use Docker Compose to test cluster network connectivity

This document provides a simple Compose file used to realize one-click deployment and you can test the container network connectivity by visiting the service access endpoint.

Scenarios

When deploying interdependent applications in a Docker cluster, you must make sure that the applications can access each other to realize cross-host container network connectivity. However, sometimes containers on different hosts cannot access each other due to network problems. If this happens, it is difficult to troubleshoot the problem. Therefore, an easy-to-use Compose file can be used to test the connectivity among cross-host containers within a cluster.

Solutions

Use the provided image and orchestration template to test the connectivity among containers.

```
web:
  image: registry.aliyuncs.com/xianlu/test-link
  command: python test-link.py
  restart: always
  ports:
    - 5000
  links:
    - redis
  labels:
    aliyun.scale: '3'
    aliyun.routing.port_5000: test-link;
redis:
  image: redis
  restart: always
```

This example uses Flask to test the container connectivity.

The preceding orchestration template deploys a Web service and a Redis service. The Web service contains three Flask containers and these three containers will be evenly distributed to three nodes when started. The three containers are on different hosts and the current network can realize cross-host container connectivity if the containers can ping each other. The Redis service runs on one of the three nodes. When started, each Flask container registers to the Redis service and reports the container IP address. The Redis service has the IP addresses of all the containers in the cluster after the three Flask containers are all started. When you access any of the three Flask containers, the container will send ping command to the other two containers and you can check the network connectivity of the cluster according to the ping command response.

Procedure

Create a cluster which contains three nodes.

In this example, the cluster name is **test-link**. For how to create a cluster, see [Create a cluster](#).

Note: Select to create a Server Load Balancer instance when creating the cluster.

Cluster List									
You can create up to 5 clusters and can add up to 20 nodes in each cluster. Refresh Create Cluster									
Help: Create cluster How to add existing ECS instances Cross-zone node management Log Service integration Connect to cluster through Docker Client									
Name									
Cluster Name/ID	Cluster Type	Region	Network Type	Cluster Status	Node Status	Number of Nodes	Time Created	Docker Version	Action
test-link c21772b02a01383a...	Alibaba Cloud Cluster	China East 1 (Hangzhou)	VPC vpc-3g3wquj1sgaah4kzsm79i	Running	Healthy	3	2018-01-22 13:11:34	17.06.2-ce	Manage View Logs Delete Monitor More

Use the preceding template to create an application (in this example, the application name is **test-cluster-link**) to deploy the **web** service and **redis** service.

For how to create an application, see [Create an application](#).

On the **Application List** page, click the application name to view the created services.

Application:test-cluster-link

Refresh

Overview

Name: test-cluster-link

Time Created: 2018-01-22

Time Updated: 2018-01-22

Cluster: test-link

Trigger 1. You can only have one of each trigger type.

Create Trigger

No trigger is available at the moment. Click "Create Trigger" in the upper-right corner.

Services

Containers

Logs

Events

Routes

Name	Application	Status	Container Status	Image	Action
redis	test-cluster-link	Running	Running:1 Stop:0	redis:latest	Stop Restart Reschedule Update Delete Events
web	test-cluster-link	Running	Running:3 Stop:0	registry.aliyuncs.com/xianlu/test-link:latest	Stop Restart Reschedule Update Delete Events

Click the name of the **web** service to enter the service details page.

You can see that the three containers (**test-cluster-link_web_1**, **test-cluster-link_web_2**, **test-cluster-link_web_3**) are all started and distributed on different nodes.

Service: test-cluster-link_web

RefreshScale

Overview

Service Name: webApplication: test-cluster-linkImage: registry.aliyuncs.com/xianlu/test-link:latestNumber: 3Running

Access Endpoint: http://test-link.c21772b02a01383a...cn-hangzhou.alicontainer.com

ContainersLogsConfigurationsEvents

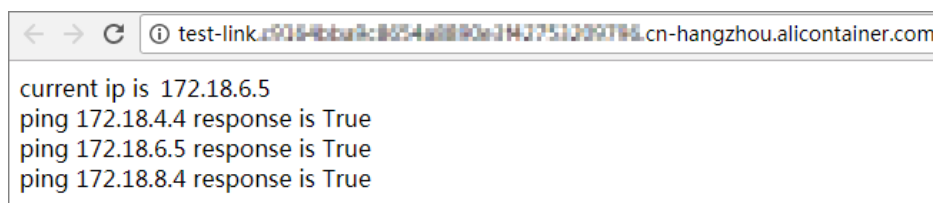
Name/ID	Status	Health Check	Image	Port	Container IP	Node IP	Action				
test-cluster-link... c21772b02a01383a...	running	Normal	registry.aliyuncs.com/sha256:f5a856388...	3306:3306:3306:3306:3306:3306:3306:3306	172.18.6.5	192.168.1.146	Delete	Stop	Monitor	Logs	Web Terminal
test-cluster-link... bc9ad2776f54f80d...	running	Normal	registry.aliyuncs.com/sha256:f5a856388...	3306:3306:3306:3306:3306:3306:3306:3306	172.18.8.4	192.168.1.147	Delete	Stop	Monitor	Logs	Web Terminal
test-cluster-link... 170a83454a0c5027...	running	Normal	registry.aliyuncs.com/sha256:f5a856388...	3306:3306:3306:3306:3306:3306:3306:3306	172.18.4.4	192.168.1.145	Delete	Stop	Monitor	Logs	Web Terminal

Visit the access endpoint of the **web** service.

As shown in the following figure, the container **test-cluster-link_web_1** can access the container **test-cluster-link_web_2** and container **test-cluster-link_web_3**.

test-link.c21772b02a01383a...cn-hangzhou.alicontainer.com									
current ip is 172.18.8.4 ping 172.18.4.4 response is True ping 172.18.6.5 response is True ping 172.18.8.4 response is True									

Refresh the page. As shown in the following figure, the container **test-cluster-link_web_2** can access the container **test-cluster-link_web_1** and container **test-cluster-link_web_3**.



As the preceding results show, the containers in the cluster can access each other.

Log

Use ELK in Container Service

Background

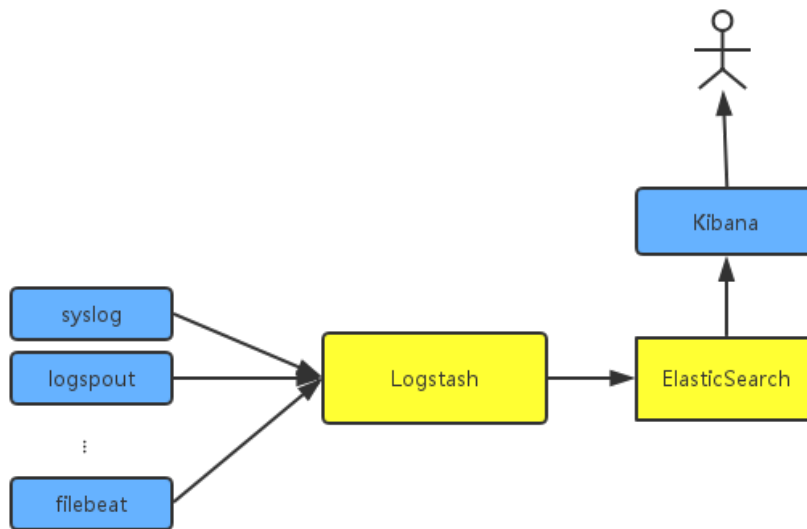
Logs are an important component of the IT system. They record system events and the time when the events occur. We can troubleshoot system faults according to the logs and make statistical analysis.

Logs are usually stored in the local log files. To view logs, log on to the machine and filter keywords by using `grep` or other tools. However, when the application is deployed on multiple machines, viewing logs in this way is inconvenient. To locate the logs for a specific error, you have to log on to all the machines and filter files one after another. That is why concentrated log storage has emerged. All the logs are collected in Log Service and you can view and search for logs in Log Service.

In the Docker environment, concentrated log storage is even more important. Compared with the traditional operation and maintenance mode, Docker usually uses the orchestration system to manage containers. The mapping between container and host is not fixed and containers might be constantly migrated between hosts. You cannot view the logs by logging on to the machine and the concentrated log becomes the only choice.

