阿里云
容器服务

最佳实践

文档版本：20190321

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 容器服务swarm集群与Kubernetes集群的主要功能比对

1.1 概述

本文将介绍容器服务swarm集群与Kubernetes集群主要功能比对的前提条件及使用限制。

前提条件

您已经成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。

> 说明:
>
> 。
>
> - 目前容器服务Kubernetes版支持四种集群：经典集群、Kubernetes托管版、多可用区Kubernetes及Serverless Kubernetes（公测）。
> - 本文以创建Kubernetes集群为例，进行容器服务Swarm集群与Kubernetes集群的功能比对。

使用限制

- 本文主要介绍以下两种场景的功能比对：
  - 应用均为无状态应用。
  - 应用的数据存在数据库或存储中。

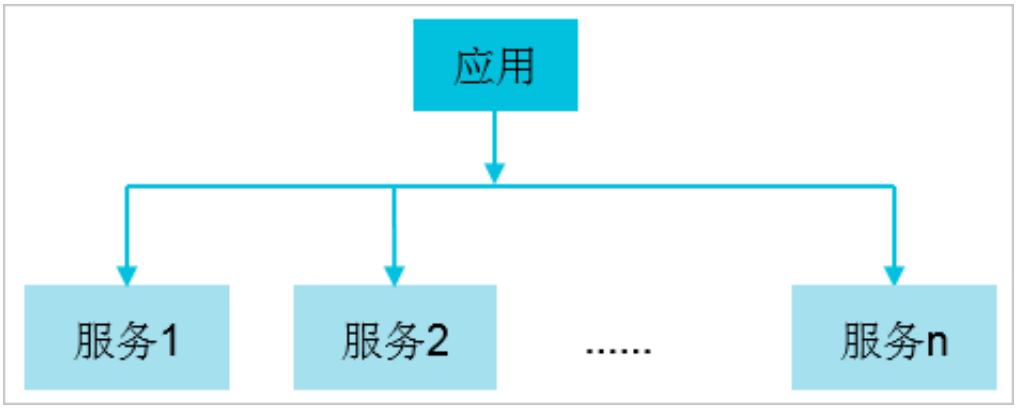
1.2 概念比对

本文主要介绍容器服务Swarm集群与Kubernetes集群主要概念的比对。

应用

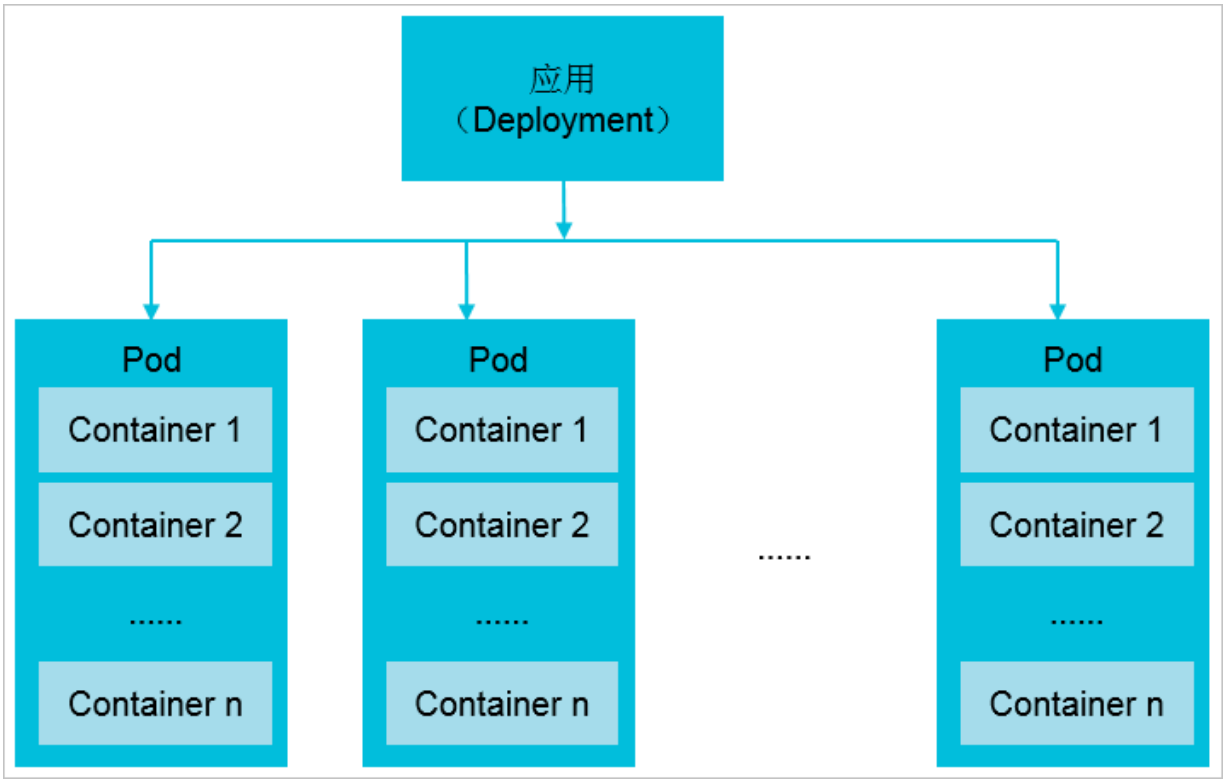
容器服务Swarm集群

容器服务Swarm中，应用类似于项目，一个应用下面可以有多个服务。服务是具体提供应用功能的实例。服务可以水平扩展。



容器服务Kubernetes集群

容器服务Kubernetes中，应用是指部署（deployment），能够提供应用对外暴露的功能。部署中会有Pod和container，但Kubernetes中最小的调度单位是一个pod，其中Pod可包含多个container。一个Pod可以认为是应用的一个实例，多实例（多Pod）可以调度到不同的节点上，也就是说Pod可以水平扩展。



说明:
虽然上图中一个Pod中有多个container，但是在实际使用时，建议一个Pod对应一个container，这里对应多个container是为了说明Pod的能力。

