Container Service

Best Practices

Best Practices

In a distributed system, we often need to check service availability by using health checks to prevent exceptions during other service calls. Docker introduced native health check implementation after version 1.12. This article introduces the health check mechanism of Docker containers and new features in Docker Swarm mode.

Process-level health checks check whether or not the process is alive and are the most simple health checks for containers. Docker daemon will automatically monitor the PID1 process in the container. If the docker run command specifies the restart policy, closed containers can be restarted automatically. In practical use, process-level health checks alone are often far from enough. For example, if a container process is still alive, but is locked by an app deadlock and fails to respond to user requests, such problems will not be discovered by process monitoring.

Kubernetes provides Liveness and Readness probes to check the container and its service health 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 app can be declared in the Dockerfile. The HEALTHCHECK command declares the health check command that can be used to determine whether the status of the container master process service is normal. This can reflect the real status of the container.

HEALTHCHECK command format are as follows:

- 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 are present, only the last one takes effect.

Images built by using Dockerfiles that contain HEALTHCHECK instructions are capable of checking health 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 value is 30 seconds

by default.
- --timeout=<interval>: The timeout time of the health check command. If a health check lasts longer than this time period, the health check is determined to have failed. The default value is 30 seconds.
- --retries=<number of retries>: When the health check fails for a specified number of times, the container will be regarded as unhealthy. The default value is 3.
- --start-period=<interval>: The initialization time of app startup. Failed health check during the startup period is not counted. The default value is 0 second (inherited 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 tells 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 will wait for a period of interval to start regularly running the health check command. If the returned value of a single check is not "0" or the running takes longer than the specified timeout value, the health check is deemed as failed. If the health check fails for consecutive retries times, the status turns to **Unhealthy**.

- Once one health check succeeds, the Docker returns the container back to **Healthy** status.
- When the container health status changes, Docker Engine issues a health_status event.

Supposing we have an image as a simple Web service, we hope to enable health check to determine whether its Web service is working normally. We can use curl to help with the determination and the HEALTHCHECK instruction in its Dockerfile can be written like this:

```
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
```

We can use docker ps. After several seconds, we might find that 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-entrypoin..." 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-entrypoin..." 14 seconds ago Up 13 seconds (healthy) 9200/tcp,
9300/tcp elasticsearch
```

Another method is to 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 assist in troubleshooting, all output results of health check commands (including stdout and stderr) are stored in health statuses and you can view them with the docker inspect command. We can use the following command to retrieve the health check results from 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": "..."
},
...
}
```

Since app developers know better about the app SLA, we usually recommend declaring the corresponding health check policy in the Dockerfile to facilitate the use of images. App deployment and O&M personnel can adjust the health check policies as needed by using the command line parameters and REST API.

The Docker community provides some instance images that contain health checks. We can access 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 checking mechanism.
- At present, 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.