Container Service integrates with Alibaba Cloud Log Service and automatically collects container logs to Log Service by using declarations. However, some users might prefer the ELK (Elasticsearch+ Logstash+ Kibana) combination. This document introduces how to use ELK in Container Service.

Overall structure



An independent Logstash cluster needs to be deployed. Logstash is large and resource-consuming, so we do not run it on each machine, not to mention in every Docker container. To collect the container logs, syslog, Logspout, and filebeat are used. You might also use other collection methods.

To try to fit the actual scenario, two clusters are created here: one is the **testelk** cluster for deploying ELK, and the other is the **app** cluster for deploying applications.

Procedure

Note: The clusters and Server Load Balancer instance created in this document must be in the same region.

Step 1. Create a Server Load Balancer instance

To enable other services to send logs to Logstash, create and configure a Server Load Balancer instance before configuring Logstash.

1. Log on to the **Server Load Balancer** console before creating an application.
2. Create a Server Load Balancer instance whose **Instance type** is **Internet**.

Add 2 listeners for the created Server Load Balancer instance. The frontend and backend port mappings of the 2 listeners are 5000: 5000 and 5044: 5044 respectively, with no backend server added.

Add Listener

1.Listener Configuration 2.Health Check 3.Success

Front-end Protocol [Port]* TCP : 5000
Port range is 1-65535.

Backend Protocol [Port]* TCP : 5000
Port range is 1-65535.

Peak Bandwidth: No Limits [Configure](#)
Instances charged by traffic are not limited by peak bandwidth. Peak bandwidth range is 1-5000.

Scheduling Algorithm: Weighted f

Use Server Group: ☐

Automatically Enable Listener After Creation: ☒ Enable

☐ [Show Advanced Options](#)

[Next](#) [Cancel](#)

Step 2. Deploy ELK

Log on to the Container Service console.

Create a cluster named **testelk**. For how to create a cluster, see [Create a cluster](#).

Note: The cluster and the Server Load Balancer instance created in step 1 must be in the same region.

Bind the Server Load Balancer instance created in step 1 to this cluster.

On the **Cluster List** page, click **Manage** at the right of **testelk**. Click **Load Balancer Settings** in the left-side navigation pane. Click **Bind Server Load Balancer**. Select the created Server Load Balancer instance from the **Server Load Balancer ID** list and then click **OK**.

Deploy ELK by using the following orchestration template. In this example, an application named **elk** is created.

For how to create an application by using an orchestration template, see [Create an application](#).

Note: Replace `${SLB_ID}` in the orchestration file with the ID of the Server Load Balancer instance created in step 1.

```
version: '2'
services:
  elasticsearch:
    image: elasticsearch

  kibana:
    image: kibana
    environment:
      ELASTICSEARCH_URL: http://elasticsearch:9200/
    labels:
      aliyun.routing.port_5601: kibana
    links:
      - elasticsearch

  logstash:
    image: registry.cn-hangzhou.aliyuncs.com/acs-sample/logstash
    hostname: logstash
    ports:
      - 5044:5044
      - 5000:5000
    labels:
      aliyun.lb.port_5044: 'tcp://${SLB_ID}:5044' #Create a Server Load Balancer instance first.
      aliyun.lb.port_5000: 'tcp://${SLB_ID}:5000'
    links:
      - elasticsearch
```

In this orchestration file, the official images are used for Elasticsearch and Kibana, with no changes made. Logstash needs a configuration file, so make an image on your own to include the configuration file. The image source codes can be found in [demo-logstash](#).

The Logstash configuration file is as follows. This is a simple Logstash configuration. Two input formats, syslog and filebeats, are provided and their external ports are 5044 and 5000 respectively.

```
input {
  beats {
    port => 5044
    type => beats
  }

  tcp {
```

```

port => 5000
type => syslog
}

}

filter {
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
  }

  stdout { codec => rubydebug }
}

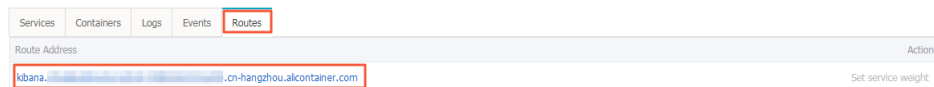
```

Configure the Kibana index.

Access Kibana.

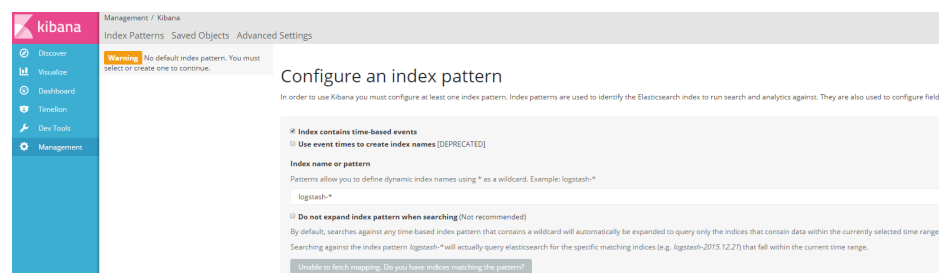
The URL can be found under the **Routes** tab of the application.

On the **Application List** page, click the application name **elk**. Click the **Routes** tab and then click the route address to access Kibana.



Create an index.

Configure the settings as per your needs and then click **Create**.



Step 3. Collect logs

In Docker, the standard logs adopt Stdout file pointer. The following example first demonstrates how to collect Stdout to ELK. If you are using file logs, you can use filebeat directly. WordPress is used for the demonstration. The following is the orchestration template of WordPress. An application **wordpress** is created in another cluster.

Log on to the Container Service console.

Create a cluster named **app**. For how to create a cluster, see [Create a cluster](#).

Note: The cluster and the Server Load Balancer instance created in step 1 must be in the same region.

Create the application **wordpress** by using the following orchestration template:

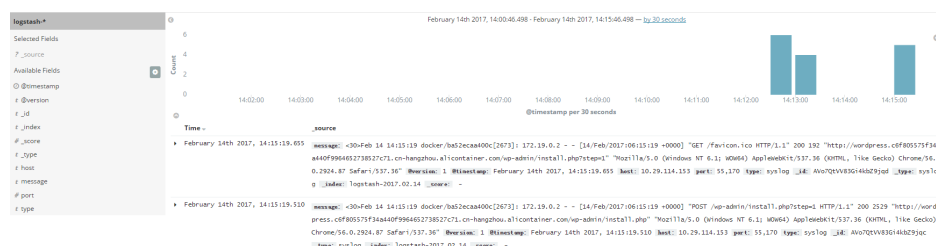
Note: Replace `${SLB_IP}` in the orchestration file with the IP address of the Server Load Balancer instance created in step 1.

```
version: '2'
services:
  mysql:
    image: mysql
    environment:
      - MYSQL_ROOT_PASSWORD=password

  wordpress:
    image: wordpress
    labels:
      aliyun.routing.port_80: wordpress
    links:
      - mysql:mysql
    environment:
      - WORDPRESS_DB_PASSWORD=password
    logging:
      driver: syslog
    options:
      syslog-address: 'tcp://${SLB_IP}:5000'
```

After the application is deployed successfully, click the application name **wordpress** on the **Application List** page. Click the **Routes** tab and then click the route address to access the WordPress application.

On the **Application List** page, click the application name **elk**. Click the **Routes** tab and then click the route address to access Kibana and view the collected logs.



A new Docker log collection scheme: log-pilot

This document introduces a new log collection tool for Docker: log-pilot. Log-pilot is a log collection image we provide for you. You can deploy a log-pilot instance on each machine to collect all the Docker application logs.

Note: Docker of Linux version is supported, while Docker of Windows or Mac version is not supported.

Log-pilot has the following features:

- A separate log process collects the logs of all the containers on the machine. No need to start a log process for each container.
- Log-pilot supports file logs and stdout logs. Docker log driver or Logspout can only process stdout, while log-pilot supports collecting the stdout logs and the file logs.
- Declarative configuration. When your container has logs to collect, log-pilot will automatically collect logs of the new container if the path of the log file to be collected is declared by using the label. No other configurations need to be changed.
- Log-pilot supports multiple log storage methods and can deliver the logs to the correct location for powerful Alibaba Cloud Log Service, popular Elasticsearch combination, or even Graylog.
- Open-source. Log-pilot is fully open-sourced. You can download the codes from [log-pilot GitHub project](#). If the current features cannot meet your requirements, welcome to raise an issue.