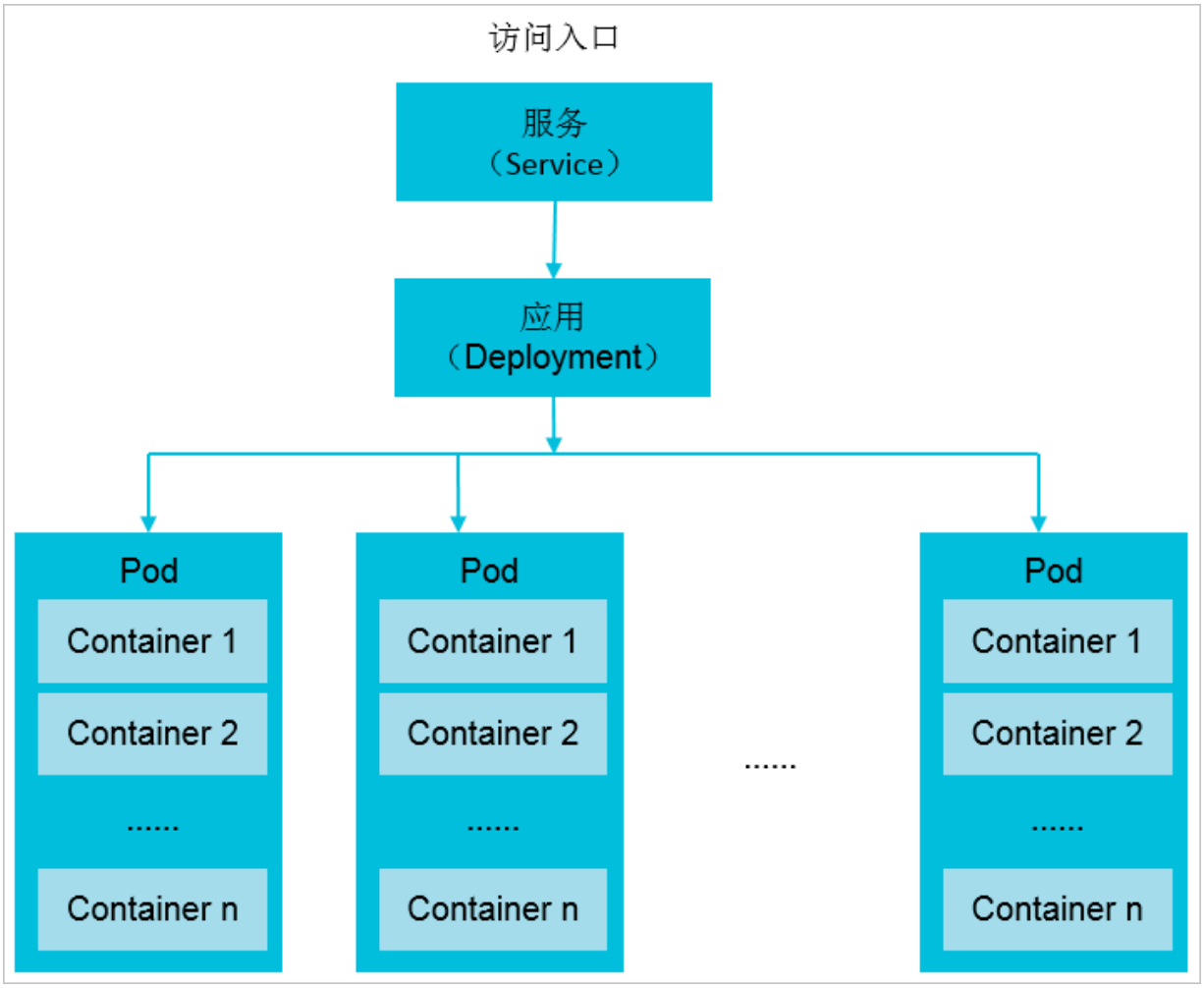
服务

容器服务Swarm集群

容器服务Swarm中的服务即提供应用功能的具体实例。当在Swarm集群中创建一个应用的时候，服务的访问方式会直接暴露给外部。

容器服务Kubernetes集群

容器服务Kubernetes中的服务是一个抽象概念，通过服务（Service）可以将应用（Deployment）的访问方式暴露给外部。



应用访问

容器服务Swarm集群

容器服务Swarm在部署应用的时候，可以选择三种应用访问方式，无论哪一种方式都可以直接暴露应用访问，不需要额外的操作：

- <HostIP>:<port>
- 简单路由
- 负载均衡

容器服务Kubernetes集群

容器服务Kubernetes在创建应用（即部署deployment）后，不能直接暴露应用的访问方式，需要通过创建服务（Service）进行应用访问的暴露。在容器服务Kubernetes集群内部应用之间可以通过服务名（Service Name）进行访问，服务名也只能用于集群内部访问。若要在集群外部访问应用，需要创建NodePort类型和LoadBalancer类型的服务进行对外暴露：

- ClusterIP（集群内部访问使用，也可以使用服务名）
- NodePort（类似于Swarm集群的<HostIP>:<port>）
- LoadBalancer（类似于Swarm集群的负载均衡）
- 域名，通过创建路由（ingress）来实现（类似于Swarm集群的简单路由）

1.3 使用镜像创建应用-基本配置比对

本文介绍容器服务Swarm集群与Kubernetes集群使用镜像创建应用时，基本配置的比对。

使用镜像创建应用

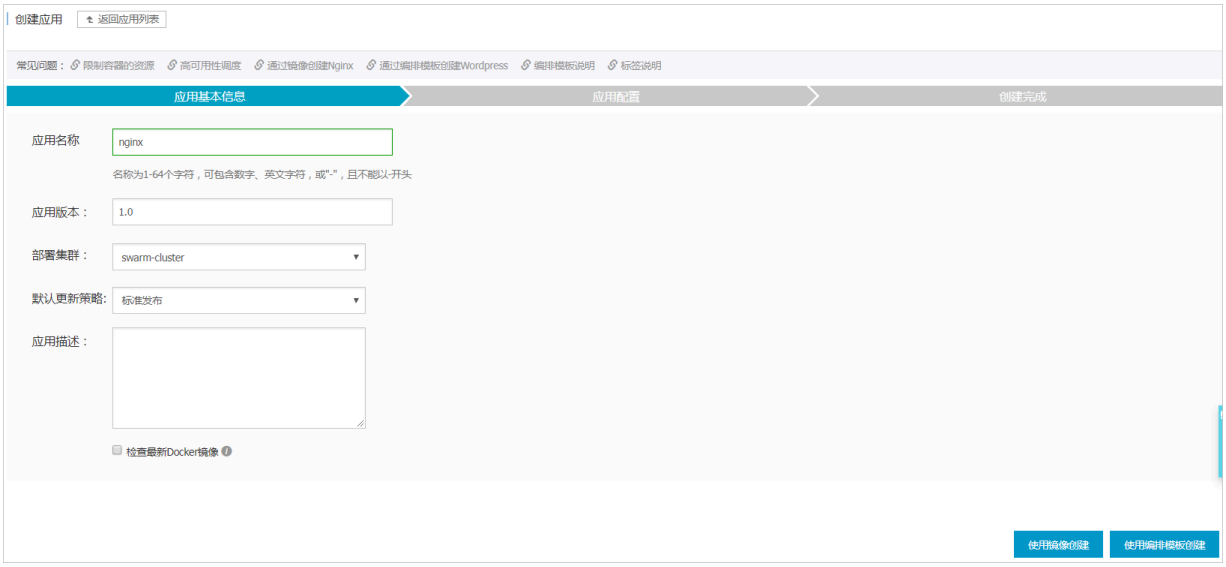
容器服务Swarm与Kubernetes集群在使用镜像创建应用时，部署界面差异较大。