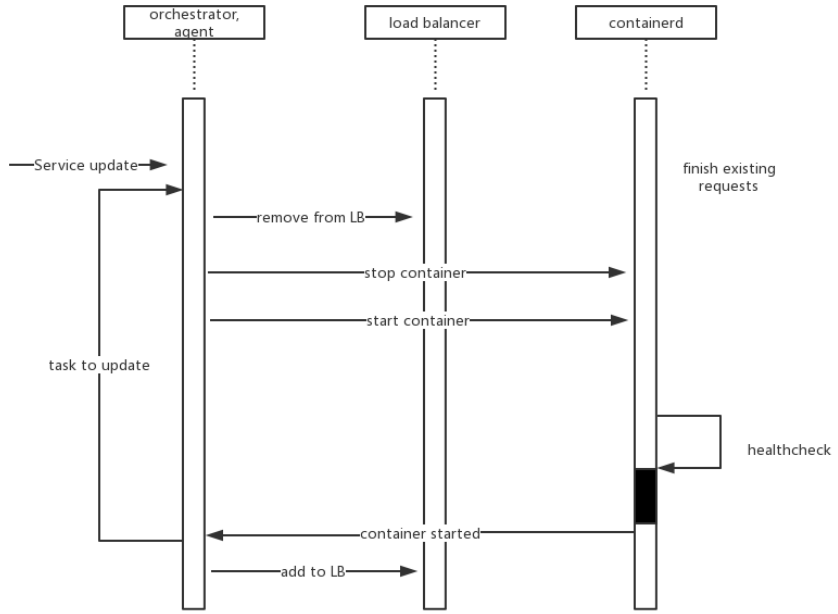
You can specify the health check policy in the docker service create command:

```
$ 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 statuses of service tasks. When a container enters the **Unhealthy** status, the container is stopped and a new container is restarted to replace the unhealthy one. The backend or DNS records of the Server Load Balancer (routing mesh) are automatically updated during this process to ensure 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 forward requests to a new container before it is fully started and enters the **Healthy** status, which ensures 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 help to ensure app availability. Many current application frameworks are already built with monitoring and checking capabilities, such as Spring Boot Actuator. With the help of Docker built-in health check mechanism, you can implement app availability monitoring, automatic fault handling and fault-free updating in a concise manner.

This document provides a Compose file for you to test the container connectivity within a cluster by visiting the access endpoint of the service.

Scenario

When you need to deploy interdependent applications in a Docker cluster, you must ensure that the applications can access one another, namely cross-host container network connectivity is available. However, due to network problems, the containers deployed on different hosts might not be able to access one another. If this happens, it is very difficult to troubleshoot the problem. Therefore, an easy-to-use Compose file that can be used to test the connectivity among cross-host containers within a cluster can really help a lot.

Solution

You can use the image and application template as shown below 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 above application template deploys a Web service and a Redis service. The Web service contains three Flask containers and the containers will be distributed on three nodes when started. Therefore, if the containers can ping one another, it means that the current network can realize cross-host container access.

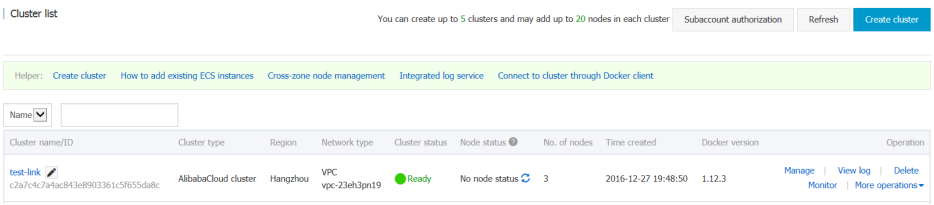
The Redis service runs on one of the three nodes. After the Flask containers start, they register to the Redis service and report their IP addresses; therefore, the Redis service has the IP addresses of all the three Flask containers after the Flask containers start. When you visit any of the three Flask containers, it will send ping commands to the other two 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 name of the cluster is **test-link**. For more information about how to create a cluster, refer to Create a cluster.