Quick start

See a simple scenario as follows: start a log-pilot and then start a Tomcat container, letting log-pilot collect Tomcat logs. For simplicity, here Alibaba Cloud Log Service or ELK is not involved. To run locally, you only need a machine that runs Docker.

First, start log-pilot.

Note: When log-pilot is started in this way, all the collected logs will be directly output to the console because no log storage is configured for backend use. Therefore, this method is mainly for debugging.

Open the terminal and enter the following commands:

```
docker run --rm -it \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /:/host \
--privileged \
registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.1
```

You will see the startup logs of log-pilot.

```
bash-3.2$ docker run --rm -it \
> -v /var/run/docker.sock:/var/run/docker.sock \
> -v /:/host \
> registry.cn-hangzhou.aliyuncs.com/acs-sample/fluentd-pilot:0.1
use default output
DEBI [0000] ad93dbee9691cc6a1ed1f9fab9ee365774d2f432628704256339405475de3515 has not log config, skip
WARN [0000] start fluentd
2017-02-08 14:09:24 +0000 [info]: reading config file path="/etc/fluentd/fluentd.conf"
2017-02-08 14:09:24 +0000 [info]: starting fluentd-0.12.32
2017-02-08 14:09:24 +0000 [info]: gem 'fluent-plugin-elasticsearch' version '1.9.2'
2017-02-08 14:09:24 +0000 [info]: gem 'fluentd' version '0.12.32'
2017-02-08 14:09:24 +0000 [info]: adding match pattern="*" type="stdout"
2017-02-08 14:09:24 +0000 [info]: using configuration file: <ROOT>
<match **>
  @type stdout
</match>
</ROOT>
DEBI [0108] Process container start event: f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860
INFO [0108] Logs: [[catalina /host/var/lib/docker/containers/f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860/json f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372
on log map[]] [access /host/var/lib/docker/volumes/424d96798c3255402168df54886e1855884a8140389c7578391799aff176547f/_data /usr/local/tomcat/logs/none_local/host_access_log.*.txt map[]]]
```

Do not close the terminal. Open a new terminal to start Tomcat. The Tomcat image is among the few Docker images that use stdout and file logs at the same time, and is suitable for the demonstration here.

```
docker run -it --rm -p 10080:8080 \
-v /usr/local/tomcat/logs \
--label aliyun.logs.catalina=stdout \
--label aliyun.logs.access=/usr/local/tomcat/logs/localhost_access_log.*.txt \
tomcat
```

Note:

- aliyun.logs.catalina=stdout tells log-pilot that this container wants to collect stdout logs.
- aliyun.logs.access=/usr/local/tomcat/logs/localhost_access_log.*.txt indicates to collect all log files whose names comply with the localhost_access_log.*.txt format under the /usr/local/tomcat/logs/ directory in the container. The label usage will be introduced in details later.

Note: If you deploy Tomcat locally, instead of in the Alibaba Cloud Container Service, specify -v /usr/local/tomcat/logs. Otherwise, log-pilot cannot read log files. Container Service has implemented the optimization and you do not need to specify -v on your own.

Log-pilot will monitor the events in the Docker container. When it finds any container with aliyun.logs.xxx, it will automatically parse the container configuration and start to collect the corresponding logs. After you start Tomcat, you will find many contents are output immediately by the log-pilot terminal, including the stdout logs output at the Tomcat startup, and some debugging information output by log-pilot itself.

```

2017-02-08 14:27:28 +0000 f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860.access: {"message":"192.168.2.1 - [08/Feb/2017:14:27:26 +0000] \"GET / HTTP/1.1\" 200 11250","host":"f9f2id1973e3","target":"access","docker_container":"mad_spence"}
2017-02-08 14:27:38 +0000 f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860.access: {"message":"192.168.2.1 - [08/Feb/2017:14:27:33 +0000] \"GET / HTTP/1.1\" 200 11250","host":"f9f2id1973e3","target":"access","docker_container":"mad_spence"}
2017-02-08 14:27:38 +0000 f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860.access: {"message":"192.168.2.1 - [08/Feb/2017:14:27:35 +0000] \"GET / HTTP/1.1\" 200 11250","host":"f9f2id1973e3","target":"access","docker_container":"mad_spence"}
2017-02-08 14:27:48 +0000 f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860.access: {"message":"192.168.2.1 - [08/Feb/2017:14:27:39 +0000] \"GET / HTTP/1.1\" 200 11250","host":"f9f2id1973e3","target":"access","docker_container":"mad_spence"}
2017-02-08 14:27:48 +0000 f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860.access: {"message":"192.168.2.1 - [08/Feb/2017:14:27:40 +0000] \"GET / HTTP/1.1\" 200 11250","host":"f9f2id1973e3","target":"access","docker_container":"mad_spence"}
2017-02-08 14:27:48 +0000 f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860.access: {"message":"192.168.2.1 - [08/Feb/2017:14:27:40 +0000] \"GET / HTTP/1.1\" 200 11250","host":"f9f2id1973e3","target":"access","docker_container":"mad_spence"}
2017-02-08 14:27:48 +0000 f04e7fc992e5c17d5c18086831b2ce1a9af8815870d09b02833b372c1bf27860.access: {"message":"192.168.2.1 - [08/Feb/2017:14:27:41 +0000] \"GET / HTTP/1.1\" 200 11250","host":"f9f2id1973e3","target":"access","docker_container":"mad_spence"}

```

You can access the deployed Tomcat in the browser, and find that similar records are displayed on the log-pilot terminal every time you refresh the browser. The contents after message are the logs collected from `/usr/local/tomcat/logs/localhost_access_log.XXX.txt`.

Use Elasticsearch + Kibana

Deploy Elasticsearch + Kibana. See [Use ELK in Container Service](#) to deploy ELK in Alibaba Cloud Container Service, or deploy them directly on your machine by following the [ElasticSearch/Kibana](#) documents. This document assumes that you have deployed the two components.

If you are still running the log-pilot, close it first, and then start it again by using the following commands:

Note: Before running the following commands, replace the two variables `ELASTICSEARCH_HOST` and `ELASTICSEARCH_PORT` with the actual values you are using. `ELASTICSEARCH_PORT` is generally 9200.

```

docker run --rm -it \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /:/host \
--privileged \
-e FLUENTD_OUTPUT=elasticsearch \
-e ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST} \
-e ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT} \
registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.1

```

Compared with the previous log-pilot startup method, here three environment variables are added:

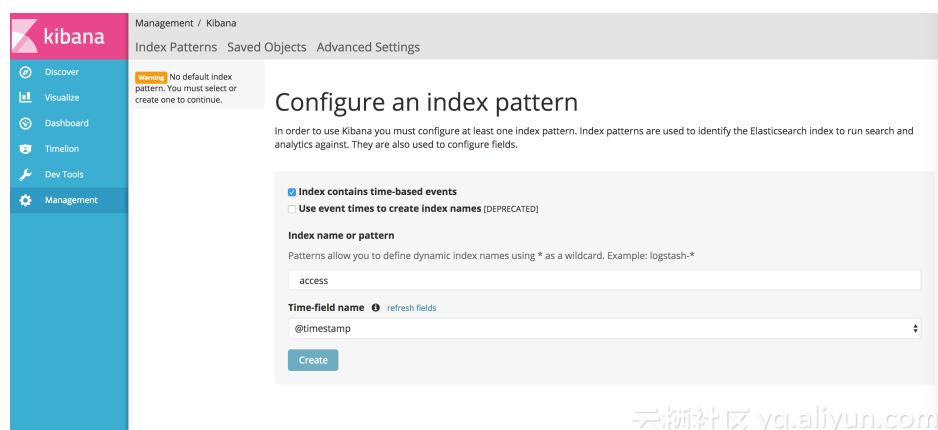
- `FLUENTD_OUTPUT=elasticsearch`: Send the logs to Elasticsearch.
- `ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST}`: The domain name of Elasticsearch.
- `ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT}`: The port number of Elasticsearch.