- 容器服务Swarm集群，请参考创建应用。
- 容器服务Kubernetes集群，请参考使用镜像创建无状态Deployment应用。

应用基本信息

容器服务Swarm集群

在应用基本信息中，部署的内容包括应用名称、应用版本、部署集群、默认更新策略及应用描述。



容器服务Kubernetes集群

与Swarm集群不同的是：需要配置命名空间、副本数量及类型。

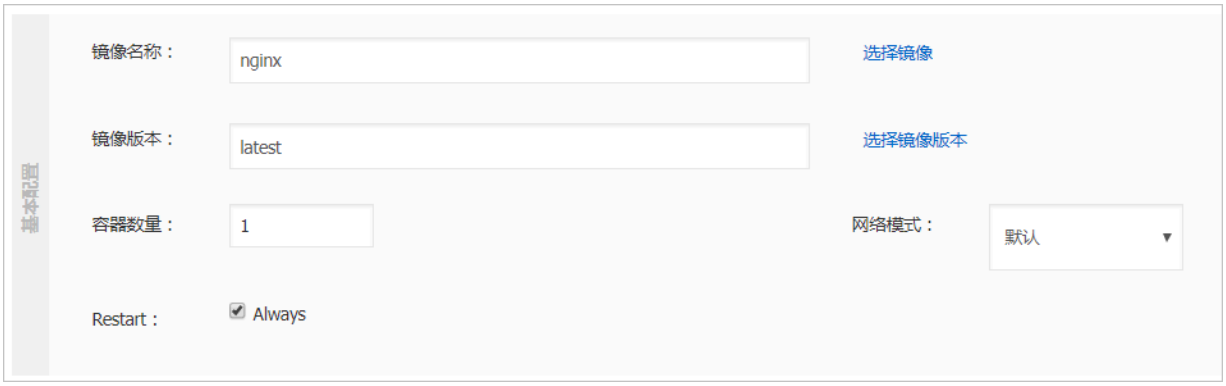
命名空间是容器服务Kubernetes集群特有的概念，Kubernetes通过命名空间（namespace）进行资源的隔离，如CPU等。同时也可以区分不同的使用环境，比如测试环境、开发环境。如果涉及生产环境，建议通过集群隔离。Kubernetes的命名空间，可参考基本概念。

基本配置

基本配置，主要是选择镜像和镜像版本。

容器服务Swarm集群

网络模式目前支持默认和host两种。



容器服务Kubernetes集群

- 网络模式，在创建集群时已经选定，可选择两种网络插件Flannel和Terway，可参考如何选择Kubernetes集群网络插件#Terway和Flannel。
- 所需资源是声明需要的CPU和内存资源，资源限制是实际使用中不能超过的资源上限。类似于容器服务Swarm集群的容器配置部分的CPU限制和内存限制。

1.4 使用镜像创建应用-网络配置比对

本文介绍容器服务Swarm集群与Kubernetes集群使用镜像创建应用时，网络配置的比对。

使用镜像创建应用

容器服务Swarm与Kubernetes集群在使用镜像创建应用时，部署界面差异较大。

- 容器服务Swarm集群，请参考创建应用。
- 容器服务Kubernetes集群，请参考使用镜像创建无状态Deployment应用。

网络配置

容器服务Swarm集群的网络配置主要完成应用对外访问方式的暴露。

端口映射

容器服务Swarm集群

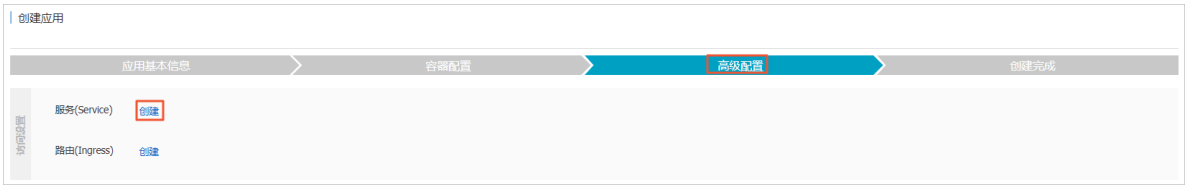
容器服务Swarm集群的端口映射是将应用端口映射到宿主机，在每个宿主机上都会启用相同的端口，这样访问应用的时候只需要`<HostIP>:<Port>`即可访问。

容器服务Kubernetes集群

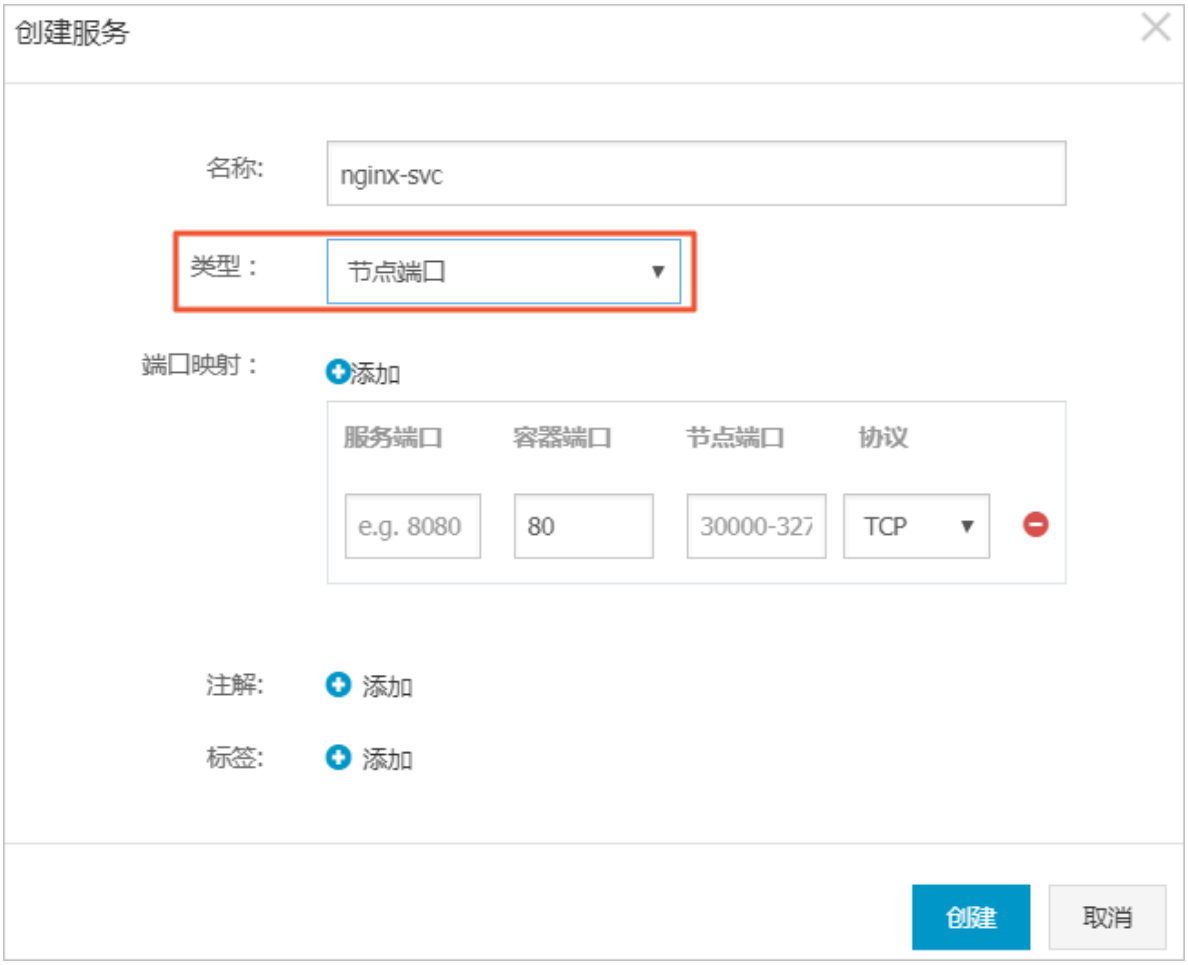
在容器服务Kubernetes集群可以通过NodePort类型的Service来实现，有以下两种方法：