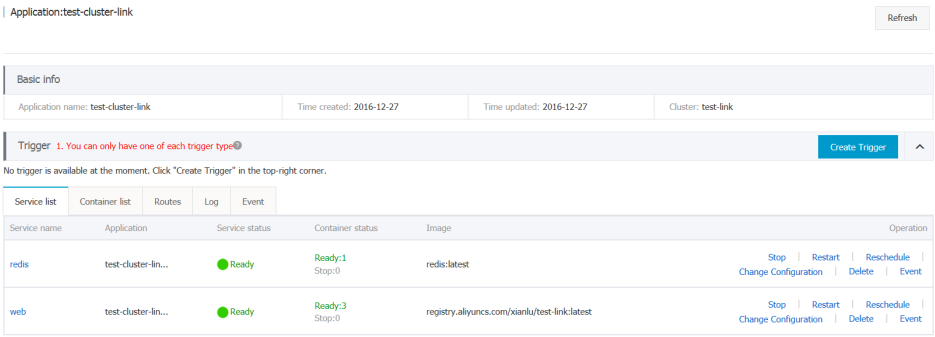
Note: Create a Server Load Balancer instance when you create the cluster.



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

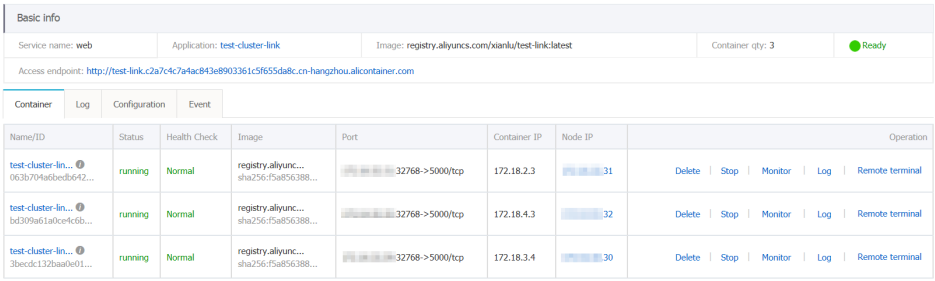
For more information about how to create an application, refer to Create an application.

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



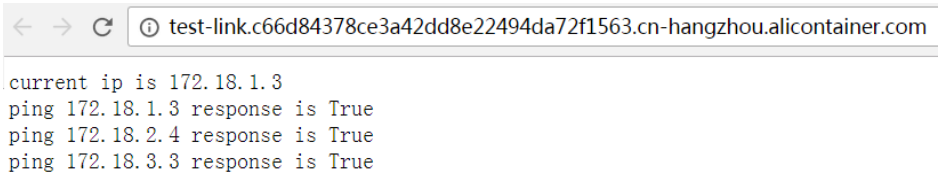
Click the name of the **web** service to enter the service details page.

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

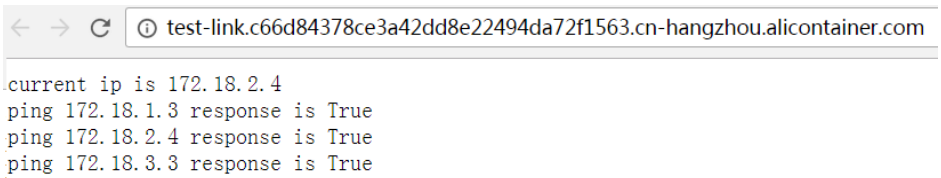


Visit the access endpoint of the **web** service.

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



Refresh the webage. As shown in the figure below, the container **test-cluster-link_web_2** can visit the container **test-cluster-link_web_1** and container **test-cluster-link_web_3**.



As the above results show, the containers in the cluster can access one another.

Use OSSFS data volumes to share WordPress attachments

This document describes how to share WordPress attachments among different containers by creating OSSFS data volumes on Alibaba Cloud Container Service.

Scenario

Docker containers simplify WordPress deployment. With Alibaba Cloud Container Service, you can use an application template for one-click deployment of WordPress.

> **Note:** For details about how to use Alibaba Cloud Container Service to create a WordPress application, refer to Create WordPress by using an application template.

In this example, the following application 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 consists of 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 details about the labels supported by Alibaba Cloud Container Service, refer to Label description). The WordPress containers access MySQL through a link. The aliyun.routing.port_80: http://wordpress label defines the load balancing among the three WordPress containers (for details, refer to Exposing HTTP service through acsrouting).

In this example, the application can be deployed with 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 once requests are routed to other containers.

Solution

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

OSSFS data volume is a third-party data volume provided by Alibaba Cloud Container Service to package various cloud storages (for example, OSS) into data volumes and to directly mount these data volumes to the containers. This means the data volumes can be shared across different containers and automatically re-mounted upon container restart and migration.

Operating procedure