Continue to run the Tomcat started previously, and access it again to make Tomcat generate some logs. All these newly generated logs will be sent to Elasticsearch.

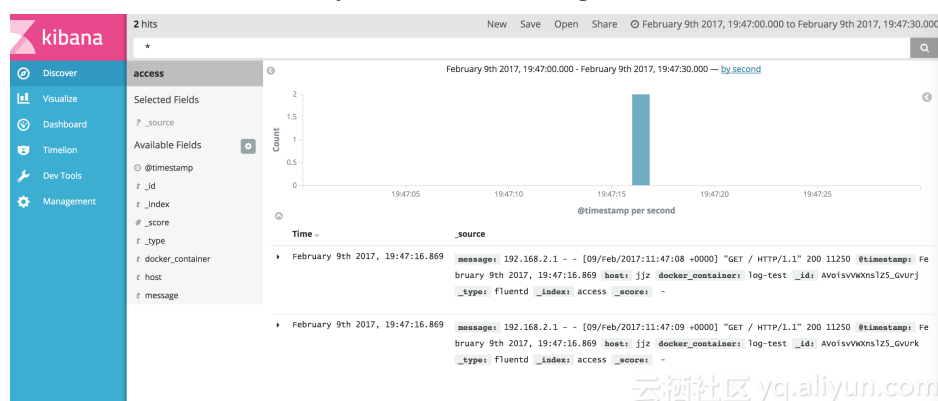
Open Kibana, and no new logs are visible yet. Create an index first. Log-pilot will write logs to the specific index of Elasticsearch. The rules are as follows:

If label `aliyun.logs.tags` is used in the application, and `tags` contains `target`, use `target` as the index of Elasticsearch. Otherwise, use `XXX` in the label `aliyun.logs.XXX` as the index.

In the previous example about Tomcat, the label `aliyun.logs.tags` is not used, so `access` and `catalina` are used by default as the index. First create the index access.



After the index is created, you can view the logs.



Use log-pilot in Alibaba Cloud Container Service

Container Service makes some special optimization for log-pilot, which adapts to running log-pilot best.

To run log-pilot in Container Service, create an application by using the following orchestration file. For how to create an application, see [Create an application](#).

```
pilot:
image: registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.1
volumes:
- /var/run/docker.sock:/var/run/docker.sock
- /:/host
privileged: true
environment:
FLUENTD_OUTPUT: elasticsearch #Replace based on your requirements
ELASTICSEARCH_HOST: ${elasticsearch} #Replace based on your requirements
ELASTICSEARCH_PORT: 9200
labels:
aliyun.global: true
```

Then, you can use the `aliyun.logs.xxx` label on the application that you want to collect logs.

Label description

When Tomcat is started, the following two labels are declared to tell log-pilot the location of the container logs.

```
--label aliyun.logs.catalina=stdout
--label aliyun.logs.access=/usr/local/tomcat/logs/localhost_access_log*.txt
```

You can also add more labels on the application container.

`aliyun.logs.$name = $path`

- The variable name is the log name and can only contain 0–9, a–z, A–Z, and hyphens (-).
- The variable path is the path of the logs to be collected. The path must specify the file, and cannot only be a directory. Wildcards are supported as part of the file name, for example, `/var/log/he.log` and `/var/log/*.log` are both correct. However, `/var/log` is not valid because the path cannot be only a directory. `stdout` is a special value, indicating standard output.

`aliyun.logs.$name.format`: The log format. Currently, the following formats are supported.

- none: Unformatted plain text.
- json: JSON format. One complete JSON string in each line.
- csv: CSV format.

`aliyun.logs.$name.tags`: The additional field added when the logs are reported. The format is `k1=v1,k2=v2`. The key-value pairs are separated by commas, for example, `aliyun.logs.access.tags="name=hello,stage=test"`. Then, the logs reported to the storage will contain the name field and the stage field.

If ElasticSearch is used for log storage, the target tag will have a special meaning, indicating the corresponding index in ElasticSearch.

Log-pilot extension

For most users, the existing features of log-pilot can meet their requirements. If log-pilot cannot meet your requirements, you can:

- Submit an issue at <https://github.com/AliyunContainerService/log-pilot>.
- Directly change the codes and then raise the PR.

Health check mechanism of Docker containers

In a distributed system, the service availability needs to be frequently checked by using the health check mechanism to avoid exceptions when being called by other services. Docker introduced native health check implementation after version 1.12. This document introduces the health check mechanism of Docker containers and the new features in Docker swarm mode.

Process-level health check checks whether or not the process is alive and is the simplest health check for containers. Docker daemon automatically monitors the PID1 process in the container. If the docker run command specifies the restart policy, closed containers can be restarted automatically according to the restart policy. In practical use, process-level health check alone is far from enough. For example, if a container process is still alive, but cannot respond to user requests because of application deadlock, such problems cannot be discovered by process monitoring.

Kubernetes provides Liveness and Readiness probes to check the health status of the container and its service respectively. Alibaba Cloud Container Service also provides a similar Service health check mechanism.

Docker native health check capability

Docker introduced the native health check implementation after version 1.12. The health check configurations of an application can be declared in the Dockerfile. The HEALTHCHECK instruction declares the health check command that can be used to determine whether the service status of the container master process is normal. This can reflect the real status of the container.

HEALTHCHECK instruction format:

- HEALTHCHECK [option] CMD <command>: The command that sets the container health check.
- HEALTHCHECK NONE: If the basic image has a health check instruction, this line can be used to block it.

Note: The HEALTHCHECK can only appear once in the Dockerfile. If multiple HEALTHCHECK instructions exist, only the last one takes effect.

Images built by using Dockerfiles that contain HEALTHCHECK instructions can check the health status when instantiating Docker containers. Health check is started automatically after the container is started.

HEALTHCHECK supports the following options:

- `--interval=<interval>`: The time interval between two health checks. The default value is 30 seconds.
- `--timeout=<interval>`: The timeout time for running the health check command. The health check fails if it lasts longer than this time period. The default value is 30 seconds.
- `--retries=<number of times>`: When the health check fails continuously for a specified number of times, the container status is regarded as unhealthy. The default value is 3.
- `--start-period=<interval>`: The initialization time of application startup. Failed health check during the startup is not counted. The default value is 0 second (introduced from version 17.05).

The command after HEALTHCHECK [option] CMD follows the same format as ENTRYPOINT, in either the shell or the exec format. The returned value of the command determines the success or failure of the health check:

- 0: Success.
- 1: Failure.
- 2: Reserved value. Do not use.

After a container is started, the initial status is **Starting**. Docker Engine waits for a period of interval to regularly run the health check command. If the returned value of a single check is not 0 or the running lasts longer than the specified timeout time, the health check is considered as failed. If the health check fails continuously for retries times, the health status changes to **Unhealthy**.

- If the health check succeeds once, Docker will change the container status back to **Healthy**.
- When the container health status changes, Docker Engine issues a `health_status` event.

Assume that an image is a simple Web service. To enable health check to determine whether its Web service is working normally or not, curl can be used to help with the determination and the HEALTHCHECK instruction in its Dockerfile can be written as follows:

```
FROM elasticsearch:5.5
HEALTHCHECK --interval=5s --timeout=2s --retries=12 \
CMD curl --silent --fail localhost:9200/_cluster/health || exit 1
```

```
docker build -t test/elasticsearch:5.5 .
docker run --rm -d \
--name=elasticsearch \
test/elasticsearch:5.5
```

You can use `docker ps`. After several seconds, the Elasticsearch container changes from the **Starting** status to **Healthy** status.

```
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
```



```
c9a6e68d4a7f test/elasticsearch:5.5 "/docker-entypoin..." 2 seconds ago Up 2 seconds (health: starting) 9200/tcp,
9300/tcp elasticsearch
$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
c9a6e68d4a7f test/elasticsearch:5.5 "/docker-entypoin..." 14 seconds ago Up 13 seconds (healthy) 9200/tcp,
9300/tcp elasticsearch
```

Another method is to directly specify the health check policy in the docker run command.

```
$ docker run --rm -d \
--name=elasticsearch \
--health-cmd="curl --silent --fail localhost:9200/_cluster/health || exit 1" \
--health-interval=5s \
--health-retries=12 \
--health-timeout=2s \
elasticsearch:5.5
```

To help troubleshoot the issue, all output results of health check commands (including stdout and stderr) are stored in health status and you can view them with the docker inspect command. Use the following commands to retrieve the health check results of the past five containers.

```
docker inspect --format='{{json .State.Health}}' elasticsearch
```

Or

```
docker inspect elasticsearch | jq ".[].State.Health"
```

The sample result is as follows:

```
{
  "Status": "healthy",
  "FailingStreak": 0,
  "Log": [
    {
      "Start": "2017-08-19T09:12:53.393598805Z",
      "End": "2017-08-19T09:12:53.452931792Z",
      "ExitCode": 0,
      "Output": "..."
    },
    ...
  ]
}
```

We usually recommend that you declare the corresponding health check policy in the Dockerfile to facilitate the use of images because application developers know better about the application SLA. The application deployment and Operation & Maintenance personnel can adjust the health check policies as needed for deployment scenarios by using the command line parameters and REST API.

