容器服务

最佳实践

最佳实践

在阿里云容器服务上运行基于 TensorFlow 的 Alexnet

AlexNet 是 2012 年由 Alex Krizhevsky 使用五层卷积、三层完全连接层开发的 CNN 网络，并赢得了 ImageNet 竞赛（ILSVRC）。AlexNet 证明了 CNN 在分类问题上的有效性（15.3% 错误率），而此前的图片识别错误率高达 25%。这一网络的出现对于计算机视觉在深度学习上的应用具有里程碑意义。

AlexNet 也是深度学习框架常用的性能指标工具，TensorFlow 就提供的 alexnet_benchmark.py 可以测试 GPU 和 CPU 上的性能。本文档以 AlexNet 为例，向您展示如何在阿里云容器服务上简单快速地运行 GPU 应用。

前提条件

需要基于北京 HPC 或者 GN4 规格族 GPU 云服务器的容器服务。

- 创建基于北京 HPC 的容器集群
- 创建 GN4 型 GPU 云服务器集群

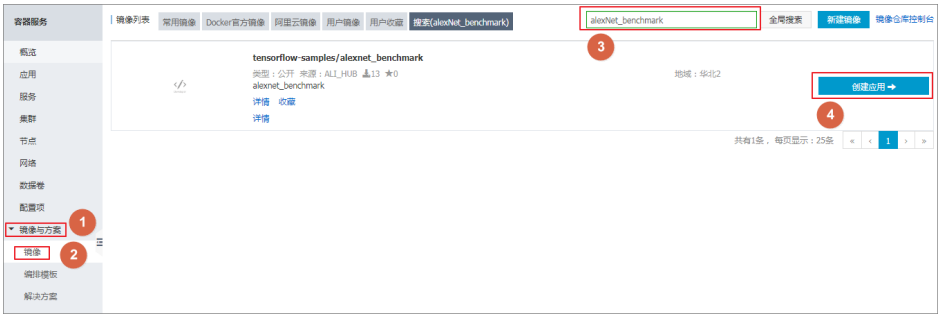
操作步骤

登录 容器服务的管理控制台。

单击左侧导航栏中的 **镜像与方案** > **镜像**。

在搜索框中输入 alexNet_benchmark 并单击 **全局搜索**。

单击 registry.cn-beijing.aliyuncs.com/tensorflow-samples/alexnet_benchmark:1.0.0-devel-gpu 右边的 **创建应用**。



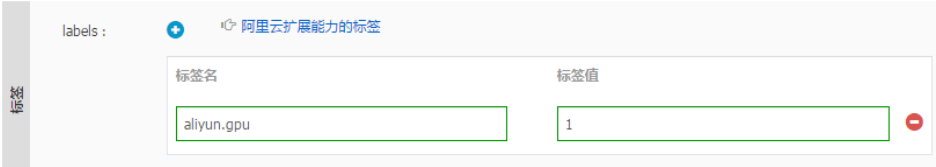
输入应用名称（本示例中为 **alexNet**）并选择北京 HPC 或者 GN4 规格族 ECS 集群, 单击 **下一步**。

配置应用。

在 **基本配置** 中，单击 **选择镜像版本**，选择镜像版本为 1.0.0-devel-gpu。

在 **容器配置** 中，填写运行的命令行，比如 python /alexnet_benchmark.py --batch_size 128 --num_batches 100。

在 **标签** 中，填写阿里云 gpu 标签，标签名为 aliyun.gpu，标签值为调度的 GPU 数量，本示例中为 1。

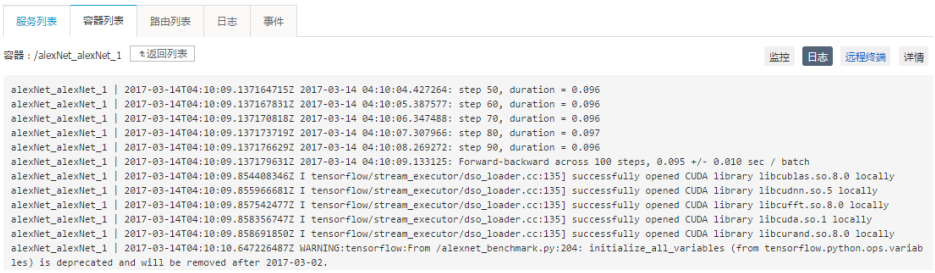


完成应用配置后，单击 **创建** 创建应用。

您可以在 **应用列表** 页面，查看创建的 **alexNet** 应用。

这样您就可以在管理控制台，直接通过容器日志服务查看 AlexNet 在 EGS 或者 HPC 上的性能。

操作路径：在应用列表页面，单击应用名称 **alexNet** > 单击 **容器列表** 页签 > 单击容器右边的 **日志**

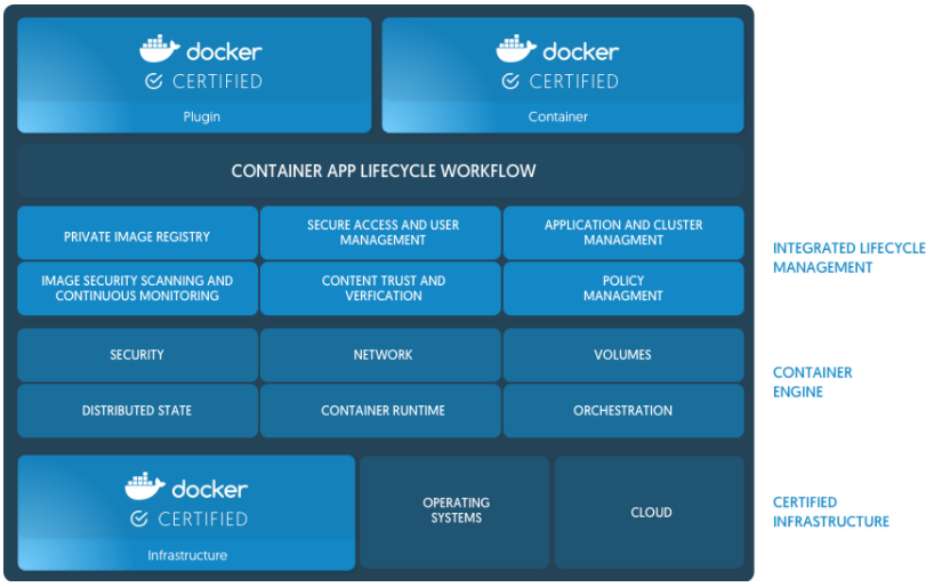


DDC 简介

Docker Datacenter（DDC）是 Docker 发布的企业级容器管理和服务部署的整体解决方案平台。DDC 由以下三个组件构成：

- Docker Universal Control Plane（Docker UCP）：一套图形化管理界面。
- Docker Trusted Registry（DTR）：授信的 Docker 镜像仓库。
- Docker Engine 商业版：提供技术支持的 Docker 引擎。

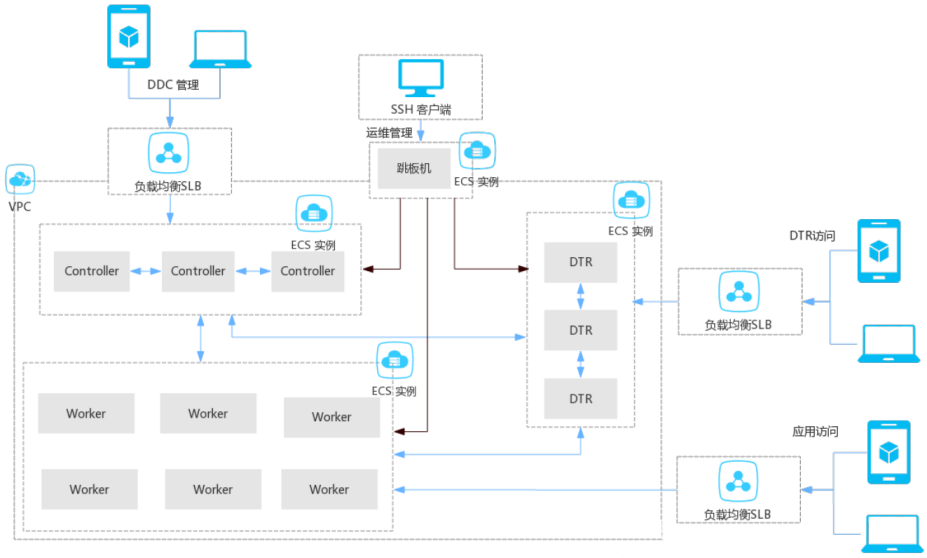
DDC 在 Docker 官网的地址为 http://www.docker.com/products/docker-datacenter#。



DDC 与 Docker 公司的另外一个在线产品 Docker Cloud 对应。不过 DDC 主要针对企业用户在内部部署。用户注册自己的 Dokcer 镜像到 DTR，UCP 管理整个 Docker 集群。并且这两个组件都提供了 Web 界面。

使用 DDC 需要购买 Licence， 但是 Docker 公司提供了一个月的试用 Licence，可以在 Docker 官网注册后直接下载。

DDC 部署架构



在上面的基础架构图里，Controller 主要运行 UCP 组件，DTR 运行 DTR 组件， Worker 主要运行客户自己的 Docker 服务。整个 DDC 环境都部署在 VPC 网络之下，所有的 ECS 加入同一个安全组。每个组件都提供了一个负载均衡，供外网访问。而运维操作则是通过跳板机实现。另一方面为了提升可用性，整个 DDC 环境都是高可用部署，也就是说 Controller 至少有两台，同理 DTR 也至少有两台。