Step 1: Create OSSFS data volumes

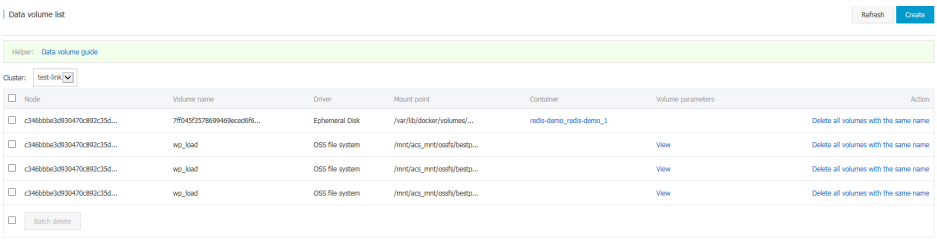
Log on to the Container Service console.

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

Select the desired cluster and click **Create** in the upper-right corner.

For details about how to create OSSFS data volumes, refer to Create an OSSFS data volume.

Here the created OSSFS data volumes are named **wp_upload**. The Container Service uses the same name to create data volumes on all nodes of a cluster.



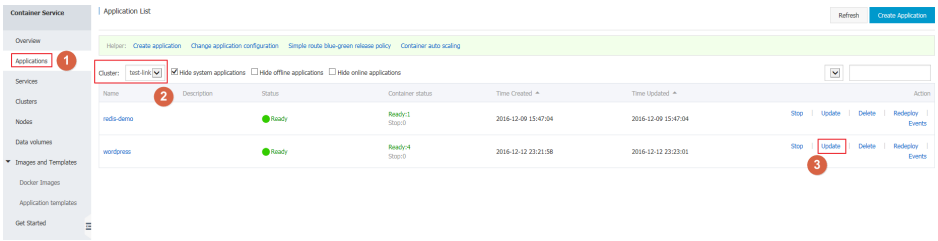
Step 2: Use the OSSFS data volumes

The WordPress attachments are stored in the /var/www/html/wp-content/uploads directory by default. In this example, we only need to map OSSFS data volumes to this directory for sharing of an OSS bucket across different WordPress containers.

Log on to the Container Service console.

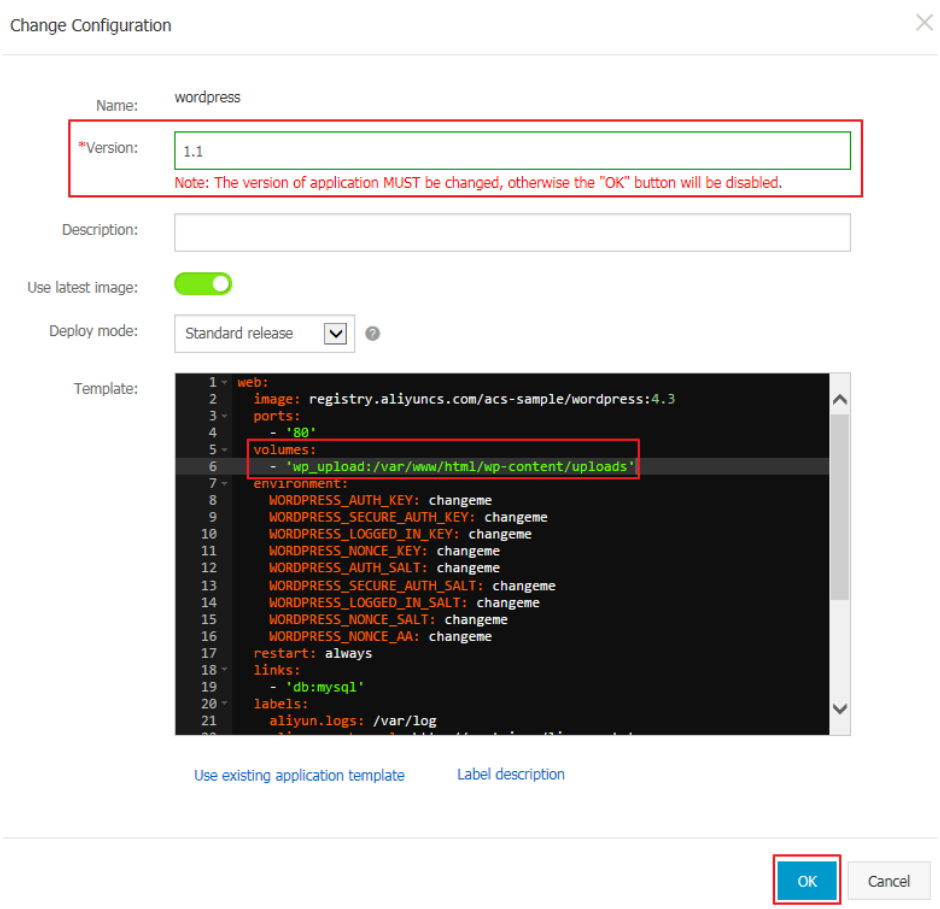
Click **Applications** in the left navigation pane.

Select the target cluster as well as the created application **wordpress** and click **Update** at the right side.



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

Note: You must modify the **Version**; otherwise, the application cannot be re-deployed.



Click **OK** to re-deploy the application.

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

Log

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.

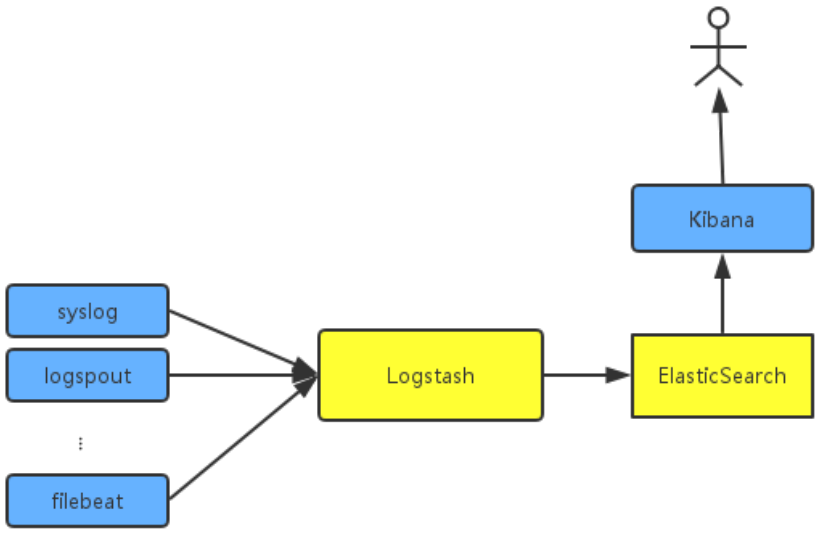
Logs are usually stored in the local log files. You need to log on to the server and filter keywords by using grep or other tools to view the logs. But when the application is deployed on multiple servers, this log viewing method is very inconvenient. To locate the logs for a specific error, you have to log

on to all the servers 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 containers and hosts is not fixed and containers might be constantly migrated between hosts. You cannot view the logs by logging on to the server 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. But 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 every server, not to mention in every Docker container. To collect the container logs, Syslog, Logspout, and Filebeat are used. Of course, 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 instances created in this document are all 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**.
3. 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.

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 created Server Load Balancer instance 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 orchestration templates, see Create an application.

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