方法一：创建应用时配置

1. 部署完容器配置后，进行高级配置时，在访问设置区域，单击服务（Service）右侧的创建。



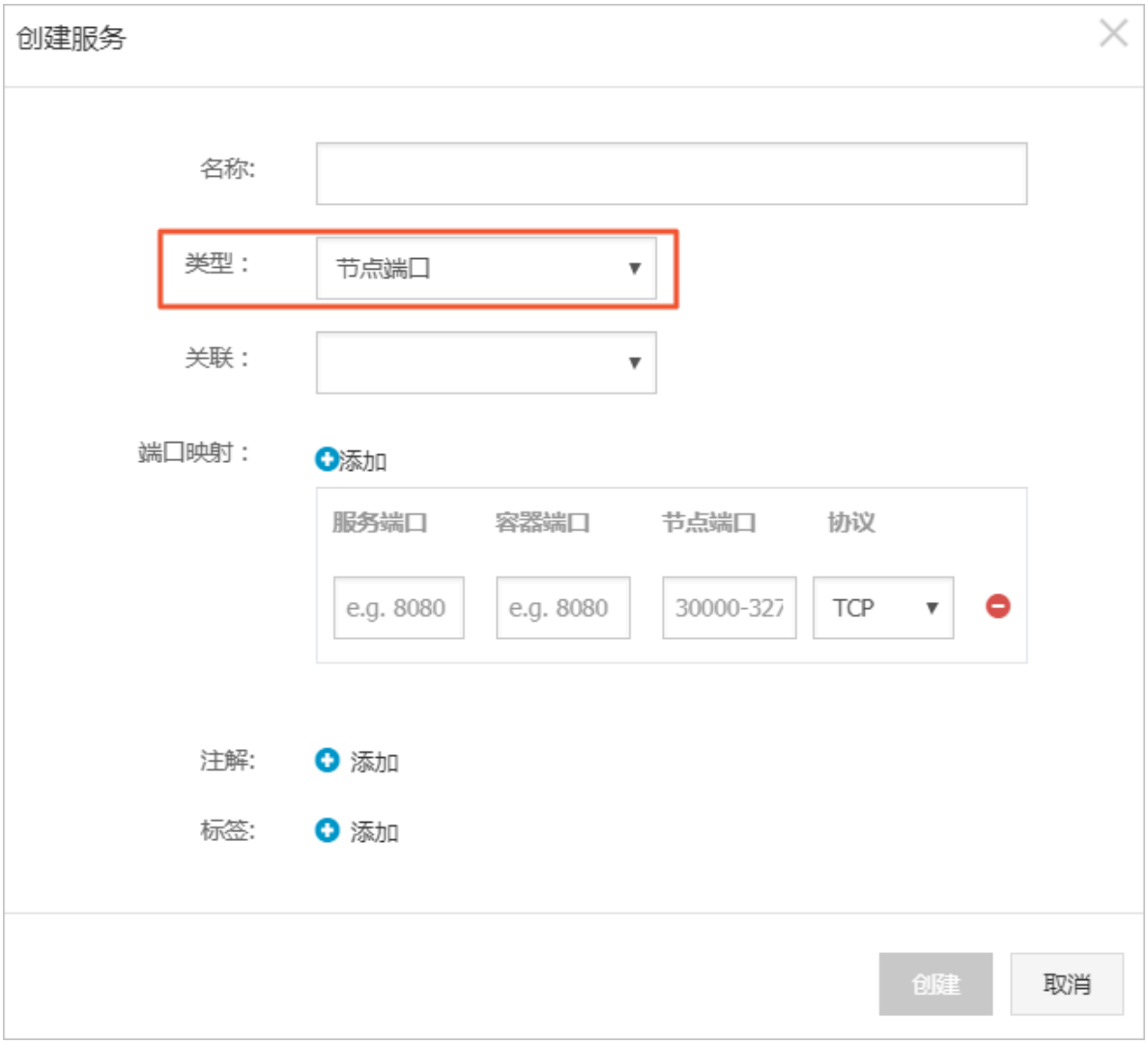
2. 类型请选择节点端口。详细信息可参考使用镜像创建无状态Deployment应用。



方法二：通过创建服务配置

1. 在容器管理控制台，单击左侧导航栏路由与负载均衡 > 服务，进入服务（Service）页面。

2. 选择目标集群和命名空间，单击创建，在创建服务页面，类型选节点端口。详细信息可参考创建服务。



简单路由配置

容器服务Swarm集群

容器服务Swarm集群的简单路由配置为用户提供了通过域名访问应用的方式，用户可以选择使用容器服务提供的域名也可以自定义域名。

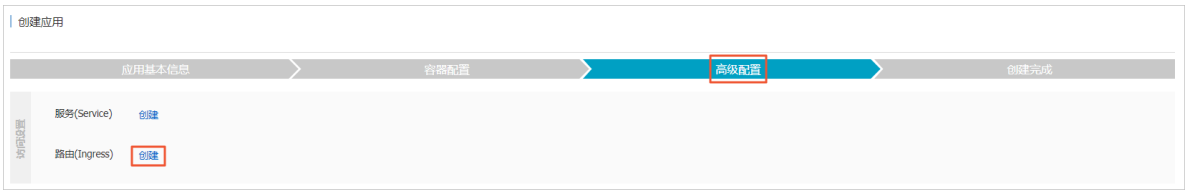
容器服务Kubernetes集群

在容器服务Kubernetes集群可以通过路由（Ingress）功能来实现，用户可以通过创建路由的方式进行相关功能的创建。同时容器服务Kubernetes的Ingress还提供了蓝绿发布、灰度发布等功能，可参考：Kubernetes集群中通过Ingress实现灰度发布和蓝绿发布的概述。