The Docker community provides some instance images that contain health checks. Obtain them in

the following project: <https://github.com/docker-library/healthchec>.

Note:

- Alibaba Cloud Container Service supports Docker native health check mechanism and Alibaba Cloud extension health check mechanism.
- Currently, Kubernetes does not support Docker native health check mechanism.

Health check capability for Docker swarm mode services

After Docker 1.13, health check policies are supported in the Docker swarm mode.

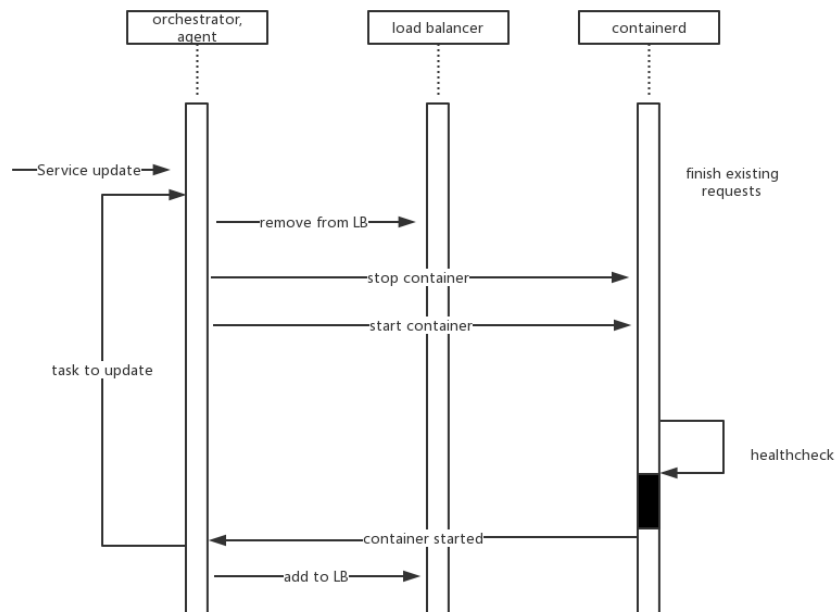
You can specify the health check policy in the docker service create command:

```
$ docker service create -d \  
--name=elasticsearch \  
--health-cmd="curl --silent --fail localhost:9200/_cluster/health || exit 1" \  
--health-interval=5s \  
--health-retries=12 \  
--health-timeout=2s \  
elasticsearch
```

In swarm mode, Swarm manager monitors the health status of service tasks. When a container enters the **Unhealthy** status, Swarm manager stops the container and starts a new container to replace the unhealthy one. The backend or DNS records of the Server Load Balancer (routing mesh) are automatically updated during this process to guarantee the service availability.

After version 1.13, health checks are supported in the service updating phase. In this way, the Server Load Balancer/DNS resolution do not send requests to a new container before it is fully started and enters the **Healthy** status, which makes sure that the requests will not be interrupted when applications are being updated.

The following is a sequence chart of the service updating process.



In a corporate production environment, reasonable health check settings can guarantee the application availability. Currently, many application frameworks are already built with monitoring and checking capabilities, such as Spring Boot Actuator. Integrated with the Docker built-in health check mechanism, you can implement application availability monitoring, automatic fault handling, and zero downtime updating in a concise manner.

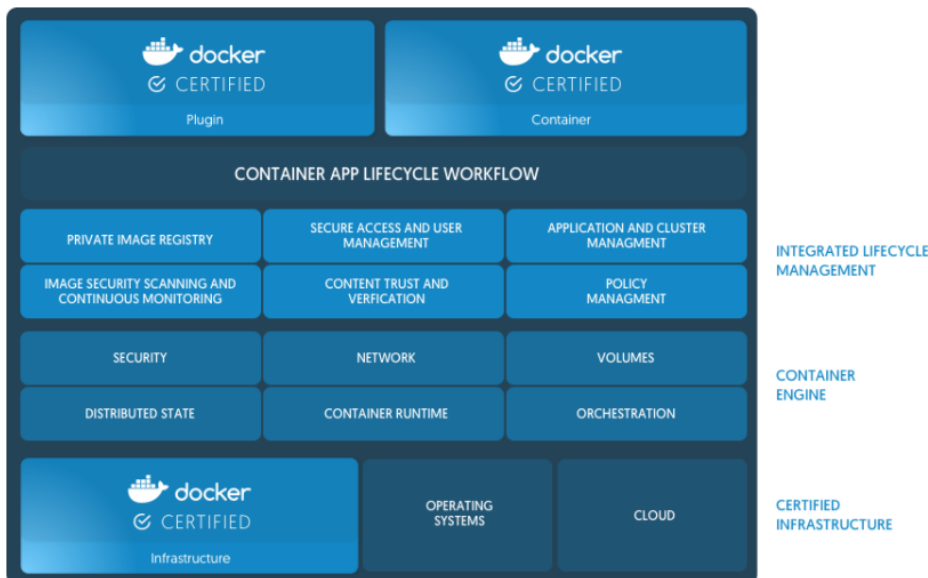
One-click deployment of Docker Datacenter

About DDC

Docker Datacenter (DDC) is an enterprise-level container management and service deployment package solution platform released by Docker. DDC is composed of the following three components:

- Docker Universal Control Plane (Docker UCP): A set of graphical management interfaces.
- Docker Trusted Registry (DTR): A trusted Docker image repository.
- Docker Engine enterprise edition: The Docker Engine providing technical support.

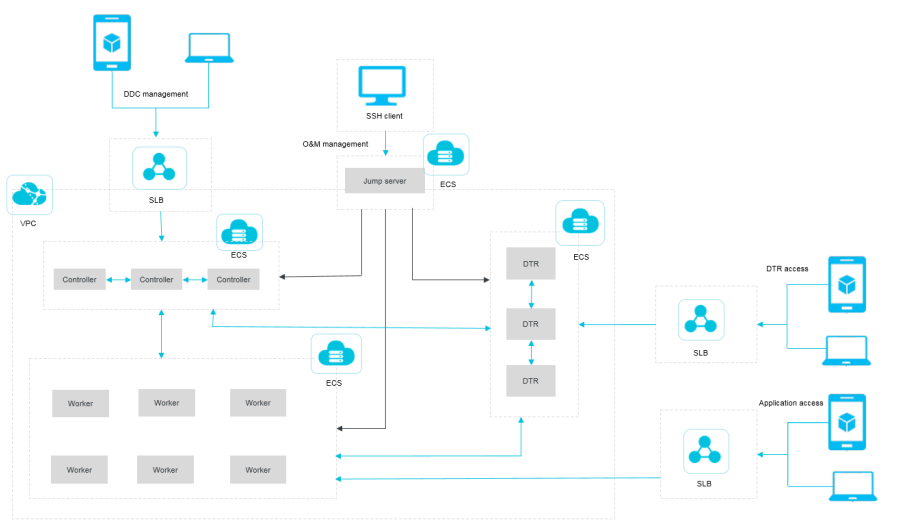
DDC is available on the [Docker official website](#).



DDC is a counterpart of Docker Cloud, another online product of the Docker company. However, DDC primarily targets enterprise users for internal deployment. You can register your own Docker image to DTR and use UCP to manage the entire Docker cluster. Both components provide web interfaces.