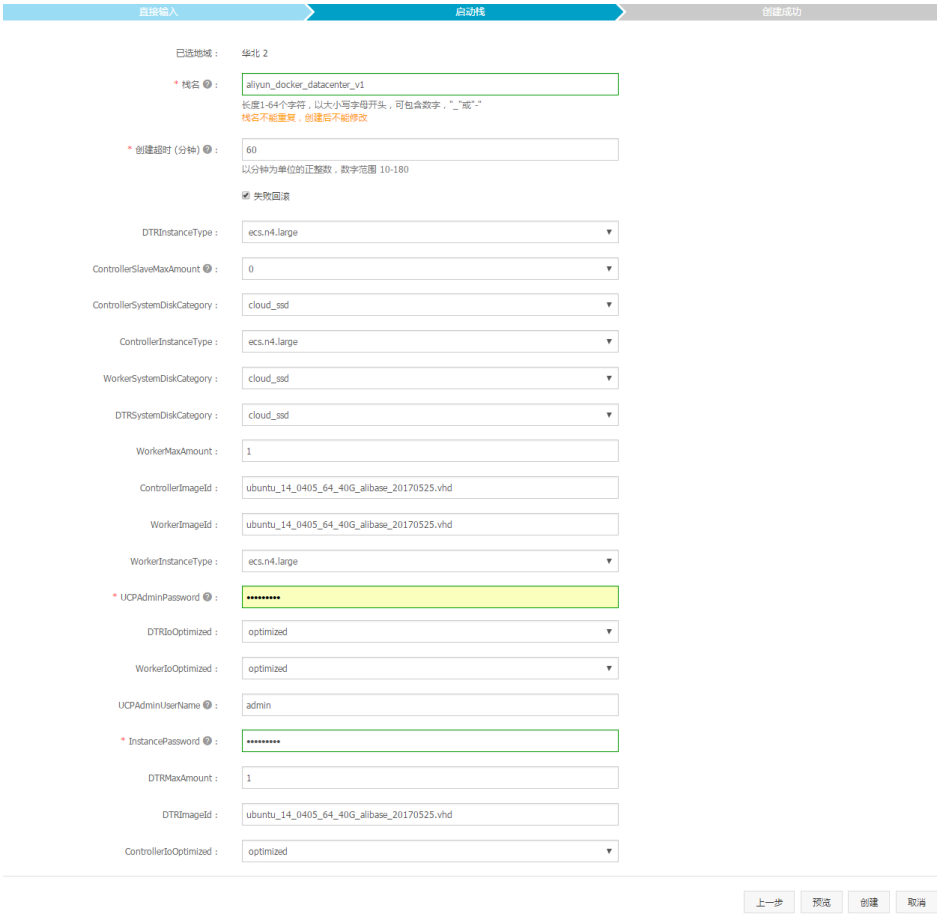
DDC 一键部署

您可以使用阿里云资源编排 ROS 通过下面的链接一键部署 DDC。

一键部署 DDC

上边的编排模板默认会在华北 2 地域部署 DDC。如果您需要调整地域，请单击页面右下角的 **上一步**，然后重新选择地域，然后单击 **下一步**。

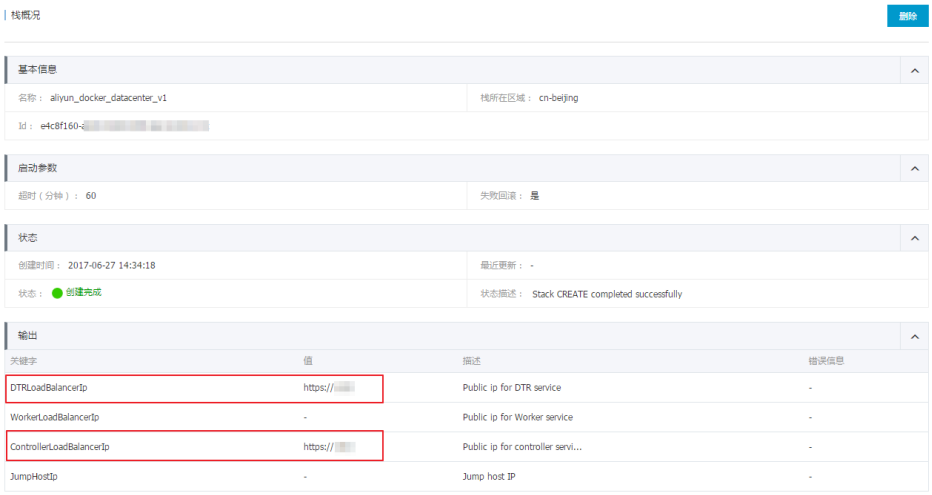
填写如下图中必填的信息或者根据您的需求调整信息后，单击 **创建** 就可以部署一套 DDC。



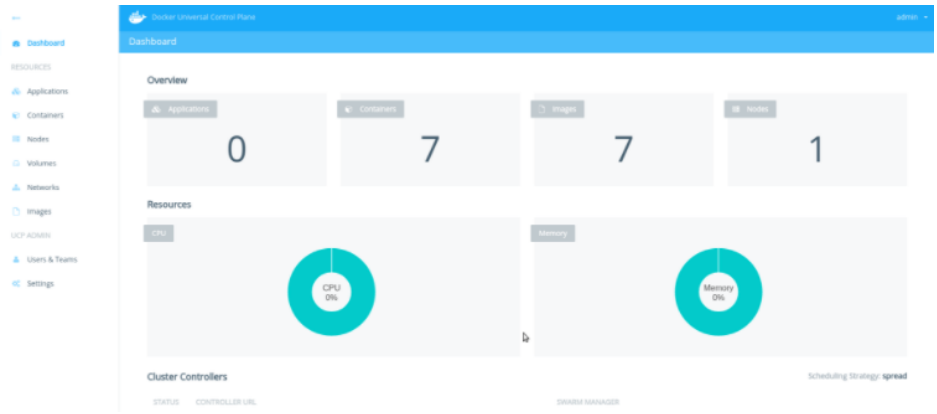
DDC 访问

使用 ROS 创建 DDC 成功后，您可以进入 ROS 的资源栈管理页面，找到所创建的资源栈，并单击资源栈名称或单击右侧的 **管理** 进入资源栈的概要信息页面。

您可以在 **输出** 查看登录 UCP 和 DTR 的地址。



在浏览器中输入 UCP 的地址就会显示 UCP 的访问页面，输入在安装 UCP 时创建的管理账号和密码，系统会提示导入 Licence 文件，把准备好的 Licence 导入，即可进入 UCP 的控制界面。



在分布式系统中，经常需要利用健康检查机制来检查服务的可用性，防止其他服务调用时出现异常。自 1.12 版本之后，Docker 引入了原生的健康检查实现。本文将介绍 Docker 容器健康检查机制，以及在 Docker Swarm mode下面的新特性。

对于容器而言，最简单的健康检查是进程级的健康检查，即检验进程是否存活。Docker Daemon 会自动监控容器中的 PID1 进程，如果 docker run 命令中指明了 restart policy，可以根据策略自动重启已结束的容器。在很多实际场景下，仅使用进程级健康检查机制还远远不够。比如，容器进程虽然依旧运行却由于应用死锁无法继续响应用户请求，这样的问题是无法通过进程监控发现的。

Kubernetes 提供了 Liveness 与 Readness 探针分别对 Container 及其服务健康状态进行检查。阿里云容器服务也提供了类似的 服务健康检查机制。

Docker 原生健康检查能力

自 1.12 版本之后，Docker 引入了原生的健康检查实现，可以在 Dockerfile 中声明应用自身的健康检测配置。HEALTHCHECK 指令声明了健康检测命令，用这个命令来判断容器主进程的服务状态是否正常，从而比较真实的反应容器实际状态。

HEALTHCHECK 指令格式：

- HEALTHCHECK [选项] CMD <命令>：设置检查容器健康状况的命令。
- HEALTHCHECK NONE ：如果基础镜像有健康检查指令，使用这行可以屏蔽。

注意：在 Dockerfile 中 HEALTHCHECK只可以出现一次，如果写了多个，只有最后一个生效。

使用包含 HEALTHCHECK 指令的 dockerfile 构建出来的镜像，在实例化 Docker 容器的时候，就具备了健康状态检查的功能。启动容器后会自动进行健康检查。

HEALTHCHECK 支持下列选项：

- --interval=<间隔>：两次健康检查的间隔，默认为 30 秒。
- --timeout=<间隔>：健康检查命令运行超时时间，如果超过这个时间，本次健康检查就被视为失败，默认 30 秒。
- --retries=<次数>：当连续失败指定次数后，则将容器状态视为 unhealthy，默认 3 次。
- --start-period=<间隔>: 应用的启动的初始化时间，在启动过程中的健康检查失效不会计入，默认 0 秒（从 V17.05 引入）。

在 HEALTHCHECK [选项] CMD 后面的命令，格式和 ENTRYPOINT 一样，分为 shell 和 exec 格式。命令的返回值决定了该次健康检查的成功与否：

- 0：成功
- 1：失败
- 2：保留值，不要使用

容器启动之后，初始状态会为 starting（启动中）。Docker Engine 会等待 interval 时间，开始执行健康检查命令，并周期性执行。如果单次检查返回值非 0 或者运行需要比指定 timeout 时间还长，则本次检查被认为失败。如果健康检查连续失败超过了 retries 重试次数，状态就会变为 unhealthy（不健康）。

- 一旦有一次健康检查成功，Docker 会将容器置回 healthy（健康）状态。
- 当容器的健康状态发生变化时，Docker Engine 会发出一个 health_status 事件。

假设我们有个镜像是个最简单的 Web 服务，我们希望增加健康检查来判断其 Web 服务是否在正常工作，我们可以用 curl 来帮助判断，其 Dockerfile 的 HEALTHCHECK 可以这么写：

```
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
```

我们可以通过 docker ps，发现过了几秒之后，Elasticsearch 容器从 starting 状态进入了 healthy 状态。

```
$ 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
```

另外一种方法是在 docker run 命令中，直接指明 healthcheck 相关策略。

```
$ 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
```

为了帮助排障，健康检查命令的输出（包括 stdout 以及 stderr）都会被存储于健康状态里，可以用 docker inspect 来查看。我们可以通过如下命令，来获取过去5个容器的健康检查结果。

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

或

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

示例结果如下：

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

由于应用的开发者会更加了解应用的 SLA，一般建议在 Dockerfile 中声明相应的健康检查策略，这样可以方便镜像的使用。对于应用的部署和运维人员，可以通过命令行参数和 REST API 针对部署场景对健康检查策略按需进行调整。

Docker社区提供了一些包含健康检查的实例镜像，我们可以在如下项目中获取 https://github.com/docker-library/healthchec 。

注意：

- 阿里云容器服务同时支持 Docker 原生健康检测机制和阿里云的扩展检查机制。
- 目前 Kubernetes 还不提供对 Docker 原生健康检查机制的支持。

Docker Swarm mode 中的服务健康检查能力

在 Docker 1.13之后，在 Docker Swarm mode 中提供了对健康检查策略的支持。

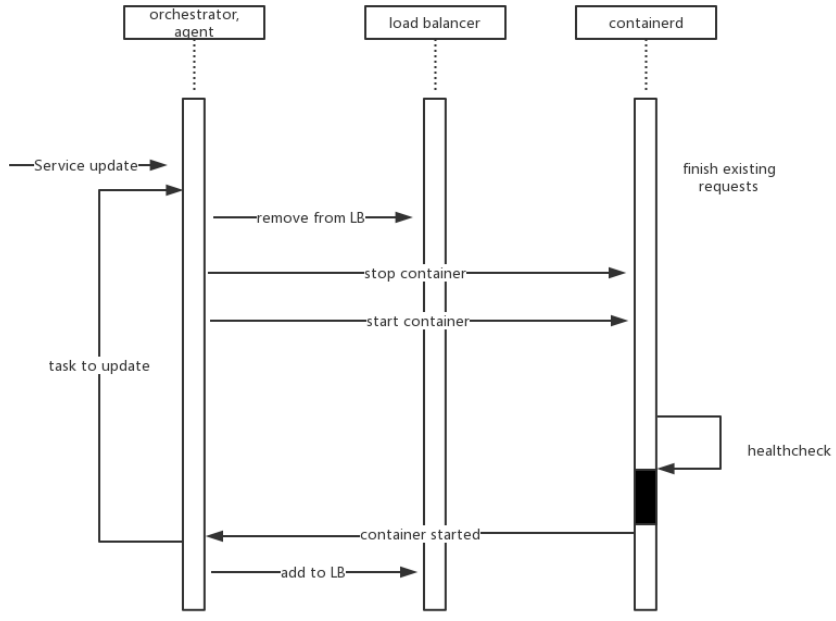
可以在 docker service create 命令中指明健康检查策略：

```
$ 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
```