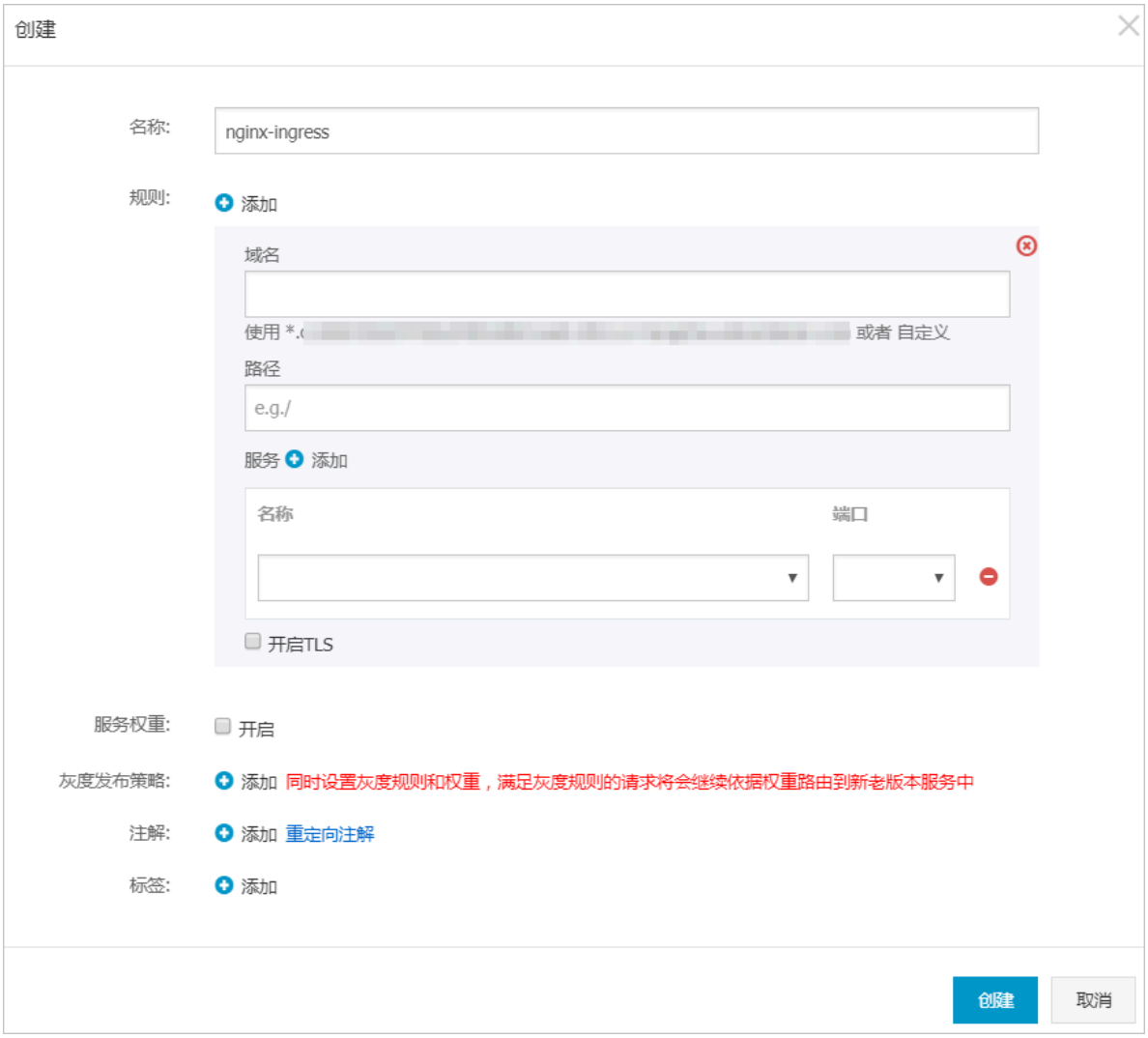
容器服务Kubernetes集群实现路由（Ingress）功能，有两种方法：

方法一：创建应用时配置

1. 部署完容器配置后，进行高级配置时，在访问设置区域，单击路由（Ingress）右侧的创建。



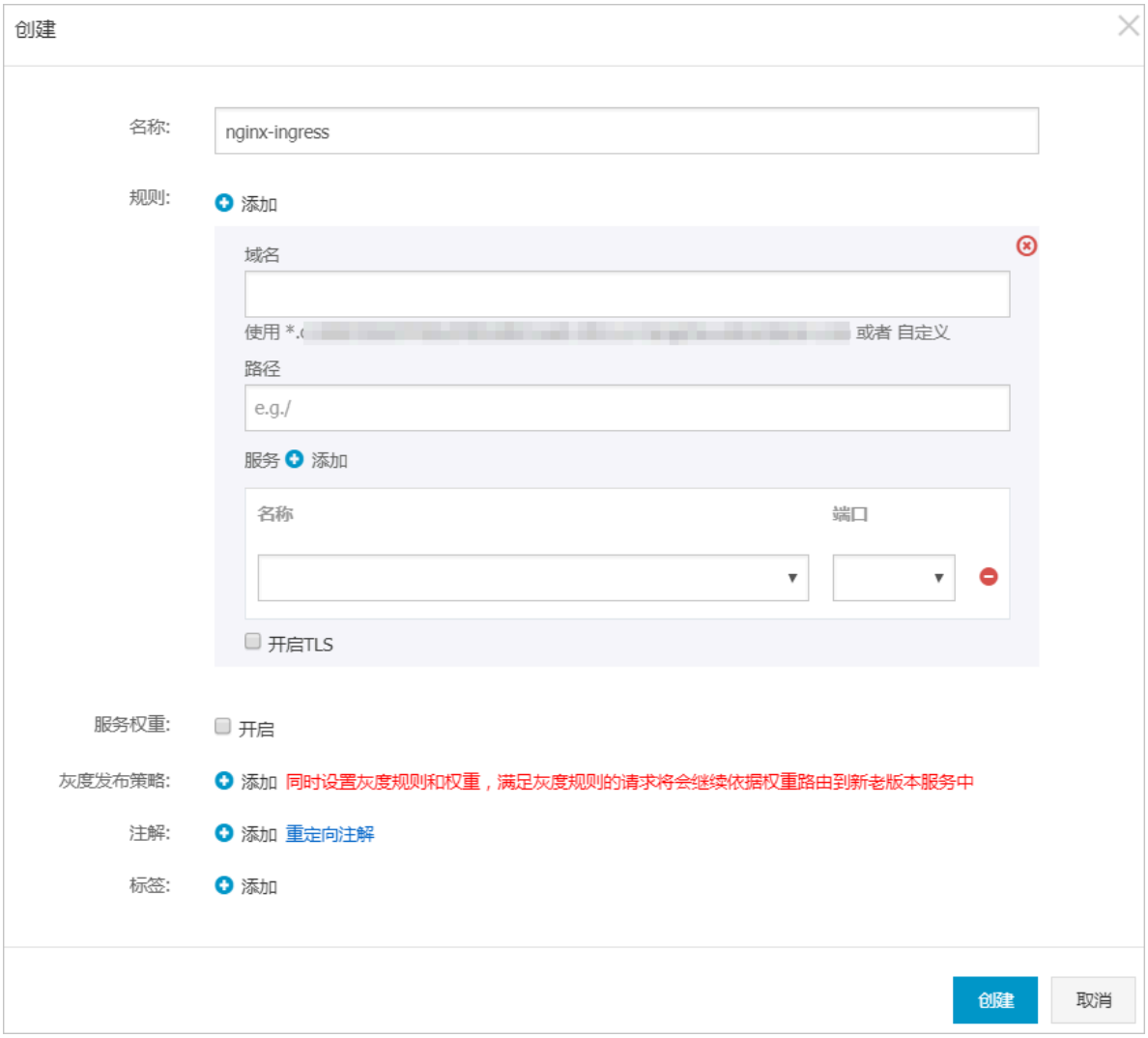
2. 详细信息可参考使用镜像创建无状态Deployment应用。



方法二：通过创建路由配置

1. 在容器管理控制台，单击左侧导航栏路由与负载均衡 > 路由，进入路由（Ingress）页面。

2. 选择目标集群和命名空间，单击创建。详细信息可参考通过 Web 界面创建路由。



负载均衡路由配置

容器服务Swarm集群

容器服务Swarm集群的负载均衡路由配置为应用提供了通过阿里云负载均衡（Server Load Balancer）暴露应用访问方式的能力，用户首先自己创建SLB，然后将创建的SLB的ID及端口信息绑定到应用上，用户可以通过<SLB_IP>:<Port>的方式访问应用。

容器服务Kubernetes集群

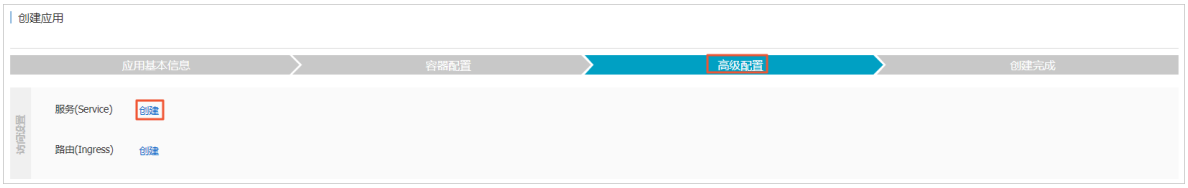
容器服务Kubernetes集群同样支持通过绑定SLB的方式进行应用访问方式的暴露。容器服务Kubernetes集群上创建SLB是通过LoadBalancer类型的Service进行自动创建，不需要手工