You must purchase a license to use DDC, but the Docker company provides a free license for a one-month trial. You can download the trial license from the Docker official website after signing up.

DDC deployment architecture



In the preceding basic architecture figure, Controller primarily runs the UCP component, DTR runs the DTR component, and Worker primarily runs your own Docker service. The entire DDC environment is deployed on the Virtual Private Cloud (VPC) and all Elastic Compute Service (ECS) instances are in the same security group. Every component provides a Server Load Balancer instance for extranet access. Operations and maintenance are implemented by using the jump server. To enhance the availability, the entire DDC environment is deployed for high availability, meaning at least two Controllers and two DTRs exist.

One-click deployment of DDC

You can use Alibaba Cloud Resource Orchestration Service (ROS) to deploy DDC in one click at the following link.

One-click deployment of DDC

In the preceding orchestration template, DDC is deployed in the region China North 2 (Beijing) by default. To change the region for deployment, click **Back** in the lower-right corner of the page. Select your region and then click **Next**.

Complete the configurations. Click **Create** to deploy a set of DDC.

The screenshot shows the 'Activate stack' step in the ROS console. The progress bar at the top indicates 'Enter directly' (blue), 'Activate stack' (active, blue), and 'Created successfully' (grey). The configuration details are as follows:

- Selected Region :** China North 2 (Beijing)
- * Stack Name :** (empty field with a tooltip: "The name must be 1-64 characters long and start with an uppercase or lowercase letter. It can contain numbers, '-' and '.'. The stack name must be unique and cannot be modified after creation.")
- * Creation timeout (minutes) :** 60 (with a tooltip: "A positive integer within 10-180 in minutes")
- Roll back :** ☒
- DTRInstanceType :** ecs.n4.large
- ControllerSlaveMaxAmount :** 0
- ControllerSystemDiskCategory :** cloud_ssd
- ControllerInstanceType :** ecs.n4.large
- WorkerSystemDiskCategory :** cloud_ssd
- DTRSystemDiskCategory :** cloud_ssd
- WorkerMaxAmount :** 1
- ControllerImageId :** ubuntu_14_0405_64_40G_alibase_20170525.vhd

DDC access

After creating DDC successfully by using ROS, you can enter the ROS stack management page by clicking **Stack Management** in the left-side navigation pane. Find the created stack, and then click the stack name or **Manage** at the right of the stack. The **Stack Overview** page appears.

The screenshot shows the 'Stack Management' page in the ROS console. The left navigation pane has 'Stack Management' highlighted. The main area shows a list of stacks with the following columns: Name, Status (All), Timeout (minutes), Roll back, Status Description, Time Created, and Operation.

Name	Status (All)	Timeout (minutes)	Roll back	Status Description	Time Created	Operation
test	Creation complete	60	Yes	Stack CREATE completed successfully	2017-11-21 17:08:40	Manage Delete More

At the bottom, it shows 'Total: 1 item(s) . Per Page: 10 item(s)' and pagination controls.

You can view the addresses used to log on to UCP and DTR in the **Output** section.

Enter the UCP address in the browser and the UCP access page appears. Enter the administrator account and password created when installing UCP and the system prompts you to import the license file. Import the license file and then enter the UCP control interface.



Deploy Container Service clusters by using Terraform

This document introduces how to use Terraform to deploy an Alibaba Cloud Container Service cluster in the Virtual Private Cloud (VPC) environment and deploy a sample WordPress application in the deployed cluster. In this document, a solution used to build Alibaba Cloud infrastructures is provided for you to use codes to automatically create, orchestrate, and manage services in Container Service.

Prerequisites

You must activate Alibaba Cloud Container Service and create an AccessKey for your account. Keep your AccessKey ID and AccessKey Secret properly.

Step 1 Install Terraform

Download Terraform

Download Terraform from the [official website](#). Select the corresponding version and platform. In this document, install the Terraform on Linux (the procedure is similar to that of installing the Terraform on Mac OS X).

Under Linux, click to download the `terraform_0.11.3_linux_amd64.zip` file.

Copy the .zip file to an appropriate path (/usr/local/terraform in this example).

Extract the .zip file and then get a binary file terraform.

Create the following entries in the `/etc/profile` directory and add the path where the binary file resides (`/usr/local/terraform` in this example) to the `PATH` environment variable.

```
export TERRAFORM_HOME=/usr/local/terraform
export PATH=$PATH:$TERRAFORM_HOME
```

Install Alibaba Cloud Terraform package

In this example, Alibaba Cloud provides its own Terraform package. You can download the package [here](#).

Download the latest version for your system. In this example, download the package for Linux.

Copy the downloaded file `terraform-provider-alicloud_linux-amd64.tgz` to the `/usr/local/terraform` folder.

Extract the downloaded file and then get a `bin` folder in which the `terraform-provider-alicloud` file resides.

Create a `.terraformrc` file in the `/usr/local/terraform` directory.

```
$ vim .terraformrc
```

Add the following contents in the file:

```
providers {
  alicloud = "/usr/local/terraform/bin/terraform-provider-alicloud"
}
```

Run the following command to test the working of Terraform. If Terraform is successfully installed, the following contents are displayed:

```
$ terraform
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
....
```

All other commands:
debug Debug output management (experimental)
force-unlock Manually unlock the terraform state
state Advanced state management

Step 2 Download Container Service Terraform scripts

You can download the script file `acs_terraform.zip` [here](#). The `.zip` file defines the resource files related to creating swarm clusters so that you can create a sample cluster quickly.

The `.zip` file contains the following files after being extracted:

main.tf

The main file of Terraform, which defines the resources to be deployed.

Region

Defines the region where resources will be created.

```
provider "alicloud" {  
  region = "cn-hongkong"  
}
```

VPC

```
resource "alicloud_vpc" "vpc" {  
  name = "${var.vpc_name}"  
  cidr_block = "${var.vpc_cidr}"  
}
```

VSwitch

```
resource "alicloud_vswitch" "vswitch" {  
  availability_zone = "${data.alicloud_zones.default.zones.0.id}"  
  name = "${var.vswitch_name}"  
  cidr_block = "${var.vswitch_cidr}"  
  vpc_id = "${alicloud_vpc.vpc.id}"  
}
```

Security group


```
resource "alicloud_security_group" "group" {
  name = "${var.sg_name}"
  vpc_id = "${alicloud_vpc.vpc.id}"
}
```

Container Service cluster

```
resource "alicloud_container_cluster" "wp_cs" {
  password = "${var.ecs_password}"
  instance_type = "${data.alicloud_instance_types.default.instance_types.0.id}"
  name = "${var.cluster_name}"
  size = "${var.node_number}"
  disk_category = "${var.disk_category}"
  disk_size = "${var.disk_size}"
  cidr_block = "${var.contaner_cidr}"
  image_id = "${data.alicloud_images.main.images.0.id}"
  vswitch_id = "${alicloud_vswitch.vswitch.id}"
}
```

RDS instance

```
resource "alicloud_db_instance" "instance" {
  engine = "${var.engine}"
  engine_version = "${var.engine_version}"
  instance_type = "${var.instance_class}"
  instance_storage = "${var.storage}"
  vswitch_id = "${alicloud_vswitch.vswitch.id}"
  security_ips = ["${var.contaner_cidr}", "${var.vswitch_cidr}"]
}
```

Database account

```
resource "alicloud_db_account" "account" {
  instance_id = "${alicloud_db_instance.instance.id}"
  name = "wp_admin"
  password = "${var.password}"
}
```

Database backup policy

```
resource "alicloud_db_backup_policy" "backup" {
  instance_id = "${alicloud_db_instance.instance.id}"
  backup_period = ["Tuesday", "Wednesday"]
  backup_time = "10:00Z-11:00Z"
}
```

Database

```
resource "alicloud_db_database" "db" {
  instance_id = "${alicloud_db_instance.instance.id}"
  name = "${var.database_name}"
}
```

Database privilege (read/write)

```
resource "alicloud_db_account_privilege" "privilege" {
  instance_id = "${alicloud_db_instance.instance.id}"
  account_name = "${alicloud_db_account.account.name}"
  privilege = "${var.db_privilege}"
  db_names = ["${alicloud_db_database.db.name}"]
}
```