在 swarm mode 模式下，Swarm manager 会监控服务 task 的健康状态，如果容器进入 unhealthy 状态，它会停止容器并且重新启动一个新容器来取代它。这个过程中会自动更新服务的 load balancer （ routing mesh ）后端或者 DNS记录，可以保障服务的可用性。

在 1.13 版本之后，在服务更新阶段也增加了对健康检查的支持，这样在新容器完全启动成功并进入健康状态之前，load balancer/DNS 解析不会将请求发送给它。这样可以保证应用在更新过程中请求不会中断。

下面是在服务更新过程的时序图。



在企业生产环境中，合理的健康检查设置可以保证应用的可用性。现在很多应用框架已经内置了监控检查能力，比如 Spring Boot Actuator。配合 Docker 内置的健康检测机制，可以非常简洁实现应用可用性监控，自动

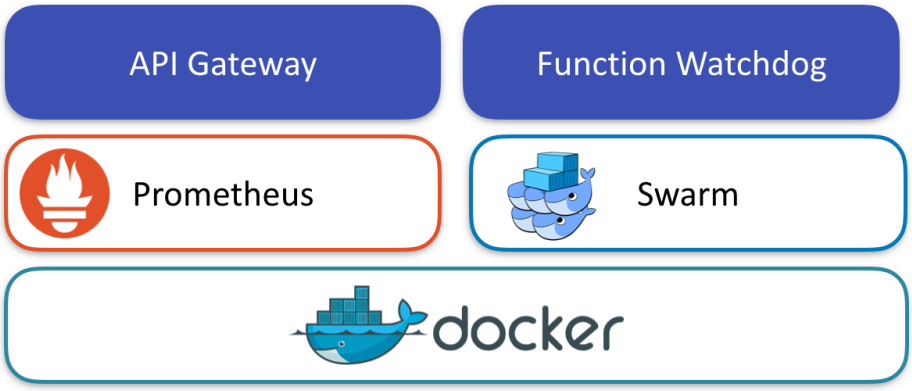
故障处理，和零宕机更新。

函数计算（FaaS）是目前最新的云服务模式，阿里云容器服务基于 swarm mode 集群上实现了一个极简的 Serverless 框架，支持将任意 Unix 进程作为函数实现来对外提供服务。更多扩展知识，可参考 阿里云函数计算。

架构原理

在 FaaS 原型系统中，包括以下几个模块，其中：

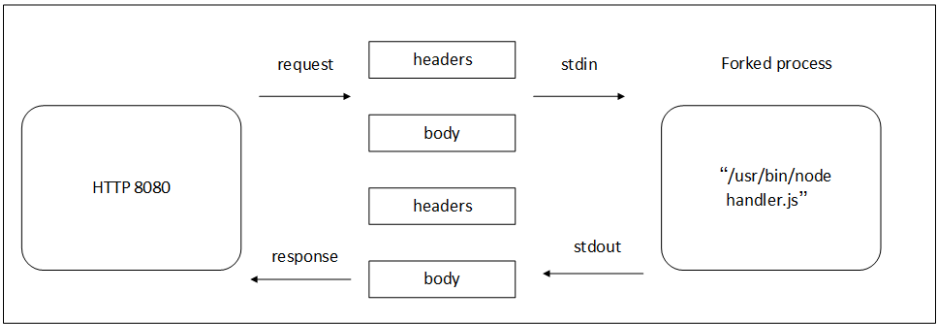
1. 任何进程都可以转化成为一个函数，并利用 Docker 镜像进行打包和交付。
2. 利用 Docker Swarm 集群的资源调度和 routing mesh 的负载均衡能力简洁地实现了函数的调度能力。其中每个函数对应一个 Docker 集群中的服务。
3. 基于 Prometheus 实现函数调用监控和自动伸缩。



其设计架构非常简单，其中：

- API Gateway 负责接受服务调用，路由请求到后端函数实现，并采集服务调用的指标发送给 Prometheus。 Prometheus 则会根据一段时间内服务调用的次数，回调 API Gateway 来动态伸缩服务容器实例数量。

  Function Watchdog 将 HTTP 请求转发为进程调用，并将请求数据通过 STDIN 传递给进程，而将进程的 STDOUT 作为 HTTP 响应的结果返回给调用者。将函数进程和 Function Watchdog 打包成一个容器镜像进行部署。其调用流程如下：



本地安装 FaaS

首先您需要准备一个本地的 Docker swarm集群，如果没有，可以安装最新 Docker Engine 并执行下面命令。

docker swarm init

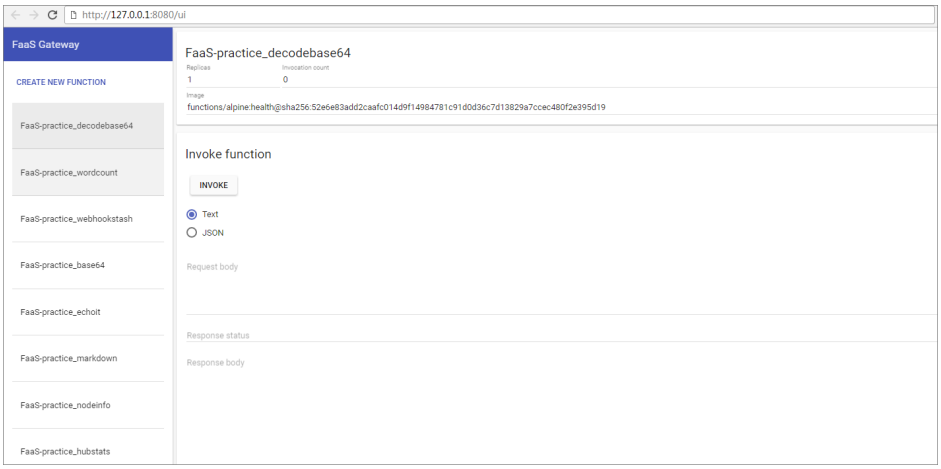
执行如下命令来部署 FaaS。

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

部署完成后，您可以通过如下命令检查 FaaS 的状态。

```
$ 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
```

随后通过浏览器来访问 http://127.0.0.1:8080/ui FaaS。



在阿里云上测试 FaaS

限制条件

需要具备创建 swarm mode 集群的条件。

- 默认情况下，您最多可以创建 5 个集群（所有地域下），每个集群中最多可以添加 20 个节点。如果您需要创建更多的集群或添加更多的节点，请提交 工单申请。
- 随集群一同创建的负载均衡实例只支持按量付费的方式。
- 用户账户需有 100 元的余额并通过实名认证，否则无法创建按量付费的 ECS 实例和负载均衡。

操作步骤

创建 swarm mode 集群。
FaaS 基于 Docker swarm mode 集群进行部署的，您首先需要在阿里云容器服务创建一个 swarm mode 集群。

使用编排模板创建应用，参见 使用编排模板创建应用。

编排示例如下：

```
 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:
```

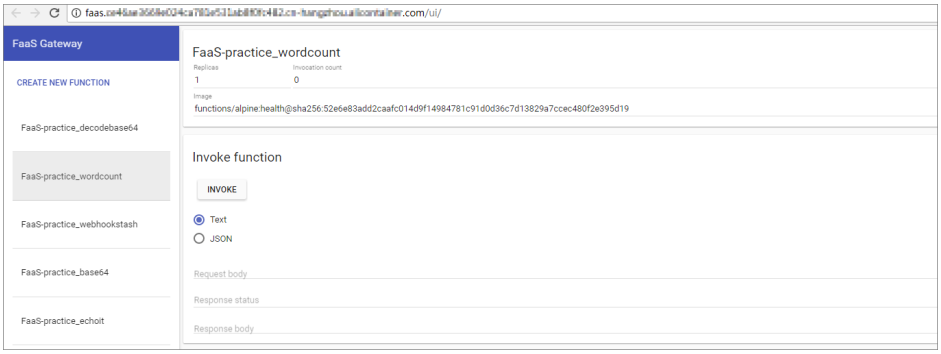
```
- gateway
networks:
- functions
environment:
fprocess: "base64"
no_proxy: "gateway"
https_proxy: $https_proxy

# 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
```

和本地部署相比只是增加了两个 label，定义了 API Gatway 和 Prometheus 的路由。

- aliyun.routing.port_8080: faas： API Gatway 的虚拟域名。
- aliyun.routing.port_9090: prometheus： prometheus 服务的虚拟域名。

查看应用的 **路由列表**。

单击路由地址，访问 FaaS 的 API Gateway 和 Prometheus 服务界面。



后续

你可以在这个基础上，测试服务的伸缩性，具体请参考：

- 夏日清风 - 基于 Docker Swarm 的极简 Serverless 实践
- 开源 github 项目地址

本文档提供一个简单的 Compose file 来实现一键部署，您可以通过访问 endpoint 来检查容器间的互通性。

场景

在 Docker 集群中部署相互依赖的应用时，需要应用之间可以互相访问，实现跨 host 间容器网络的互通性。而有时候因为网络问题，会导致 link 之间不能互相访问。类似情况下，很难定位发生错误的原因。因此，使用一套简单易用的 Compose file 来测试当前集群网络各个主机上容器之间是否互通变得很重要。

解决方案

使用我们提供的镜像和编排模板测试容器间的互通性。

```
 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
```

本示例使用 Flask 来完成测试功能。

以上编排模板部署了一个 Web 服务和一个 Redis 服务。Web 服务包含三个 Flask 容器。Flask 容器启动时，会被平均分配到三个节点上运行。这就保证了三个容器在不同的宿主机上，只要能互相 ping 通，就说明当前网络是可以实现容器跨主机互联的。Redis 运行在其中一台机器上，每个 Flask 容器启动后都会向 Redis 注册，报告自己的 IP 地址。当三个 Flask 容器都启动完毕后，Redis 中也就有了集群中所有容器的 IP 地址。访问其中任意一个 Flask 容器，该容器就会向另外两个容器发起 ping 命令，通过 ping 的结果确定当前集群的网络连通性。

操作步骤

创建一个包含三个节点的集群。

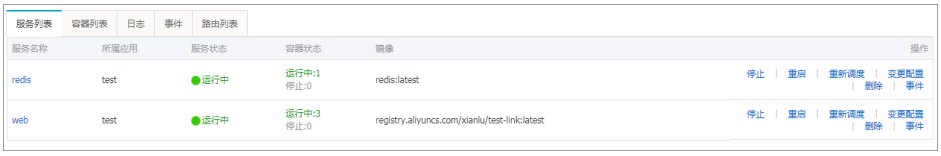
本示例中集群的名称为 **test-link**。有关如何创建集群的详细信息，参见 **创建集群**。