```
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://${SLB_ID}:5044' #First create a Server Load Balancer instance
aliyun.lb.port_5000: 'tcp://${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 you have to make an image on your own to store the configuration file. The image source codes can be found in demo-logstash.

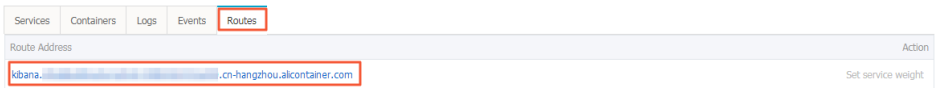
The Logstash configuration file is as follows. This is a very simple Logstash configuration. Two input formats, syslog and filebeats, are provided and their external ports are 5044 and 5000 respectively.

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

tcp {
port => 5000
type => syslog
}

}

filter {
}

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

stdout { codec => rubydebug }
}
```

Configure the Kibana index.

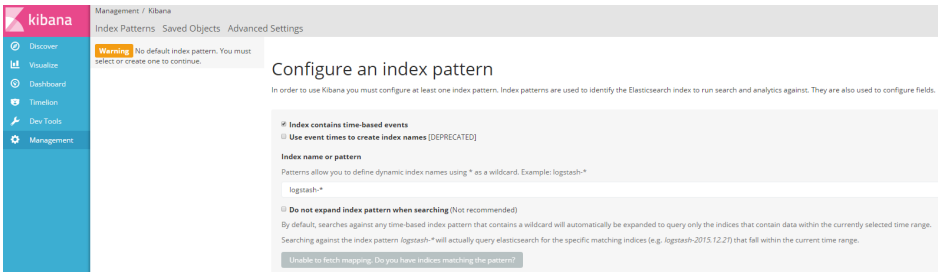
Access Kibana.

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 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 also use Filebeat directly. WordPress is used for the demonstration. The following is an 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 created Server Load Balancer instance.