Database connection string

```
resource "alicloud_db_connection" "connection" {
  instance_id = "${alicloud_db_instance.instance.id}"
}
```

outputs.tf

This file defines the output parameters. Resources created as part of the execution will generate these output parameters. This is similar to the output parameters specified in a Resource Orchestration Service (ROS) template. For example, the template will deploy an RDS instance. The following output parameter provides the value of the connection string that will be required to connect to the database.

```
output "rds_conn" {
  value = "${alicloud_db_instance.instance.connection_string}"
}
```

variables.tf

This file contains the variables that can be passed to main.tf and helps you customize the environment.

```
variable "vpc_name" {
  description = "The vpc name used to launch a new vpc."
  default = "WP-VPC"
}
```

```
variable "vpc_cidr" {  
  description = "The cidr block used to launch a new vpc."  
  default = "172.16.0.0/12"  
}  
  
variable "database_name" {  
  default = "wp_db"  
}
```

Step 3 Run Terraform scripts

To run scripts, you must first go to the directory where the `acs_terraform.zip` file is extracted, that is, `/usr/local/terraform`. You can use the following Terraform commands to run scripts and create Container Service clusters. For more information on the command usage, see [Terraform Commands \(CLI\)](#).

Run `terraform init` to initialize the environment.

```
$ terraform init  
Initializing provider plugins...  
...  
- Checking for available provider plugins on https://releases.hashicorp.com...  
- Downloading plugin for provider "alicloud" (1.7.2)...  
* provider.alicloud: version = "~> 1.7"  
Terraform has been successfully initialized!  
...
```

Run `terraform providers` to list the installed providers.

```
$ terraform providers  
.  
└─ provider.alicloud
```

Before running `terraform plan`, you must first pass the AccessKey ID and AccessKey Secret for authorization.

```
$ export ALICLOUD_ACCESS_KEY="AccessKey ID"  
$ export ALICLOUD_SECRET_KEY="AccessKey Secret"
```

Run `terraform plan` to create an execution plan and help you understand the resources that will be created or changed.

```
$ terraform plan  
Refreshing Terraform state in-memory prior to plan...  
The refreshed state will be used to calculate this plan, but will not be  
persisted to local or remote state storage.
```

```
data.alicloud_images.main: Refreshing state...
data.alicloud_instance_types.default: Refreshing state...
data.alicloud_zones.default: Refreshing state...
```

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:
...
Plan: 9 to add, 0 to change, 0 to destroy.

Note: You didn't specify an "-out" parameter to save this plan, so Terraform can't guarantee that exactly these actions will be performed if "terraform apply" is subsequently run.

After confirming the resources are created or updated as expected, run the terraform apply command to start the execution of the Terraform module.

```
$ terraform apply
```

```
data.alicloud_instance_types.default: Refreshing state...
data.alicloud_images.main: Refreshing state...
data.alicloud_zones.default: Refreshing state...
```

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:
...
Plan: 9 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

```
alicloud_vpc.vpc: Creating...
...
```

Apply complete! Resources: 9 added, 0 changed, 0 destroyed.

Outputs: ##Note

```
cs_cluster = cf6e1c0cfba4248808dfcd80ab09ddb6a
rds_conn = rm-bp1grduxk9z36bq58.mysql.rds.aliyuncs.com
rds_conn_port = 3306
```

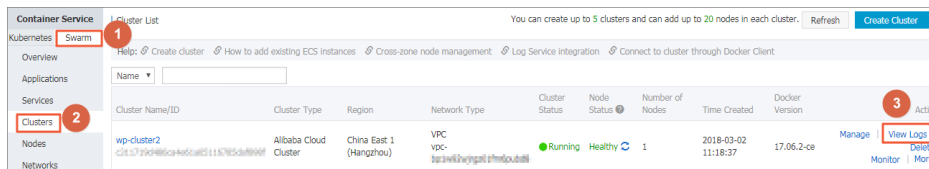
```
rds_db = wp_db
rds_db_account_name = wp_admin
```

After running the terraform apply command, the output parameters requested in the outputs.tf are displayed. In the following example, the output parameters are cs_cluster cluster name, connection string, rds_conn_port, rds_db name, and rds_db_account_name.

The output values can be listed at any time by running the terraform output command to help you configure the WordPress application.

```
$ terraform output
cs_cluster = cf6e1c0cfba4248808dfcd80ab09ddb6a
rds_conn = rm-bp1grduxk9z36bq58.mysql.rds.aliyuncs.com
rds_conn_port = 3306
rds_db = wp_db
rds_db_account_name = wp_admin
```

You can view the cluster created by using Terraform in the Container Service console. View the cluster information, node information, logs, and container information.



Step 4 Deploy a WordPress application in Container Service

Log on to the Container Service console.

Click **Applications** in the left-side navigation pane under **Swarm**. Select the cluster (**wp-cluster** in this example) created in the preceding step from the **Cluster** list. Then, click **Create Application** in the upper-right corner.

Select **wordpress**, the Docker official image, as the **Image Name**. Then, configure the **Web Routing** for this application and click **Create**. For more information, see [Simple routing - supports HTTP and HTTPS](#).

The WordPress application is displayed on the **Application List** page after being successfully created.

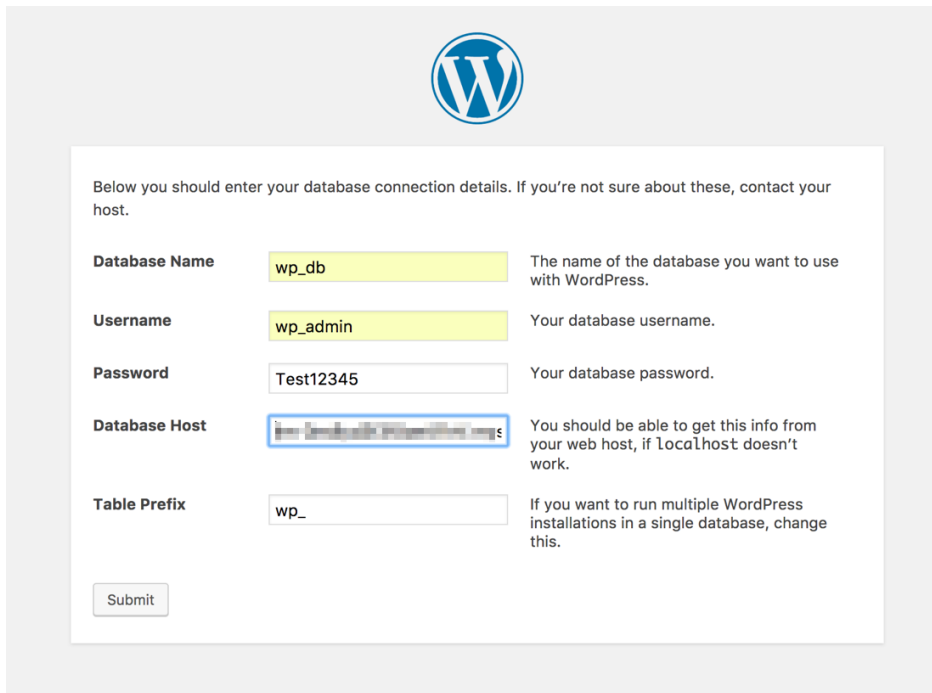
Name	Description	Status	Container Status	Time Created	Time Updated	Action
wordpress		Running	Running/0 Stop:0	2018-03-02 11:51:15	2018-03-02 11:51:15	Stop Update Delete Redeploy Events

Click the application name, and then click the **Routes** tab to obtain the route address.

Route Address	Action
wordpress.c311719a486c4e61a05116785d49981.cn-hangzhou.alicontainer.com	Set service weight

Access the WordPress welcome page. Select the language and then click **Continue**.