> **注意**：创建集群时，请创建负载均衡实例。

使用上面的模板，创建一个应用（本示例中创建名为 **test-cluster-link** 的应用），部署 **web** 服务和 **redis** 服务。

有关如何创建应用的详细信息，参见 **创建应用**。

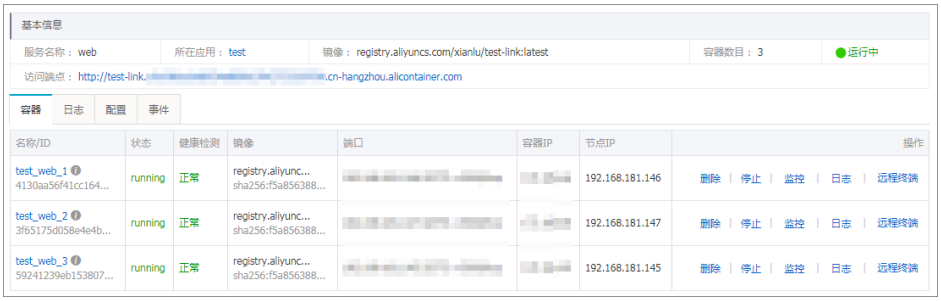
在 **应用列表** 页面，单击应用的名称，查看创建的服务。



单击 **web** 服务的名称，进入服务详情页面。

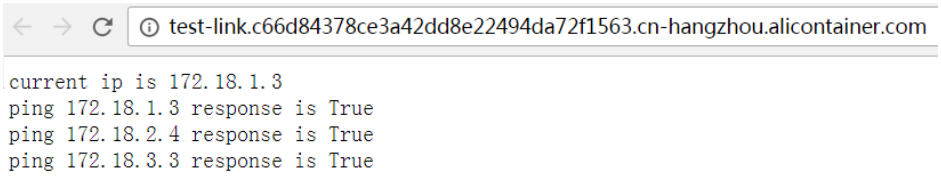
可以看到，三个容器（**test-cluster-link_web_1**、 **test-cluster-link_web_2**、**test-cluster-**

link_web_3）都已经启动了，并且分布在不同的节点上。

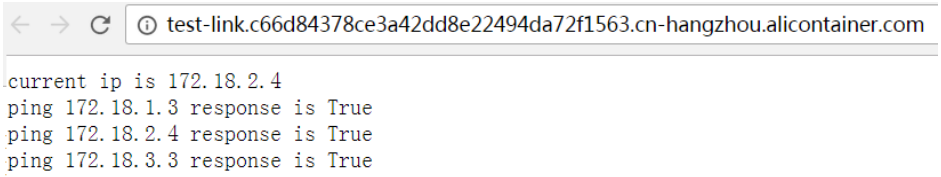


访问 web 服务的访问端点。

从页面的反馈来看，容器 test-cluster-link_web_1 可以访问容器 test-cluster-link_web_2 和 test-cluster-link_web_3。



刷新一下页面。可以看到，容器 test-cluster-link_web_2 可以访问容器 test-cluster-link_web_1 和 test-cluster-link_web_3。



以上结果显示当前集群容器之间是互通的。

本文档介绍如何通过在阿里云容器服务上创建 OSSFS 数据卷来实现 WordPress 的附件在不同容器之间的共享。

场景

Docker 容器的兴起使得 WordPress 的部署变得很简单。通过 阿里云容器服务，您可以使用编排模板一键部署 WordPress。

> **注意**：有关使用阿里云容器服务创建 WordPress 应用的详细信息，参见 通过编排模板创建 WordPress。

本示例使用以下编排模板创建一个名为 wordpress 的应用。

```
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
```

该应用包含一个 MySQL 容器和三个 WordPress 容器（aliyun.scale: '3' 是阿里云容器服务的扩展标签，指定容器的数量。有关阿里云容器服务支持的标签，参见 标签说明）。WordPress 容器通过 link 访问 MySQL。通过定义 aliyun.routing.port_80: http://wordpress 标签实现了三个 WordPress 容器的负载均衡（详细信息参见 简单路由（支持HTTP/HTTPS））。

本示例部署简单，功能齐全，但其实存在一个致命的缺陷。WordPress 上传的附件是保存在本地磁盘上的，不同容器之间不能共享。当请求被分配到其它容器时，附件就打不开了。

解决方案

本文档介绍如何利用阿里云容器服务的 OSSFS 数据卷（OSSFS volume），无需改动任何代码，即可实现 WordPress 附件在不同容器之间的共享。

OSSFS 数据卷是阿里云容器服务提供的第三方数据卷，通过将各种云存储（比如 OSS）包装成数据卷，直接挂载在容器上。不同容器间可以共享数据卷，并在容器重启、迁移时自动重新挂载数据卷。

操作流程

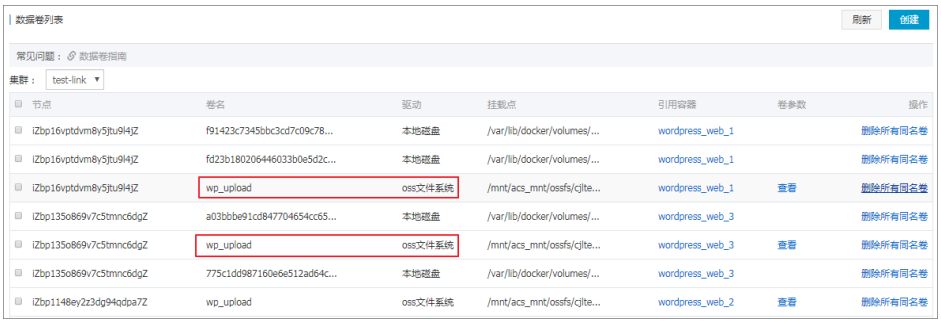
创建 OSSFS 数据卷。

在 **容器服务管理控制台**，单击左侧导航栏中的 **数据卷**，即可使用数据卷功能。

选择需要创建数据卷的集群并单击右上角的 **创建**，按照提示创建 OSSFS 数据卷。

有关如何创建 OSSFS 数据卷的详细信息，参见 **创建 OSSFS 数据卷**。

本示例中创建的 OSSFS 数据卷名称为 **wp_upload**。容器服务会在集群的所有节点上使用同一名称创建数据卷。如下图所示。

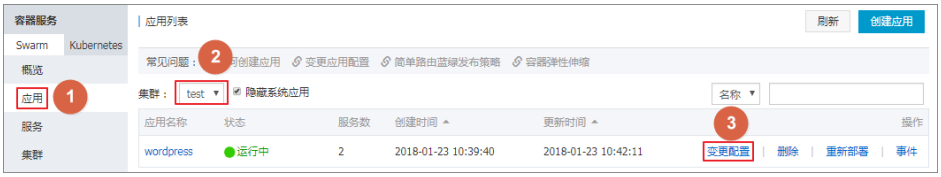


使用 OSSFS 数据卷。

WordPress 的附件，默认存放在 /var/www/html/wp-content/uploads 中。本示例中，只需将 OSSFS 数据卷映射到该目录，即可实现在不同的 WordPress 容器之间共享同一个 OSS bucket。

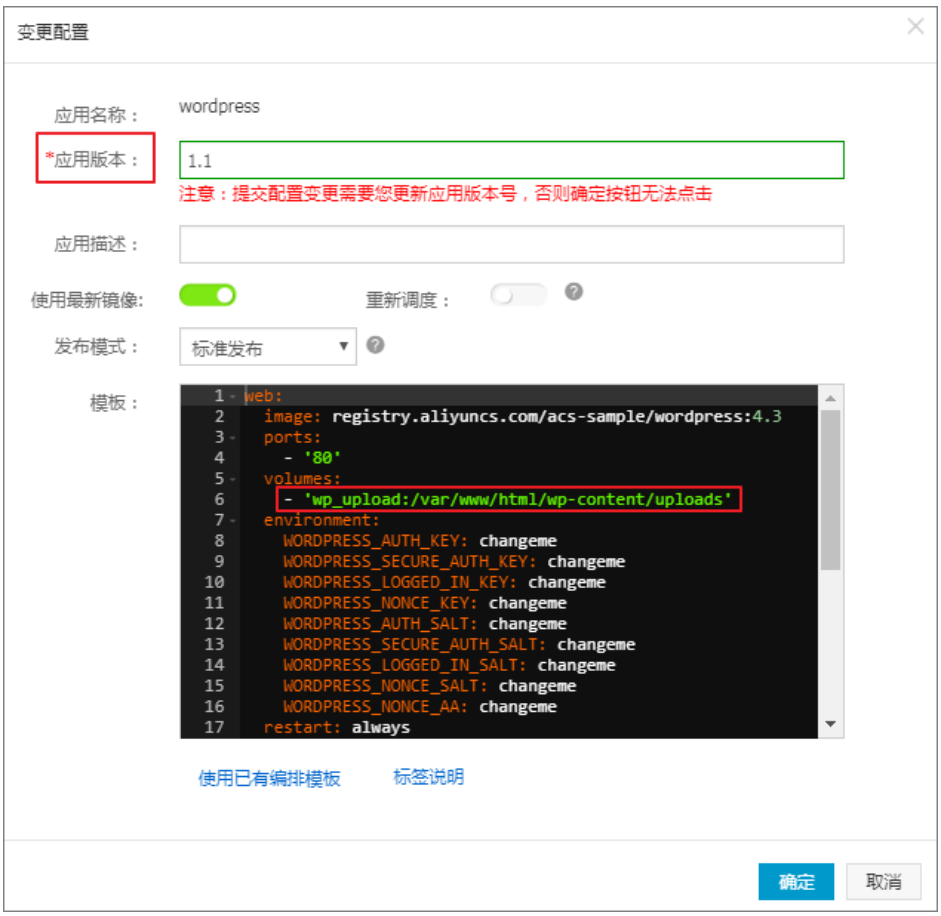
在 **容器服务管理控制台**，在 Swarm 菜单下，单击左侧导航栏中的 **应用**。

选择本示例中所使用的集群，选择本示例中所创建的应用 **wordpress** 并单击右侧的 **变更配置**。



在 **模板** 中添加 OSSFS 数据卷到 WordPress 目录的映射。

注意：您必须修改 **应用版本**，否则无法重新部署应用。



单击 **确定**，重新部署应用。

打开 WordPress，上传附件，OSS bucket 里就能看到上传的附件了。

镜像服务

Docker 的镜像存储中心通常被称为 Registry。

当您需要获取 Docker 镜像时，首先需要登录 Registry，然后拉取镜像。在您修改过镜像之后，您可以再次将镜像推送到 Registry 中去。

基本概念

Docker的镜像地址是什么？我们来看一个完整的例子。以容器服务的公共镜像为例，registry.cn-hangzhou.aliyuncs.com/acs/agent:0.8。

- registry.cn-hangzhou.aliyuncs.com：Registry 域名。
- acs：命名空间。
- agent：仓库名称。
- 0.8：Tag、镜像标签（非必须，默认为 latest）。

将这个几个完全独立的概念组合一下：

- registry.cn-hangzhou.aliyuncs.com/acs/agent：仓库坐标。
- acs/agent：仓库全名（通常在 API 中使用）。