```
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: fluentd-pilot

This document introduces a new log collector for Docker: Fluentd-pilot. Fluentd-pilot is a log collecting image we provide for you. You can deploy a Fluentd-pilot instance on each machine to collect logs of all the Docker applications.

Fluentd-pilot has the following features:

- A separate fluentd process to collect logs of all the containers on the machine. There is no need to start a fluentd process for each container.

- It supports file logs and stdout logs. Docker log drivers or Logspout can only process stdout, while fluentd-pilot not only supports collection of stdout logs, but also supports collection of file logs.
- Declarative configuration. When your container has logs to collect, the fluentd-pilot will automatically collect logs of the new container as long as the path of the log file to be collected is declared through the label, requiring no changes to any other configuration.
- It supports multiple log storage methods. Whether it is a powerful Alibaba Cloud Log Service, or the more popular ElasticSearch combination, or even Graylog, Fluentd-pilot can deliver the log to the correct location.
- Open-source. Fluentd-pilot is fully open-source. You can download its code here. If the current features cannot meet your requirements, welcome to raise an issue.

Quick boot

Next I will show a simple scenario: first start Fluentd-pilot, then start a Tomcat container, and let Fluentd-pilot collect Tomcat logs. For the sake of simplicity, here Alibaba Cloud Log Service or ELK is not involved. If you want to run it locally, you just need a machine that runs Docker.

First, start Fluentd-pilot.

> **Note:** When Fluentd-pilot is started in this approach, because there is no log storage configured for backend use, all the collected logs will be directly output to the console.

Open the terminal and input the following commands:

```
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
```

You will see the startup logs of Fluentd-pilot.

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 it is very 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 Fluentd-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 detail later.

Note: If you deploy Tomcat locally, instead of on the Alibaba Cloud container service, you should specify -v /usr/local/tomcat/logs. Otherwise, Fluentd-pilot may be unable to read log files. The Container Service has implemented the optimization and you don't need to specify -v on your own.

Fluentd-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 starts to collect the corresponding logs. After you start Tomcat, you will find a pile of contents is output immediately by the Fluentd-pilot terminal, including the stdout logs output at the Tomcat startup, and some debugging information output by Fluentd-pilot itself.

You can access the just-deployed Tomcat in the browser, and you will find that similar records can be found on the Fluentd-pilot terminal every time you refresh the browser. In specific, the content after message is the logs collected from /usr/local/tomcat/logs/localhost_access_log.XXX.txt.

Use ElasticSearch + Kibana

First you should deploy ElastichSearch + Kibana. You can refer to Use ELK in the Container Service to deploy ELK in the Alibaba Cloud Container Service, or deploy them directly on your machine following the ElasticSearch/Kibana documents. This document assumes that you have deployed the two components.

If you are still running the Fluentd-pilot, close it first, and then start it again using the command below.

Note: Before executing the following commands, first replace the two variables of ELASTICSEARCH_HOST and ELASTICSEARCH_PORT with the actual values you are using.

ELASTICSEARCH_PORT is usually 9200.