创建后配置。自动创建SLB时，可以选择公网访问或内部访问。同时如果使用yaml文件进行创建的话，还可以指定已有SLB及支持会话保持等配置，详细信息可参考创建服务。

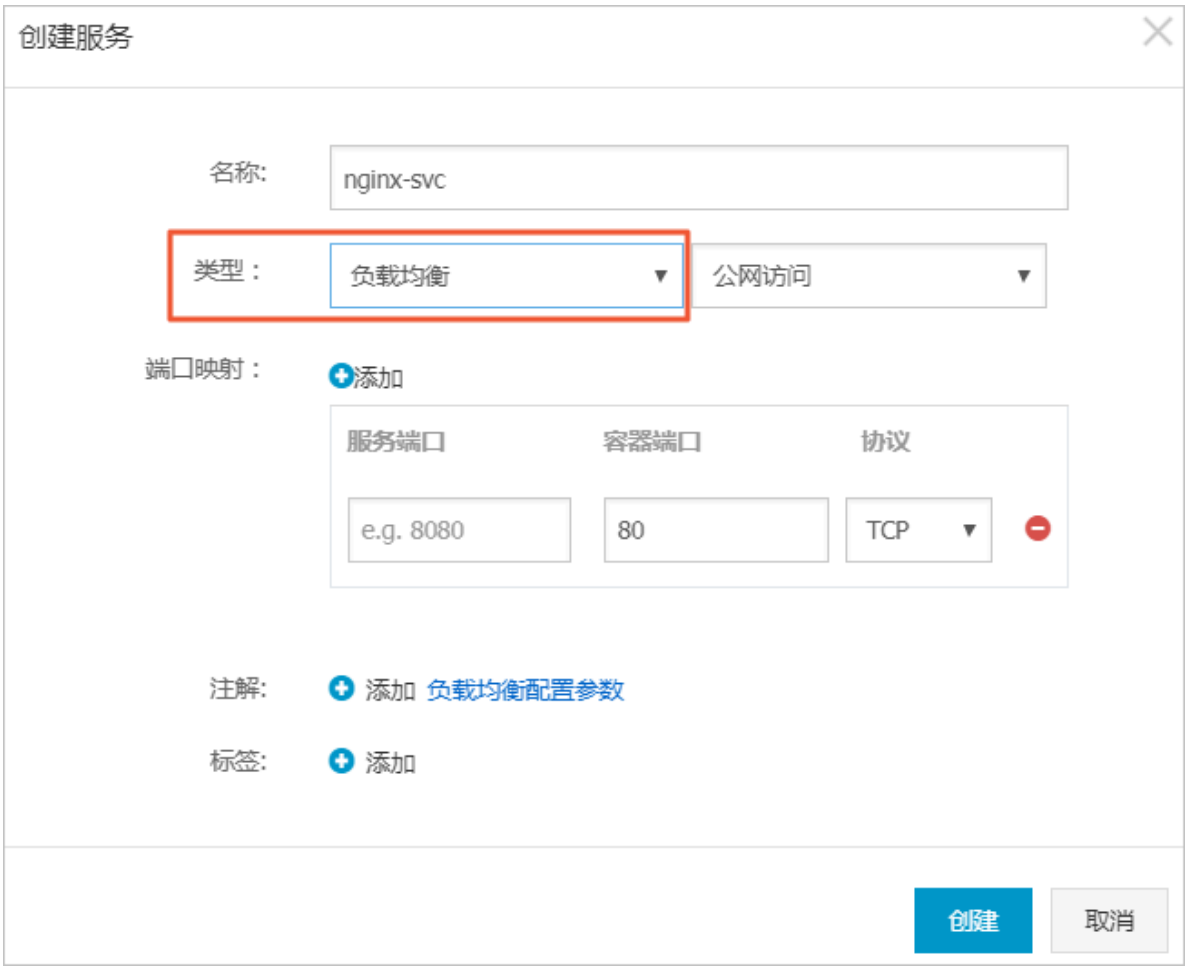
容器服务Kubernetes集群上创建Loadbalancer类型的Service有两种方式：

方法一：创建应用时配置

1. 部署完容器配置后，进行高级配置时，在访问设置区域，单击服务（Service）右侧的创建。



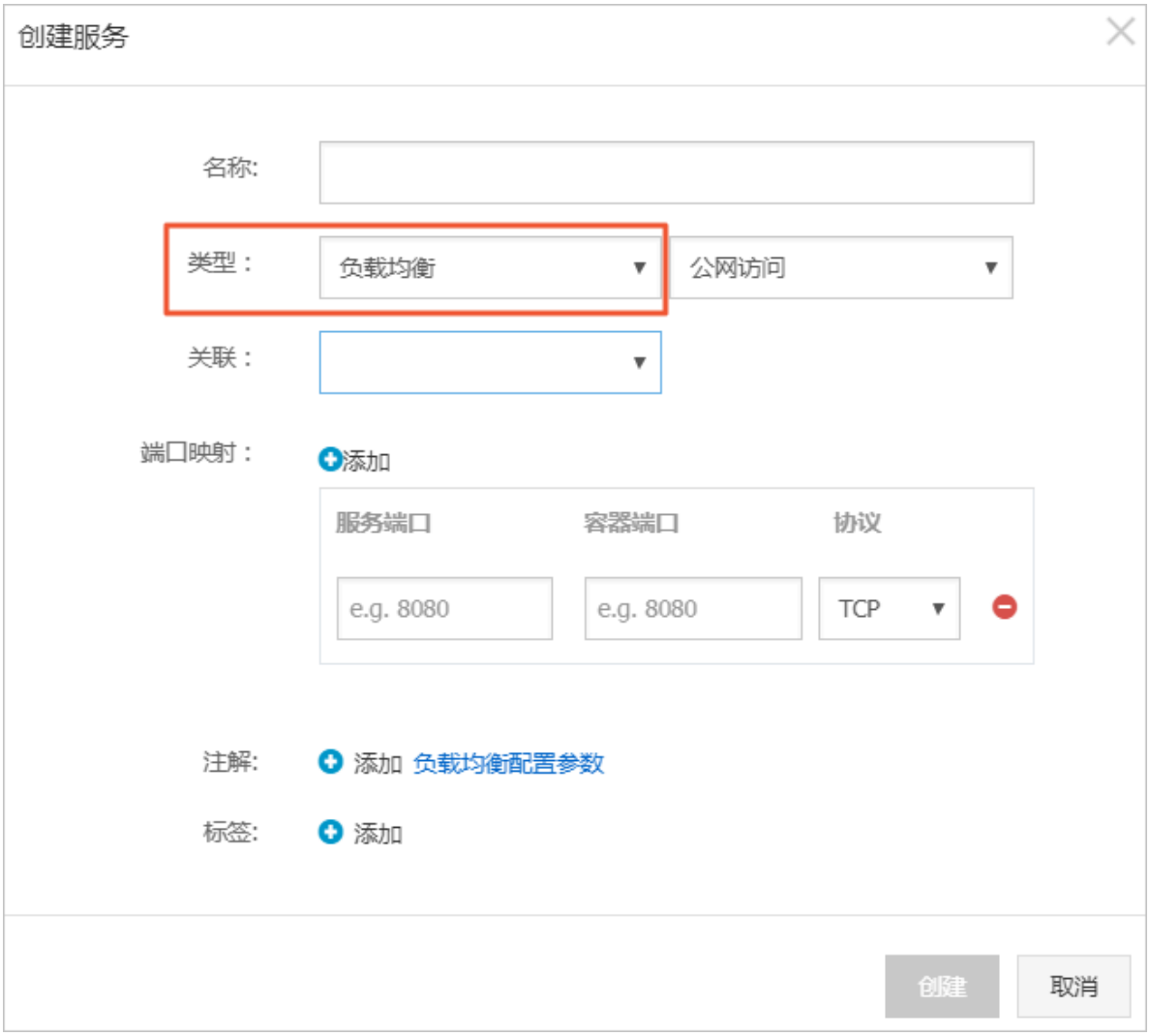
2. 类型请选择负载均衡。详细信息可参考使用镜像创建无状态Deployment应用。