基本使用

本文档的重点是介绍 Docker 最常用的三个命令：login、pull、push。

docker login

以阿里云杭州公网 Registry 为例。

登录时必须指明 Registry 域名，并输入您的用户名和登录密码。

注意：此处的登录密码是您在 **镜像仓库管理控制台** 设置的，而不是您的阿里云登录密码。


```
docker@default-online:~$ docker login registry.cn-hangzhou.aliyuncs.com
Username: sample@alibaba-inc.com
Password:
Login Succeeded
```

登录成功之后会显示 Login Succeeded。

另外，您还可以通过查看 config.json 文件，确认您的登录信息。

```
docker@default-online:~$ cat ~/.docker/config.json
{
"auths": {
"registry.cn-hangzhou.aliyuncs.com": {
"auth": "XXXXXXXXXXXXXXXXXXXXXXX"
}
}
}
```

docker pull

注意：

- 如果您要拉取 Docker 官方的镜像，参考下方相关链接中的加速器文档。
- 如果您要拉取公共仓库下的镜像，不登录 Registry 也是可以拉取的。

以容器服务的公共镜像 registry.cn-hangzhou.aliyuncs.com/acs/agent:0.8 为例。

```
docker@default-online:~$ docker pull registry.cn-hangzhou.aliyuncs.com/acs/agent:0.8
0.8: Pulling from acs/agent
5a026b6c4964: Already exists
e4b621e8d9cb: Already exists
8bc2fd04bdd4: Pull complete
a977b0087b3e: Pull complete
8f6e00ea13c6: Pull complete
875dd8c9666f: Pull complete
9c07bcabc35d: Pull complete
Digest: sha256:cac848bd31bccf2a041bda7b57e3051341093abde6859df9ee9d332dfec6ddd9
Status: Downloaded newer image for registry.cn-hangzhou.aliyuncs.com/acs/agent:0.8
```

您可以使用下边的命令查看下载下来的镜像（注意仓库坐标和 Tag）。

```
docker@default-online:~$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
registry.cn-hangzhou.aliyuncs.com/acs/agent 0.8 b9ba5841bdb0 24 hours ago 42.18 MB
```

docker push

镜像在本地环境构建或是打包好之后，就可以推到 Registry。

前提条件是，您有对这个仓库的读写权限或是读写授权。否则您会看到下面的报错信息。

```
docker@default-online:~$ docker push registry.cn-hangzhou.aliyuncs.com/acs/agent:0.8
The push refers to a repository [registry.cn-hangzhou.aliyuncs.com/acs/agent]
359f80267111: Layer already exists
7e5fa28d90b8: Layer already exists
b20d7f600f63: Layer already exists
4a159b4f8370: Layer already exists
7c3712ebe877: Layer already exists
d91d130a53aa: Layer already exists
fcad8ad5a40f: Layer already exists
unauthorized: authentication required
```

FAQ

docker login 失败

主要需要排查以下两种可能。

您使用了阿里云账户的登录密码，而不是 Registry 的独立登录密码。Registry 的登录密码是在 **镜像仓库管理控制台** 上设置与修改的。

您使用了 sudo 进行登录。使用 sudo 时，系统第一个要求输入的密码是 Linux 的用户密码。您可能在这里输入了 Registry 的登录密码，导致登录操作失败。

区分这个错误的方式很简单，Linux 的用户密码大多允许尝试三次，错误时会提示 try again。而 Registry 的登录密码错误一次之后就会退出，并返回以下错误。

```
Error response from daemon: Get https://registry.cn-hangzhou.aliyuncs.com/v2/: unauthorized: authentication required
```

docker pull 失败，提示 Error: image xxx not found

主要的排查步骤。

如果您下载的是公共仓库，那么问题应该为镜像地址不正确。请在 **镜像仓库管理控制台** 搜索一下这个公共仓库，检查一下想要下载的这个镜像版本是不是真实存在。

您想要下载一个私有仓库中的镜像，这时首先确认一下 Registry 的登录状态。运行下边的命令，可以看到所有登录的 Registry 域名。

```
cat ~/.docker/config.json
```

查看里面是不是包括您想要下载镜像的 Registry 域名。如果没有的话，您需要先进行登录操作。

如果显示已经登录的话，那么您需要确认您登录的这个账户是否有权限下载这个镜像。子账户默认没有任何权限，参见下方相关链接中主子账户授权的文档。

docker push 失败

主要的排查步骤和 docker pull 基本一致，仅仅是授权要求的级别较 pull 更高一些。

主子账号功能满足了企业账号分权管理的场景。子账号开通服务之后，主账号就可以对子账号进行授权。允许指定的子账号拥有镜像的推送、拉取权限，或是修改仓库信息等功能。

子账号开通服务

在进行各类授权之前，您需要确认子账号已经开通服务。

> **注意**：在子账号开通服务时，默认是没有任何仓库或命名空间授权的。

使用子账号登录 **镜像仓库管理控制台**。

填写 Docker 客户端的登录密码，即可完成服务的开通。

为子账号提供仓库授权

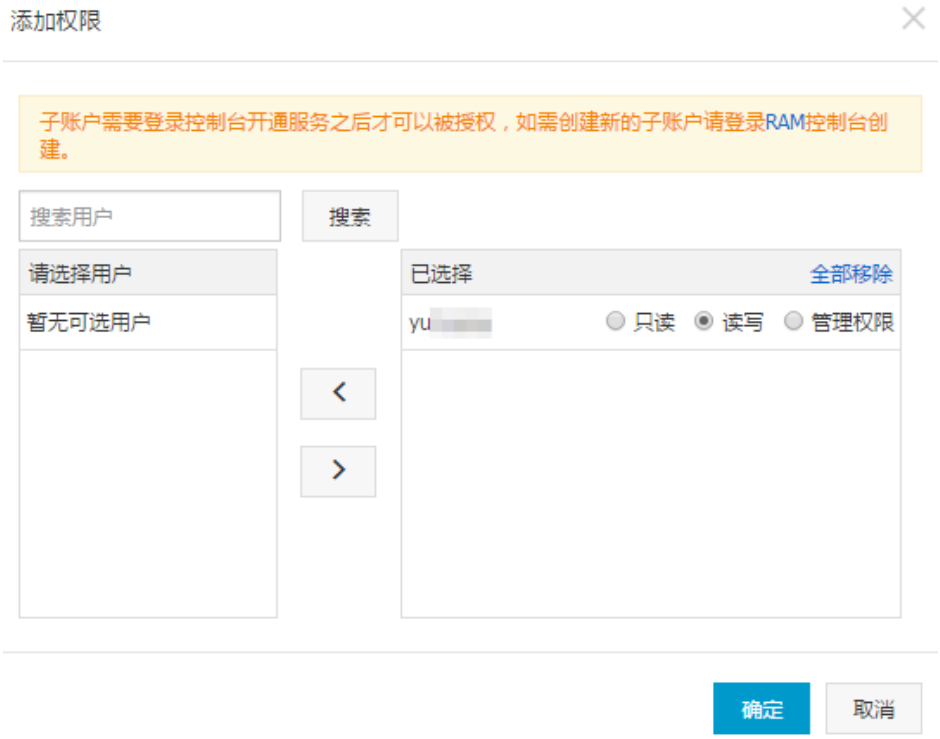
子账户开通服务之后，可以用主账号登录 **镜像仓库管理控制台** 对子账号进行授权。

使用主账号登录 **镜像仓库管理控制台**。

选择一个镜像仓库，单击右侧的 **管理**。

单击左侧导航栏中的 **仓库授权** > **添加授权**。

在弹出的对话框中，选择要授权的用户并为用户选择适当的权限，单击 **确定**。



您可以在仓库授权页面看到创建的授权信息，并可以对授权进行修改或删除。

使用子账号登录 **镜像仓库管理控制台**，您可以看到刚刚授权的镜像仓库。

为子账号提供命名空间级别的授权

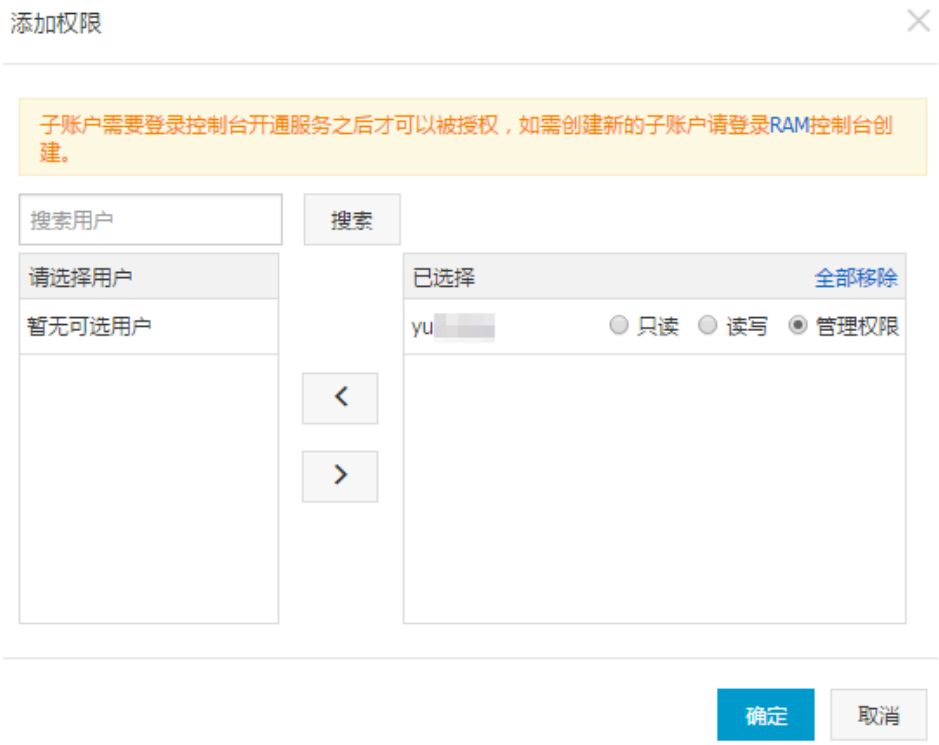
如果需要将整个命名空间的权限都交给一个子账号，那么可以在命名空间管理中为子账号进行授权。

使用主账号登录 **镜像仓库管理控制台**。

单击左侧导航栏中的 **Namespace管理**，选择一个命名空间并单击右侧的 **管理**。

单击 **添加授权**。

在弹出的对话框中，选择要授权的用户并为用户选择适当的权限，单击 **确定**。



您可以在命名空间授权页面看到创建的授权信息，并可以对授权进行修改或删除。

使用子账号登录 **镜像仓库管理控制台**，您可以看到刚刚授权的命名空间下所有的镜像仓库。

> **注意**：如果您同时对子账号进行了命名空间和其下镜像仓库的授权，授权以更高级别的授权为准。本示例中，命名空间的授权为管理，镜像仓库的授权为读写，则取高级别的权限：管理。