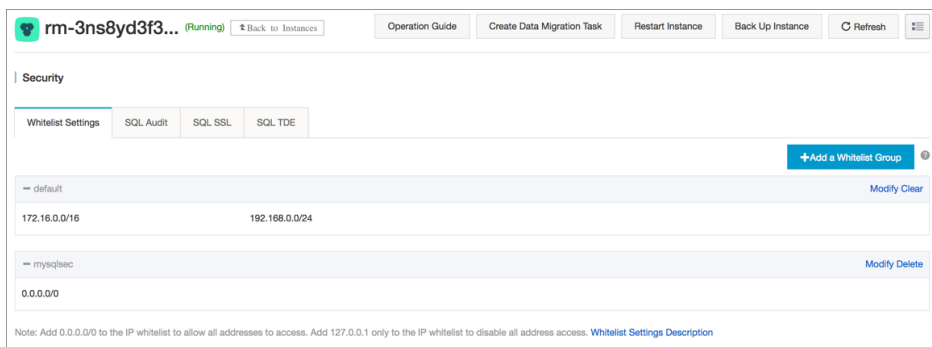
Enter the following detailed information according to the Terraform output parameters and then click **Submit**.



Below you should enter your database connection details. If you're not sure about these, contact your host.

Database Name	wp_db	The name of the database you want to use with WordPress.
Username	wp_admin	Your database username.
Password	Test12345	Your database password.
Database Host	localhost	You should be able to get this info from your web host, if localhost doesn't work.
Table Prefix	wp_	If you want to run multiple WordPress installations in a single database, change this.

After the submission, if the entered values are correct, you can go to the next configuration step to run the installation. When encountering an error, you must access the RDS database by setting the whitelist.



rm-3ns8yd3f3... (Running) | Back to Instances | Operation Guide | Create Data Migration Task | Restart Instance | Back Up Instance | Refresh |

Security

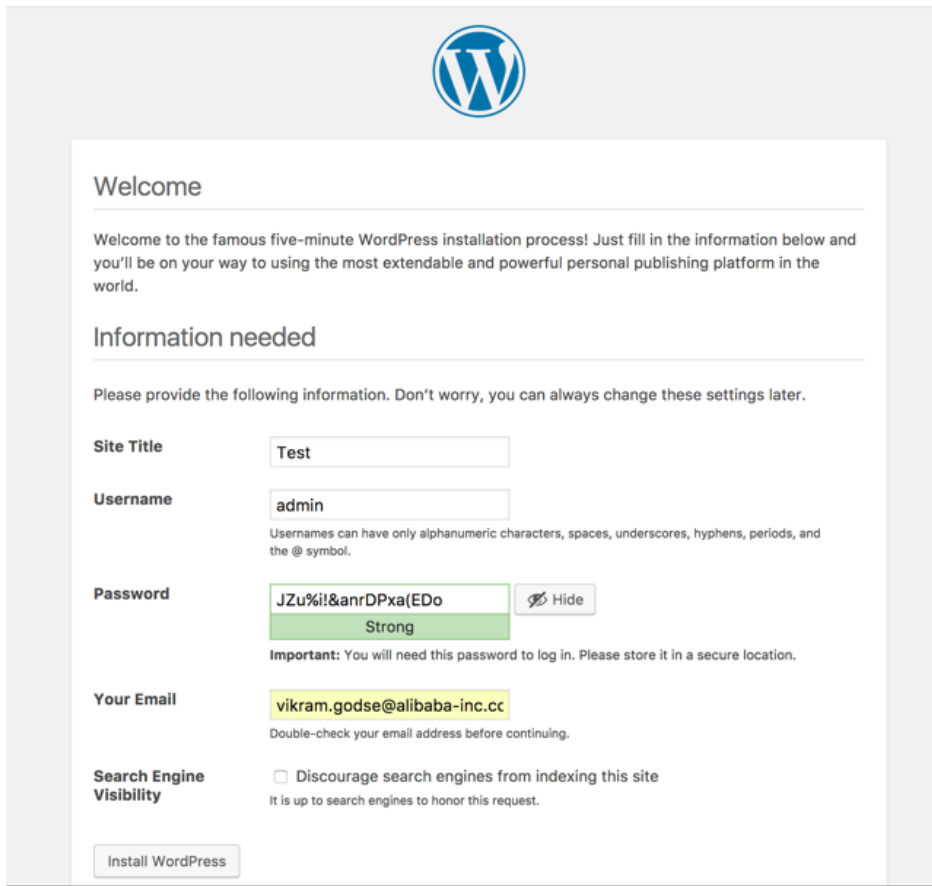
Whitelist Settings | SQL Audit | SQL SSL | SQL TDE

[+Add a Whitelist Group](#)

default	Modify	Clear
172.16.0.0/16	192.168.0.0/24	
mysqlsec	Modify	Delete
0.0.0.0/0		

Note: Add 0.0.0.0/0 to the IP whitelist to allow all addresses to access. Add 127.0.0.1 only to the IP whitelist to disable all address access. [Whitelist Settings Description](#)

Enter the **Site Title**, and the username and password of the administrator. Click **Install WordPress**.





Welcome

Welcome to the famous five-minute WordPress installation process! Just fill in the information below and you'll be on your way to using the most extendable and powerful personal publishing platform in the world.

Information needed

Please provide the following information. Don't worry, you can always change these settings later.

Site Title

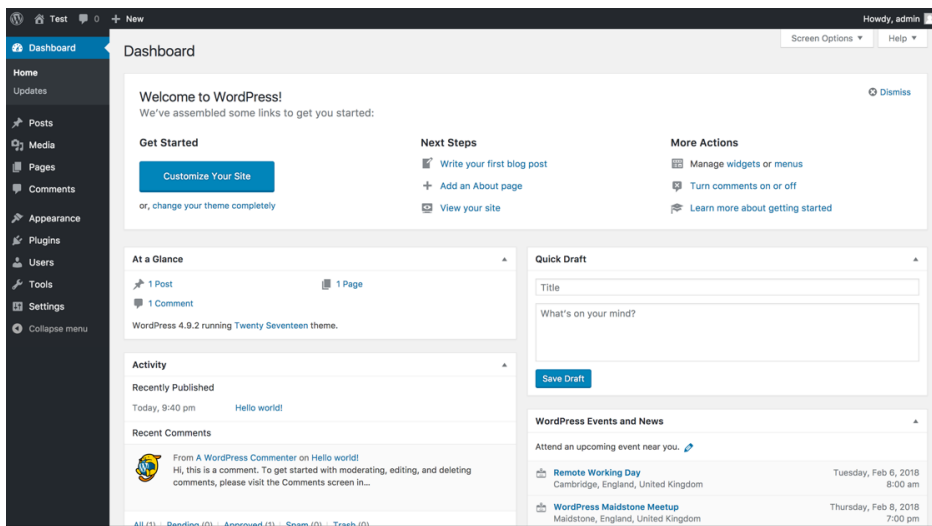
Username
Usernames can have only alphanumeric characters, spaces, underscores, hyphens, periods, and the @ symbol.

Password
Strong
Important: You will need this password to log in. Please store it in a secure location.

Your Email
Double-check your email address before continuing.

Search Engine Visibility ☐ Discourage search engines from indexing this site
It is up to search engines to honor this request.

After the installation, click **Log In**. Enter the username and password of the administrator, and then click **Log In** on the WordPress logon page to log on to the WordPress application.



The screenshot shows the WordPress dashboard for a user named 'Howdy, admin'. The left sidebar contains navigation links: Dashboard, Home, Updates, Posts, Media, Pages, Comments, Appearance, Plugins, Users, Tools, Settings, and Collapse menu. The main content area is titled 'Dashboard' and includes a 'Welcome to WordPress!' message. Below this, there are sections for 'Get Started' (with a 'Customize Your Site' button), 'Next Steps' (with links to write a post, add an about page, and view the site), and 'More Actions' (with links to manage widgets, turn comments on/off, and learn more). The 'At a Glance' section shows 1 Post, 1 Page, and 1 Comment. The 'Activity' section shows a recently published post 'Hello world!' and a recent comment. The 'Quick Draft' section has fields for Title and content, with a 'Save Draft' button. The 'WordPress Events and News' section lists upcoming events like 'Remote Working Day' and 'WordPress Maidstone Meetup'.

Further reading

Currently, Alibaba Cloud is the official major cloud provider of Terraform. To use Terraform to flexibly

build Alibaba Cloud infrastructures, see [Alibaba Cloud Provider](#) for more information and customize the resource description files to quickly build your cloud infrastructures.