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

Compared with the previous Fluentd-pilot startup method, here three environmental variables are added:

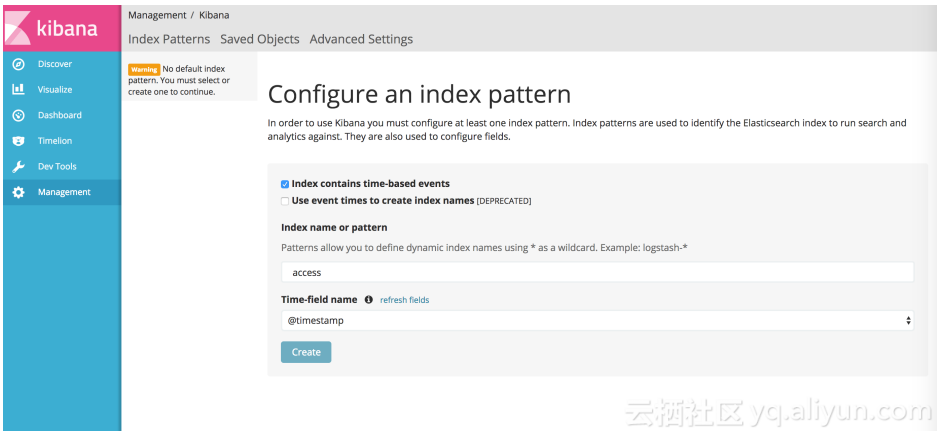
- FLUENTD_OUTPUT=elasticsearch: Send the logs to ElasticSearch.
- ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST}: The domain name of ElasticSearch.
- ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT}: The port number of ElasticSearch.

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

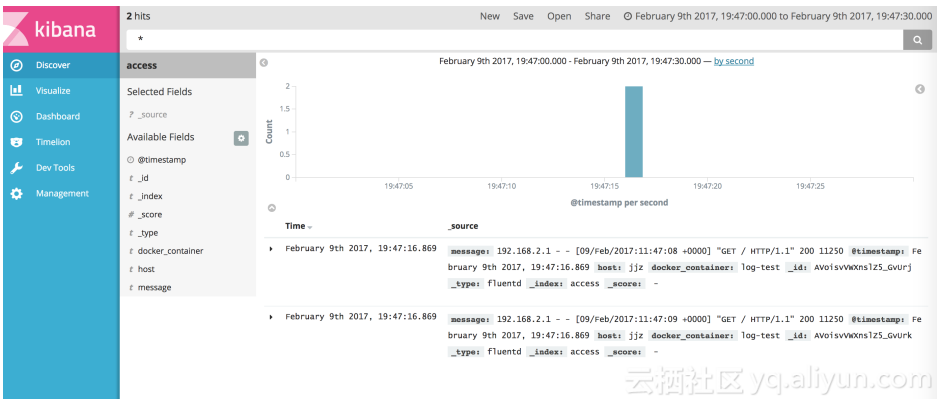
Open Kibana, and no new logs are visible yet. You need to create an index first. Fluentd-pilot will write logs to the specific index of ElasticSearch. The rules are as follows:

If the tag aliyun.logs.tags is used in the application, and the tags contains target, you should make the target as the index in ElasticSearch. Otherwise, you should make the XXX in the tag aliyun.logs.XXX as the index.

In the previous example about Tomcat, the tag 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 ready, you can view the logs.



Use Fluentd-pilot in the Alibaba Cloud Container Service

The Container Service makes some special optimization for Fluentd-pilot to best adapt to running Fluentd-pilot.

To run Fluentd-pilot in the Container Service, you only need to create a new application using the following orchestration file. For how to create an application, see Create an application.

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

Next you can use the aliyun.logs.xxx tag on the application you want to collect logs for.

Label description

When Tomcat is started, the following two labels are declared to tell Fluentd-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 hyphen

(-).

- The variable path is the path of the logs to collect. It must specify the file, and should not only be a directory. Wildcards are supported as part of the file name, such as /var/log/he.log and /var/log/*.log. However, /var/log is not valid as the path should not be only a directory. stdout is a special value, indicating standard output.

aliyun.logs.$name.format: the log format. Currently only the following formats are supported.

- none: unformatted plain text.
- json: JSON format. One 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, such as aliyun.logs.access.tags="name=hello,stage=test". 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 the ElasticSearch.

Fluentd-pilot extension

For most users, the existing features of Fluentd-pilot can meet their requirements. If Fluentd-pilot is unable to meet your requirements, you can:

- Submit an issue at https://github.com/jzwlqx/fluentd-pilot/issues.
- Directly change the code and then raise the PR.