子账号新建命名空间

子账号想要独立使用服务的话，也可以选择新建一个命名空间自己使用。

这个命名空间还是主账号所有的资源。新建的命名空间会自动为子账号授予管理权限。一个主账号的命名空间上限是五个。

使用子账号登录 **镜像仓库管理控制台**。

单击左侧导航栏中的 **Namespace管理** 并单击 **创建namespace**。

日志

本文档介绍如何在容器服务里使用 ELK。

背景信息

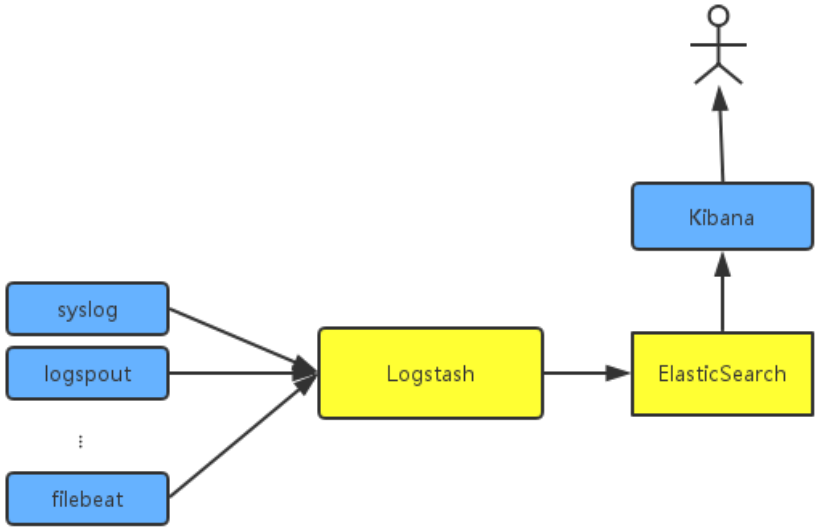
日志是 IT 系统的重要组成部分，记录了系统在什么时候发生了什么事情。我们可以根据日志排查系统故障，也可以做统计分析。

通常日志存放在本机的日志文件里，需要查看日志的时候，登录到机器上，用 grep 等工具过滤关键字。但是当应用要部署在多台机器上的时候，这种方式查看日志就很不方便了，为了找到一个特定的错误对应的日志，不得不登录到所有的机器上，逐个过滤文件。于是出现了集中式的日志存储方式：所有日志收集到日志服务里，在日志服务里可以查看和搜索日志。

在 Docker 环境里，集中式日志存储更加重要。相比传统的运维模式，Docker 通常使用编排系统管理容器，容器和宿主机之间的映射并不固定，容器也可能不断的在宿主机之间迁移，登录到机器上查看日志的方式完全没法用了，集中式日志成了唯一的选择。

容器服务集成了阿里云日志服务，通过声明的方式自动收集容器日志到日志服务。但是有些用户可能更喜欢用 ELK（Elasticsearch + Logstash + Kibana）这个组合。本文档介绍如何在容器服务里使用 ELK。

整体结构



我们要部署一个独立的 logstash 集群。logstash 比较重，很耗资源，所以不会在每台机器上都运行 logstash，更不要说每个 Docker 容器里。为了采集容器日志，我们会用到 syslog、logspout 和 filebeat，当然您还可能会用到其他的采集方式。

为了尽可能贴合实际场景，这里我们创建两个集群：一个名为 **testelk** 的集群用来部署 ELK，一个名为 **app** 的集群用于部署应用。

操作步骤

> **注意**：本文档中创建的集群和负载均衡均需位于同一地域下。

步骤 1 创建负载均衡实例

为了能让其他服务向 logstash 发送日志，我们需要在 logstash 前面配置负载均衡。

创建应用前，登录 **负载均衡管理控制台**。

创建一个 **公网类型** 的负载均衡实例。

设置两条监听规则。其中一条设置前端和后端的端口映射 5000：5000 ，另一条设置端口映射 5044：5044，不用添加后端服务器。

步骤 2 部署 ELK

登录 **容器服务管理控制台**，创建集群 **testelk**。

有关如何创建集群，参见 **创建集群**。

> **注意**：集群必须和上边创建的负载均衡实例位于同一地域。

为集群绑定上边所创建的负载均衡实例。

在集群列表页面，选择集群 **testelk** > 单击右侧的 **管理** > 单击左侧导航栏中的 **负载均衡** > 单击 **绑定SLB** > 选择上边创建的负载均衡实例并单击 **确定**。

使用下面的编排模板部署 ELK。本示例创建了一个名为 **elk** 的应用。

有关如何使用编排模板创建应用，参见 **创建应用**。

> **注意**：您需要使用您上边所创建的负载均衡实例的 ID 替换编排文件中的 ${SLB_ID}。

```
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://${SLB_ID}:5044' #先创建slb
aliyun.lb.port_5000: 'tcp://${SLB_ID}:5000'
links:
- elasticsearch
```

在这个编排文件里，Elasticsearch 和 Kibana 我们直接使用了官方镜像，没有做任何更改。logstash 需要配置文件，需要自己做一个镜像，把配置文件放进去。镜像源码参见 demo-logstash 。

logstash 的配置文件如下。这是一个非常简单的 logstash 配置，我们提供了 syslog 和 filebeats 两种输入格式，对外的端口分别是 5044 和 5000。

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

tcp {
port => 5000
type => syslog
}

}

filter {
}

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

stdout { codec => rubydebug }
}
```

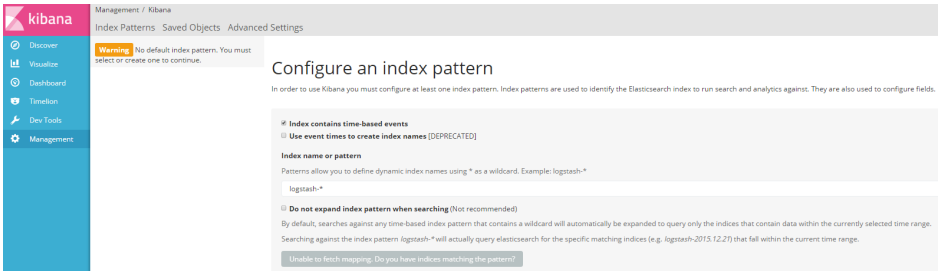
配置 kibana index。

访问 kibana。

URL 可以在应用的路由列表（单击所创建的应用的名称 **elk** > 单击 **路由列表** 页签 > 单击路由地址）里找到。

创建 index。

根据您的实际需求进行配置并单击 **Create**。



步骤 3 收集日志

Docker 里标准的日志方式是用 Stdout，所以我们先演示如何把 Stdout 收集到 ELK。如果您在使用文件日志，可以直接用 filebeat。我们用 wordpress 作为演示用的例子，下面是 wordpress 的编排模板。我们在另外一个集群中创建应用 **wordpress**。

登录 **容器服务管理控制台**，创建集群 **app**。

有关如何创建集群，参见 **创建集群**。

> **注意**：集群必须和上边创建的负载均衡实例位于同一地域。

使用下面的编排模板创建应用 **wordpress**。

> **注意**：您需要使用您上边所创建的负载均衡实例的 IP 替换编排文件中的 ${SLB_IP}。

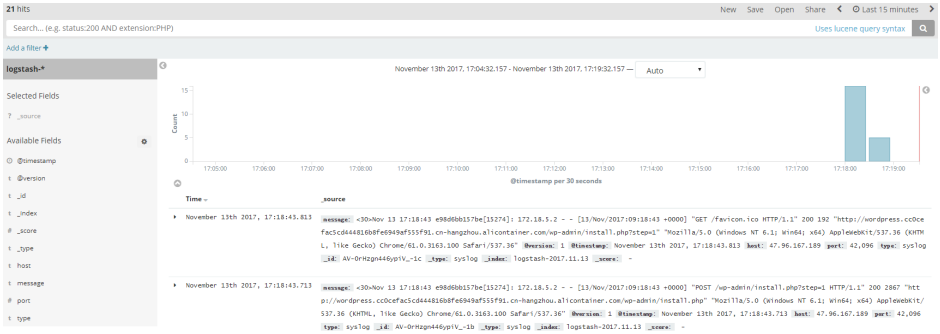
```
version: '2'
services:
```

```
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'
```

待部署成功后，找到 wordpress 的访问地址（单击所创建的 **wordpress** 应用的名称 > 单击 **路由列表** 页签 > 单击路由地址），即可访问 wordpress 应用。

在应用列表页面，单击应用 **elk** > 单击 **路由列表** 页签 > 单击路由地址。

成功访问 kibana 的页面，可以查看已经收集到的日志。



本文档介绍一款新的 Docker 日志收集工具：fluentd-pilot。fluentd-pilot 是我们为您提供的日志收集镜像。您可以在每台机器上部署一个 fluentd-pilot 实例，就可以收集机器上所有 Docker 应用日志。

fluentd-pilot 具有如下特性：

- 一个单独的 fluentd 进程收集机器上所有容器的日志。不需要为每个容器启动一个 fluentd 进程。
- 支持文件日志和 stdout。docker log dirver 亦或 logspout 只能处理 stdout，fluentd-pilot 不仅支持收集 stdout 日志，还可以收集文件日志。
- 声明式配置。当您的容器有日志要收集，只要通过 label 声明要收集的日志文件的路径，无需改动其他任何配置，fluentd-pilot 就会自动收集新容器的日志。
- 支持多种日志存储方式。无论是强大的阿里云日志服务，还是比较流行的 elasticsearch 组合，甚至是 graylog，fluentd-pilot 都能把日志投递到正确的地点。
- 开源。fluentd-pilot 完全开源，您可以从 这里 下载代码。如果现有的功能不能满足您的需要，欢迎

提 issue。

快速启动

下面演示一个最简单的场景：先启动一个 fluentd-pilot，再启动一个 tomcat 容器，让 fluentd-pilot 收集 tomcat 的日志。为了简单起见，这里先不涉及阿里云日志服务或者 ELK。如果您想在本地运行，只需要有一台运行 Docker 的机器就可以了。

首先启动 fluentd-pilot。

> **注意**：以这种方式启动，由于没有配置后端使用的日志存储，所有收集到的日志都会直接输出到控制台，所以主要用于调试。

打开终端，输入命令：

```
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
```

您会看到 fluentd-pilot 的启动日志。

```
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

DEBU[0000] ad93dbee9691cc6a1ed1f9fab9ee365774d2f43262870425633940547 5de3515 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>

DEBU[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/424a96798c3255402168df54806e1055884a8140389275703917 9aff176547f/_data /usr/local/tomcat/logs none_localhost_access_log.*.txt map[]}]
```

不要关闭终端。新开一个终端启动 tomcat。tomcat 镜像属于少数同时使用了 stdout 和文件日志的 Docker 镜像，非常适合这里的演示。