方法二：通过创建服务配置

1. 在容器管理控制台，单击左侧导航栏路由与负载均衡 > 服务，进入服务（Service）页面。

2. 选择目标集群和命名空间，单击创建，在创建服务页面，类型选负载均衡。详细信息可参考创建服务。



1.5 使用镜像创建应用-数据卷及环境变量配置比对

本文介绍容器服务Swarm集群与Kubernetes集群使用镜像创建应用时，数据卷及环境变量的配置比对。

使用镜像创建应用

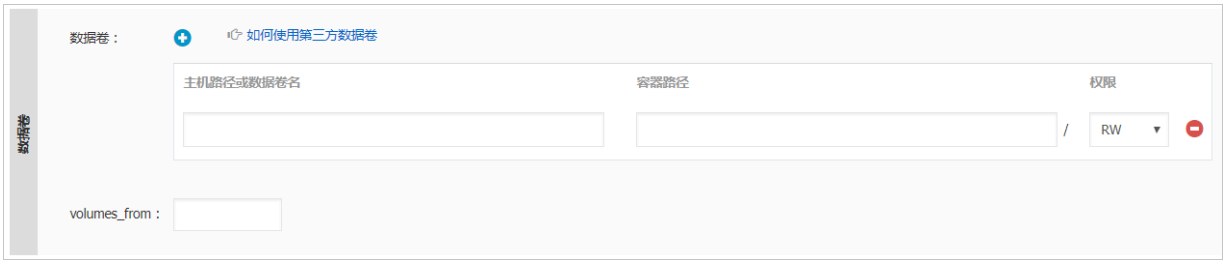
容器服务Swarm与Kubernetes集群在使用镜像创建应用时，部署界面差异较大。

- 容器服务Swarm集群，请参考创建应用。
- 容器服务Kubernetes集群，请参考使用镜像创建无状态Deployment应用。

数据卷

容器服务Swarm集群

填写用户申请的云存储或者本地存储路径。



容器服务Kubernetes集群

容器服务Kubernetes集群使用存储的方式与容器服务Swarm集群一样，只是挂载的过程不一样，控制台界面基本一致。

用户可以选择本地存储，也可以选择云存储：

- 本地存储：包括主机目录、Kubernetes集群特有的配置项（configmap）、保密字典（secret）及临时目录。
- 云存储：包括云盘、NAS及OSS。

环境变量

容器服务Swarm集群与容器服务Kubernetes集群环境变量的配置一样。用户只需输入键值即可。

1.6 使用镜像创建应用-容器配置及标签比对

本文介绍容器服务Swarm集群与Kubernetes集群使用镜像创建应用时，容器配置及标签的比对。

使用镜像创建应用

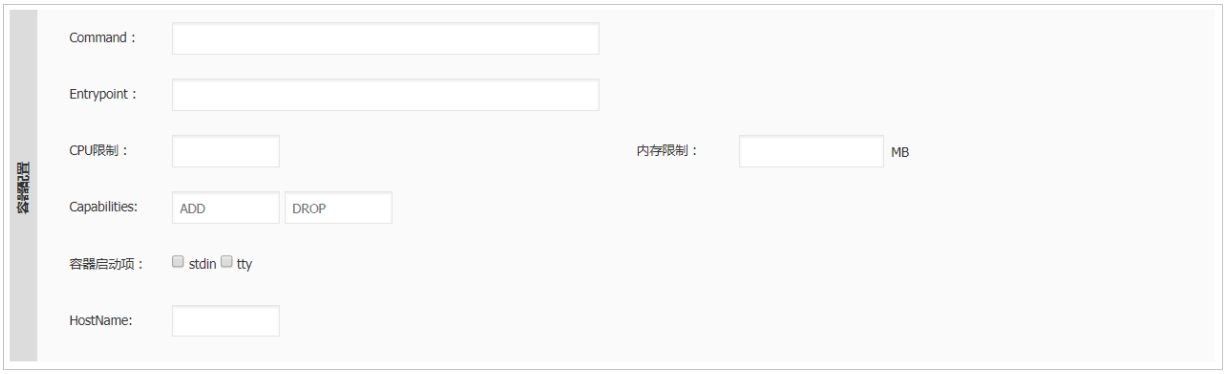
容器服务Swarm与Kubernetes集群在使用镜像创建应用时，部署界面差异较大。

- 容器服务Swarm集群，请参考创建应用。
- 容器服务Kubernetes集群，请参考使用镜像创建无状态Deployment应用。

容器配置

容器服务Swarm集群

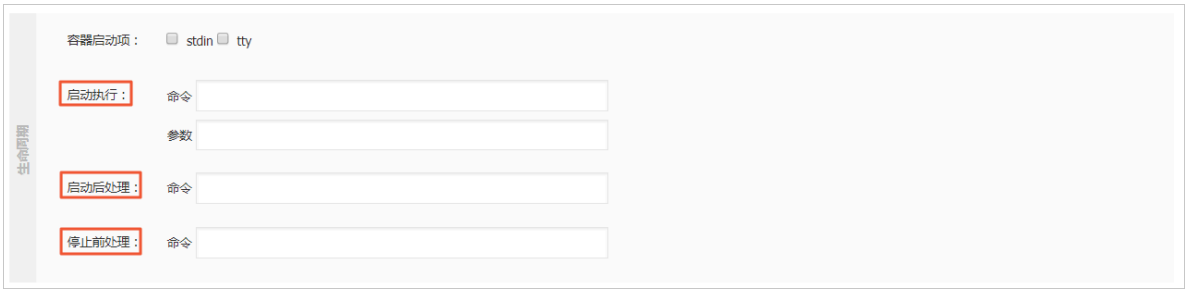
设置容器的启动命令（Command和Entrypoint）、资源限制（CPU限制和内存限制）及启动项等配置。