```
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
```

说明：

- aliyun.logs.catalina=stdout 告诉 fluentd-pilot 这个容器要收集 stdout 日志。
- aliyun.logs.access=/usr/local/tomcat/logs/localhost_access_log.*.txt 则表示要收集容器内 /usr/local/tomcat/logs/ 目录下所有名字匹配 localhost_access_log.*.txt 的文件日志。后面会详细介绍 label 的用法。

> **注意**：如果您在本地部署 tomcat，而不是在阿里云容器服务上，您需要指定 -v

/usr/local/tomcat/logs；否则 fluentd-pilot 没法读取到日志文件。容器服务已经自动做了优化，不需自己加 -v 了。

fluentd-pilot 会监控 Docker 容器事件，当发现带有 aliyun.logs.xxx 容器时，自动解析容器配置，并且开始收集对应的日志。启动 tomcat 之后，您会发现 fluentd-pilot 的终端立即输出了一大堆的内容，其中包含 tomcat 启动时输出的 stdout 日志，也包括 fluentd-pilot 自己输出的一些调试信息。

您可以打开浏览器访问刚刚部署的 tomcat，您会发现每次刷新浏览器，在 fluentd-pilot 的终端里都能看到类似的记录。其中 message 后面的内容就是从 /usr/local/tomcat/logs/localhost_access_log.XXX.txt 里收集到的日志。

使用 ElasticSearch + kibana

首先要部署一套 ElastichSearch + kibana，您可以参考 **容器服务中使用 ELK** 在阿里云容器服务里部署 ELK，或者按照 ElasticSearch/kibana 的文档直接在机器上部署。本文档假设您已经部署好了这两个组件。

如果您还在运行刚才启动的 fluentd-pilot，先关掉，然后使用下面的命令启动。

注意：执行之前，先把 ELASTICSEARCH_HOST 和 ELASTICSEARCH_PORT 两个变量替换成您实际使用的值。ELASTICSEARCH_PORT 一般为 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
```

相比前面启动 fluentd-pilot 的方式，这里增加了三个环境变量：

- FLUENTD_OUTPUT=elasticsearch：把日志发送到 ElasticSearch。
- ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST}：ElasticSearch 的域名。
- ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT}：ElasticSearch 的端口号。

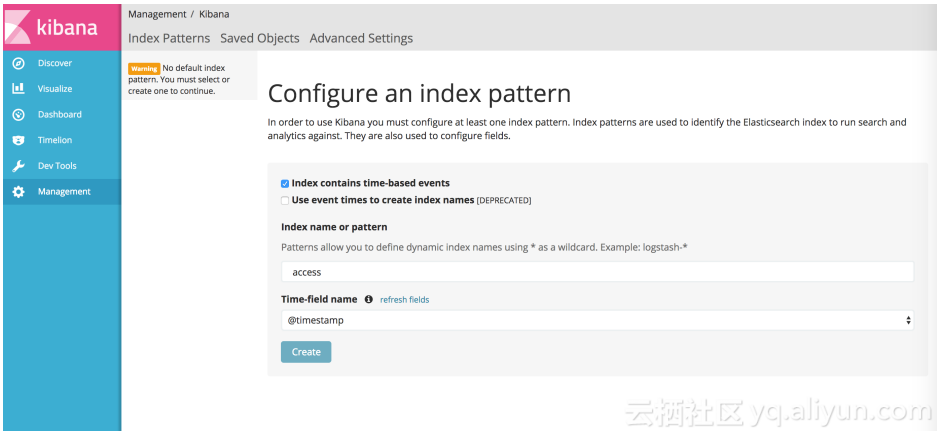
继续运行前面的 tomcat，再次访问，让 tomcat 产生一些日志，所有这些新产生的日志都将发送到 ElasticSearch 里。

打开 kibana，此时您还看不到新日志，需要先创建 index。fluentd-pilot 会把日志写到 ElasticSearch 特定的 index下，规则如下：

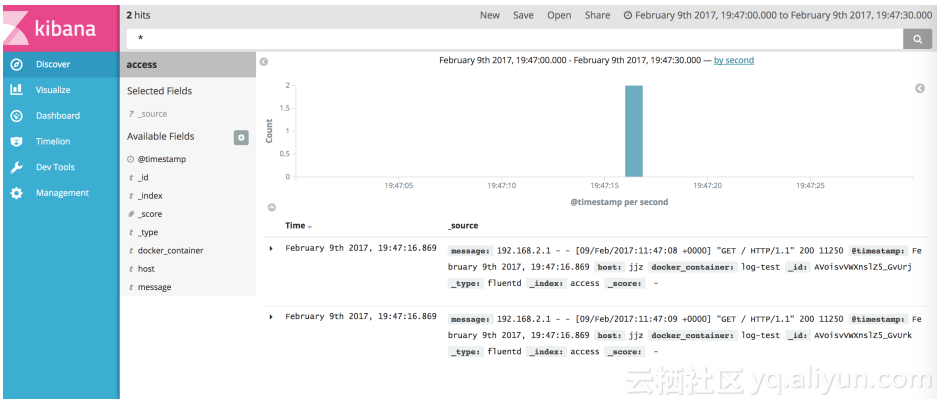
如果应用上使用了标签 aliyun.logs.tags，并且 tags 里包含 target，使用 target 作为 ElasticSearch 里的

index。否则，使用标签 aliyun.logs.XXX 里的 XXX 作为 index。

在前面 tomcat 里的例子里，没有使用 aliyun.logs.tags 标签，所以默认使用了 access 和 catalina 作为 index。我们先创建 index access。



创建好 index 就可以查看日志了。



在阿里云容器服务里使用 fluentd-pilot

容器服务专门为 fluentd-pilot 做了优化，最适合 fluentd-pilot 运行。

要在容器服务里运行 fluentd-pilot，您仅需要使用下面的编排文件创建一个新应用。有关如何创建应用，参见创建应用。

```
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 #按照您的需要替换
ELASTICSEARCH_HOST: ${elasticsearch} #按照您的需要替换
ELASTICSEARCH_PORT: 9200
labels:
aliyun.global: true
```

接下来，您就可以在要收集日志的应用上使用 aliyun.logs.xxx 标签了。

label 说明

启动 tomcat 时，声明了下面两个 label 来告诉 fluentd-pilot 这个容器的日志位置。

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

您还可以在应用容器上添加更多的标签。

aliyun.logs.$name = $path

- 变量 name 是日志名称，只能包含 0~9、a~z、A~Z 和连字符（-）。
- 变量 path 是要收集的日志路径，必须具体到文件，不能只写目录。文件名部分可以使用通配符，例如，/var/log/he.log 和 /var/log/*.log 都是正确的值，但 /var/log 不行，不能只写到目录。stdout 是一个特殊值，表示标准输出。

aliyun.logs.$name.format：日志格式，目前支持以下格式。

- none：无格式纯文本。
- json：json 格式，每行一个完整的 json 字符串。
- csv：csv 格式。

aliyun.logs.$name.tags：上报日志时，额外增加的字段，格式为 k1=v1,k2=v2，每个 key-value 之间使用逗号分隔，例如 aliyun.logs.access.tags="name=hello,stage=test"，上报到存储的日志里就会出现 name 字段和 stage 字段。

如果使用 ElasticSearch 作为日志存储，target 这个 tag 具有特殊含义，表示 ElasticSearch 里对应的 index。

扩展 fluentd-pilot

对于大部分用户来说，fluentd-pilot 现有功能足以满足需求，如果遇到没法满足的场景，您可以：

- 到 https://github.com/AliyunContainerService/fluentd-pilot 提交 issue。
- 直接改代码，再提 PR。

Kubernetes 在版本 1.6 后正式加入了 Nvidia GPU 的调度功能，支持在 Kubernetes 上运行和管理基于 GPU 的应用。而在 2017 年 9 月 12 日，阿里云发布了新的异构计算类型 GN5，基于 P100 nvidia GPU, 提供灵活强悍的异构计算模型，从基础设施到部署环境全面升级，可有效提升矩阵运算、视频识别、机器学习、搜索排序等处理计算效率。

在阿里云的 GN5 上部署一套支持 GPU 的Kubernetes集群是非常简单的，利用 ROS 模板一键部署，将阿里云强大的计算能力便捷的输送到您的手中。不出 10 分钟，您就可以开始在阿里云的 Kubernetes 集群上开始您的 Kubernetes+GPU+TensorFlow 的深度学习之旅了。

前提准备

您需要开通容器服务、资源编排（ROS）服务和访问控制（RAM）服务。 登录 **容器服务管理控制台**、**ROS 管理控制台** 和 **RAM 管理控制台** 开通相应的服务。

所创建的资源均为按量付费，根据阿里云的计费要求，请确保您的现金账户余额不少于 100 元。

目前，按量付费的异构计算 GN5 需要申请工单开通，请登录阿里云账号后，按照如下内容提交 **ECS 工单**。

我需要申请按量付费的GPU计算型gn5，请帮忙开通，谢谢。

当审批通过后，您就可以在 **ECS控制台** 实例列表页面中，单击 **创建实例**>选择 **按量付费** 页签>单击 **选择其他实例规格** 查看 GPU 节点是否可用。

使用限制

目前仅支持华北2（北京）、华东2（上海）和华南1（深圳）创建 Kubernetes 的 GPU 集群。

集群部署

在本文中， 我们提供了部署单 Master 节点，并可以配置 worker 节点数的部署方式，同时可以按需扩容和缩容，创建和销毁集群也非常简单。

选择 ROS 创建入口。

创建一个位于华北2的GPU Kubernetes集群

创建一个位于华东2的GPU Kubernetes集群

创建一个位于华南1的GPU Kubernetes集群

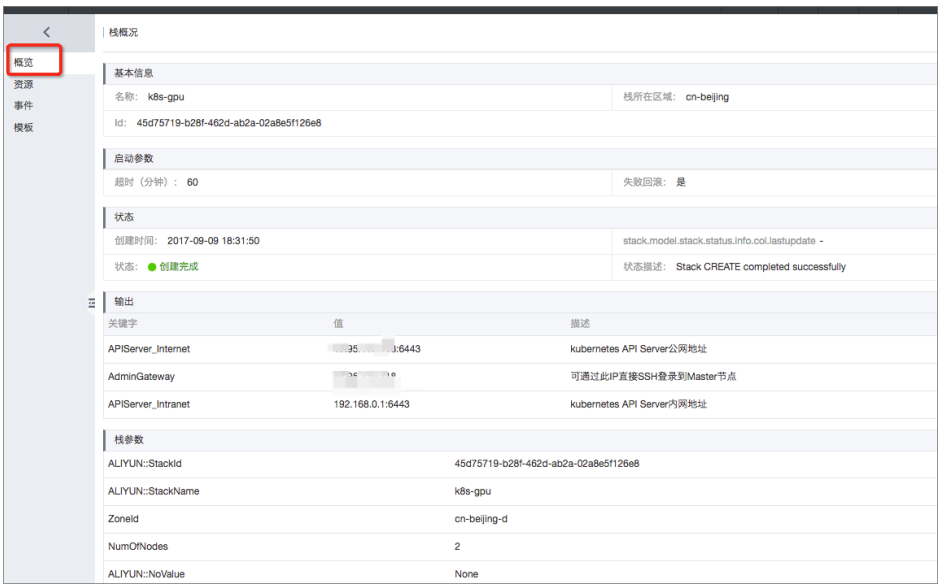
填写参数并单击 **创建**。

- **栈名**：所部署的 Kubernetes 集群属于一个 ROS 的栈，栈名称在同一个地域内不能重复。
- **创建超时**：整个部署过程的超时时间，默认为 60 分钟，无需修改。
- **失败回滚**：选择 失败回滚 时，如果部署过程中发生不可自动修复性错误，将删除所有已创建资源；反之，已创建资源将被保留，以便进行问题排查。
- **Master节点ECS实例规格**：指定 Master 节点所运行的 ECS 实例的规格，默认为 ecs.n4.large。
- **Worker节点ECS实例规格**：指定 Worker 节点所运行的包含GPU的 ECS 实例规格，默认为 ecs.gn5-c4g1.xlarge。具体配置可以查看ECS规格文档。
- **部署GPU节点的可用区**：指定 GPU节点可以部署的可用区，请根据具体地域选择。
- **ECS系统镜像**：目前指定 centos_7。
- **Worker节点数**：指定 Worker 节点数，默认为 2，支持后期扩容。
- **ECS登录密码**：所创建的 ECS 实例可通过此密码登录，请务必牢记密码。

单击 **创建**， 启动部署。

这样，部署请求已经成功提交。 可以单击 **进入事件列表** 实时监控部署过程。

点击概览查看，部署完成后的输出结果。



通过输出结果中返回的信息，可以对 Kubernetes 集群进行管理：

- APIServer_Internet：Kubernetes 的 API server 对公网提供服务的地址和端口，可以通过此服务在用户终端使用 kubectl 等工具管理集群。
- AdminGateway：可以直接通过 SSH 登录到 Master 节点，以便对集群进行日常维护。
- APIServer_Intranet：Kubernetes 的 API server 对集群内部提供服务的地址和端口。

通过 kubectl 连接 Kubernetes 集群 ，参见通过 kubectl 连接 Kubernetes 集群，并且通过命令查看 GPU 节点。

```
 kubectl describe  node {node-name}
Name: cn-beijing.i-{name}
Role:
...
Addresses:
InternalIP: 192.168.2.74
Capacity:
alpha.kubernetes.io/nvidia-gpu: 1
cpu: 4
memory: 30717616Ki
pods: 110
Allocatable:
alpha.kubernetes.io/nvidia-gpu: 1
cpu: 4
memory: 30615216Ki
pods: 110
...
```

部署GPU应用

最后我们部署一个基于 GPU 的 TensorFlow Jupyter 应用来做一下简单的测试，以下为我们的 Jupyter 的部署配置文件 jupyter.yml。

```yaml
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: jupyter
spec:
replicas: 1
template:
metadata:
labels:
k8s-app: jupyter
spec:
containers:
- name: jupyter
image: registry-vpc.cn-beijing.aliyuncs.com/tensorflow-samples/jupyter:1.1.0-devel-gpu
imagePullPolicy: IfNotPresent
env:
- name: PASSWORD
value: mypassw0rd
resources:
limits:
alpha.kubernetes.io/nvidia-gpu: 1
volumeMounts:
- mountPath: /usr/local/nvidia
name: nvidia
volumes:
- hostPath:
path: /var/lib/nvidia-docker/volumes/nvidia_driver/375.39
name: nvidia

---
apiVersion: v1
kind: Service
metadata:
name: jupyter-svc
spec:
ports:
- port: 80
targetPort: 8888
name: jupyter
selector:
k8s-app: jupyter
type: LoadBalancer
```

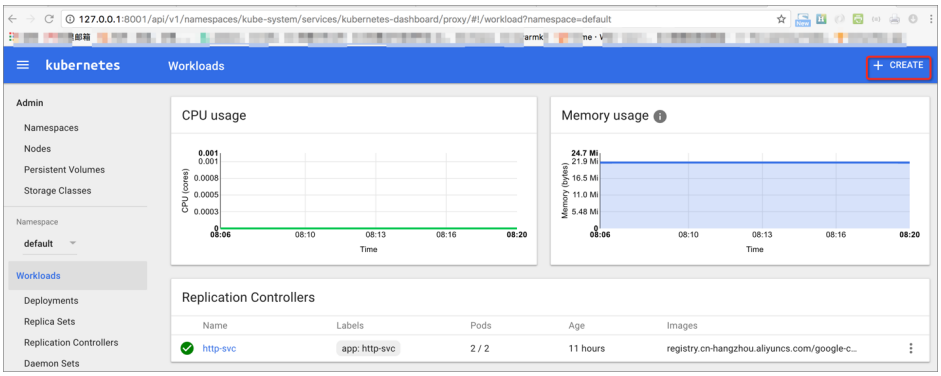
Deployment配置：

- alpha.kubernetes.io/nvidia-gpu 指定调用 Nvidia GPU 的数量。
- type=LoadBalancer 指定使用阿里云的负载均衡访问内部服务和负载均衡。
- 为了能让 GPU 容器运行起来，需要将 Nvidia 驱动和 CUDA 库文件指定到容器中。这里需要使用 hostPath，在阿里云上您只需要将 hostPath 指定到 /var/lib/nvidia-docker/volumes/nvidia_driver/375.39 即可，并不需要指定多个 bin 和 lib 目录。

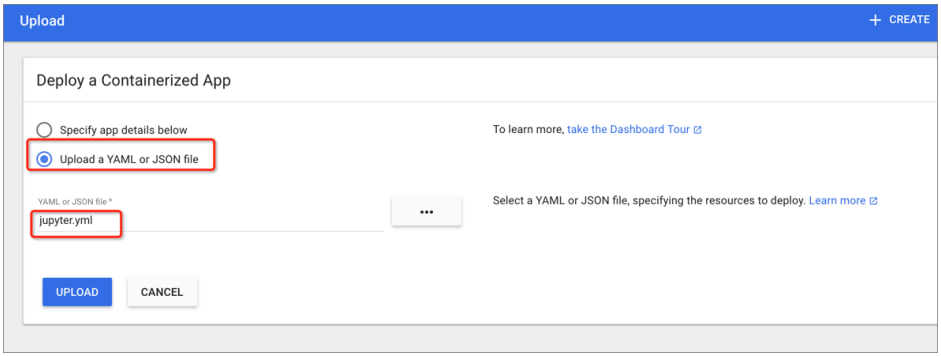
- 环境变量 PASSWORD 指定了访问 Jupyter 服务的密码，您可以按照您的需要修改。

创建应用

按照 通过 Web UI 管理应用 介绍的方式连接 Kubernetes Web UI， 点击 CREATE 创建应用



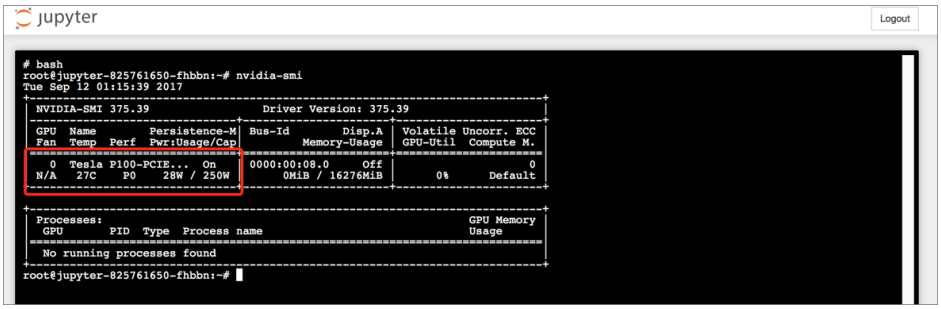
单击 Upload a YAML or JSON file。选择刚才创建的 jupyter.yml 文件



待部署成功后， 在 Kubernetes Web UI 上定位到 default 命名空间，选择 Services。

可以看到刚刚创建的 jupyter-svc 的 jupyter 服务的外部负载均衡地址(External endpoints)。

点击外部负载均衡的地址，您就可以直接访问到 Jupyter 服务， 通过 web Terminal 内执行 nvidia-smi 命令查看容器内 GPU 设备状况。我们可以看到当前的容器里已经分配了一块 Tesla P100 的 GPU 卡。



直接重启节点可能会导致集群出现异常。比如，对于 swarm mode 集群内的 Manager 节点，如果 Manager 健康节点数小于 2，则可能会导致集群无法自愈，最终导致集群不可用。本文结合阿里云历史案例经验，说明了在对容器服务进行主动运维等场景下，需要重启节点时的操作最佳实践。

检查业务高可用配置

在重启容器服务节点前，建议先检查或修正如下业务配置，以避免节点重启触发单点异常，进而导致业务可用性受损。

配置数据持久化策略

建议为日志、业务配置等重要数据配置的外部卷进行数据持久化，以避免容器重建后，原有容器被删除引发数据丢失。

关于容器服务数据卷的使用，参见 **数据卷管理**。

配置重启策略

建议为相应业务服务配置 restart: always 自重启策略，以便节点重启后，相应容器能自动拉起。

配置高可用策略

建议结合产品架构，为相应业务配置 **可用区调度**（availability:az 属性）、**指定节点调度**（affinity、constraint 属性) 和 **指定多节点调度**（constraint 属性） 等亲和性、互斥性策略，以避免由于相应节点重启引发单点异常。比如，对于数据库业务，建议主备或多实例部署，然后结合前述特性，确保不同实例落在不同节点上，并且相关节点不会同时重启。

操作最佳实践

建议首先参阅前述说明，检查业务高可用性配置。然后 **在每个节点上（切忌同时对多个节点进行操作）**，依次按如下步骤操作：

快照备份

建议先对节点所有关联磁盘创建最新快照进行备份，以避免由于长时间未重启服务器，导致节点关机后启动过程中出现异常导致业务可用性受损。

验证业务容器配置有效性（swarm mode 集群忽略）

对于非 swarm mode 集群，重启节点上的相应业务容器，确保容器能正常被重新拉起。

> **注意**：Swarm mode 集群的最小控制操作单元是服务。所以，不能直接在 swarm mode 集群节点上通过 docker start/stop 等操作直接处理业务容器，否则会引发相关报错。正确的做法是在 容器服务管理控制台 通过重新 调整应用的 REPLICAS 的方式来对业务做自动调整。

修改节点角色（适用于 swarm mode 集群）

如果相应节点是 swarm mode 集群内的 Manager 节点，则先将其 设置为 Worker 节点。

验证 Docker Engine 运行有效性

尝试重启 Docker daemon ，确保 Docker enginge 能正常重新启动。

执行相关运维操作

执行计划内的相关运维操作，比如业务代码更新、系统补丁安装、系统配置调整等。

重启节点

在控制台或系统内部，正常重启节点。

重启后状态检查

重启完节点后，到 容器服务管理控制台，检查节点健康状态，检查业务容器运行状态。

回调节点角色（适用于 swarm mode 集群）

如果相应节点是 swarm mode 集群内的 Manager 节点，则需要将其重新 设置为 Manager 节点。