容器服务Kubernetes集群

容器服务swarm集群的容器配置，类似于容器服务Kubernetes集群的：

- 生命周期，各项解释详见使用镜像创建无状态Deployment应用：
  - 启动执行
  - 启动后处理
  - 停止前处理



- 基本配置，各项解释详见使用镜像创建无状态Deployment应用，推荐配置详见高可靠推荐配置：
  - 资源限制
  - 所需资源

标签

容器服务Swarm集群

能够完成健康检查、设置访问域名、日志等功能。

容器服务Kubernetes集群

容器服务Kubernetes集群中的标签只能标识一个应用，容器服务Swarm集群的健康检查，设置访问域名等功能均通过其他方式实现。

容器服务Kubernetes部署的应用在创建时，会自动生成与应用名相同的标签，在使用镜像创建应用的配置界面没有展现，用户可以通过yaml文件的方式使用此标签。

1.7 使用镜像创建应用-健康检查及自动伸缩比对

本文介绍容器服务Swarm集群与Kubernetes集群使用镜像创建应用时，健康检查及自动伸缩的比对。

使用镜像创建应用

容器服务Swarm与Kubernetes集群在使用镜像创建应用时，部署界面差异较大。

- 容器服务Swarm集群，请参考创建应用。
- 容器服务Kubernetes集群，请参考使用镜像创建无状态Deployment应用。

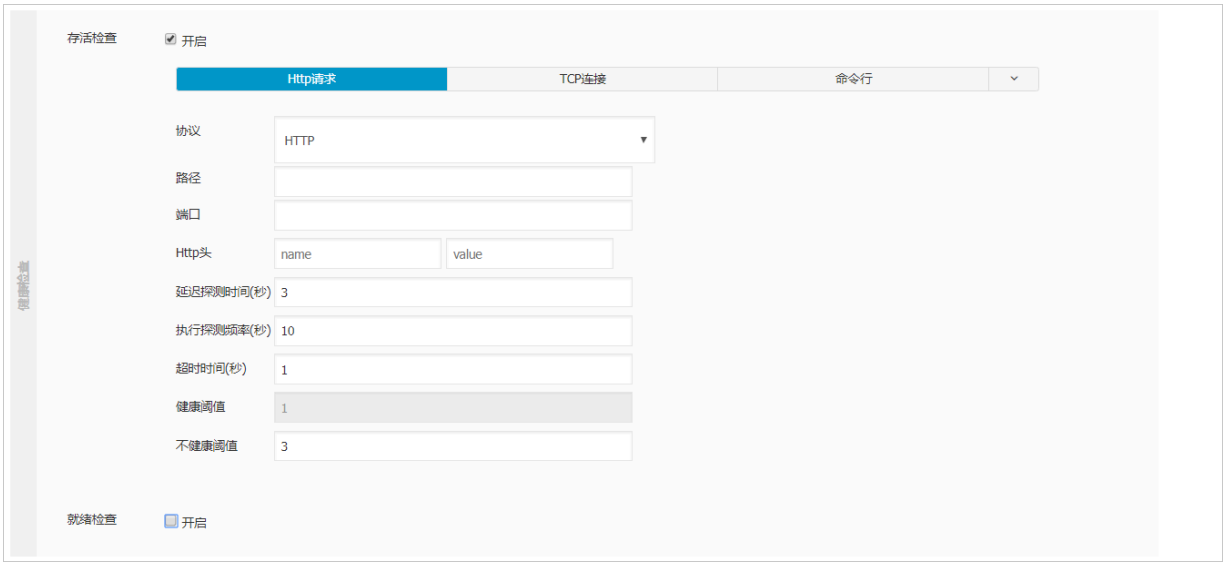
健康检查

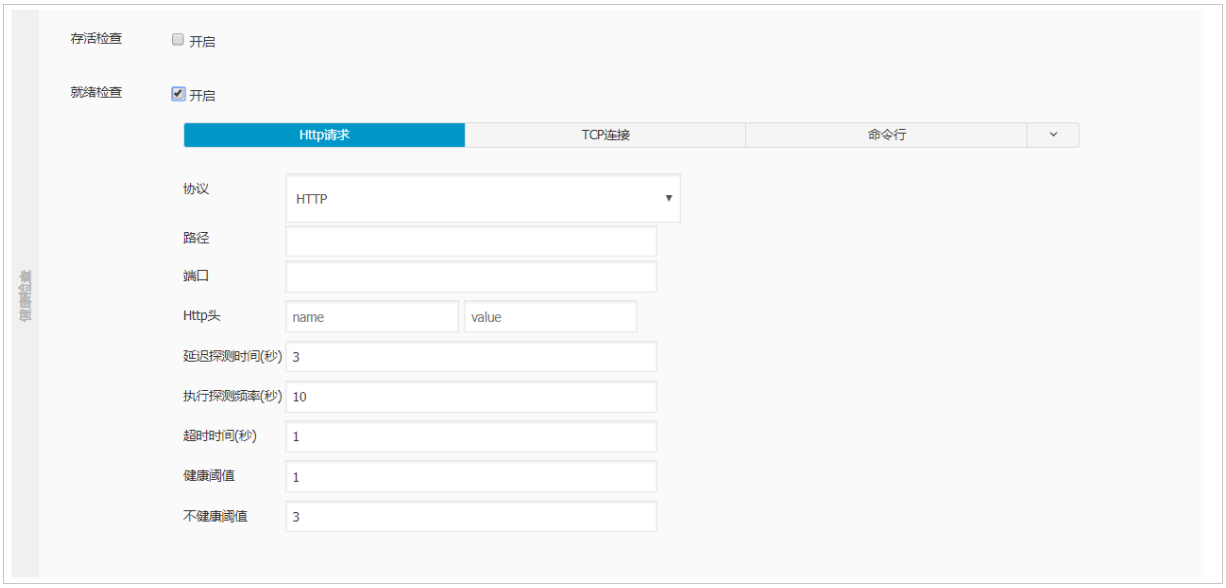
容器服务Swarm集群

健康检查是通过标签的方式实现。

容器服务Kubernetes集群

在使用镜像创建应用的容器配置页签，可进行健康检查的配置。目前支持存活检查和就绪检查。





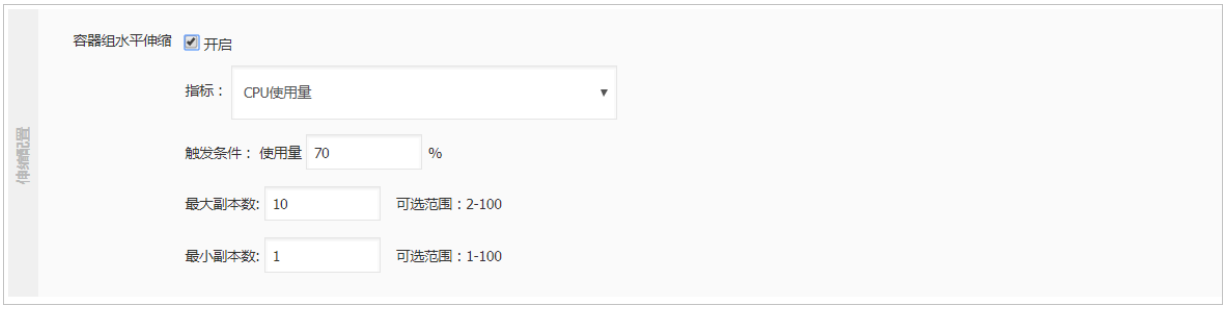
自动伸缩

容器服务Swarm集群

提供基于CPU和内存两个维度的自动伸缩。

容器服务Kubernetes集群

可配置容器组的水平伸缩，同样提供CPU和内存两个维度的自动伸缩。



1.8 使用yaml文件创建应用比对

本文介绍容器服务Swarm集群与Kubernetes集群使用yaml文件创建应用时，Swarm集群下的yaml文件与Kubernetes集群下的yaml文件的对应关系。

背景信息

在使用yaml文件创建应用时，Swarm集群与Kubernetes集群的yaml文件格式不一样：

- 您可以通过Kompose工具对Swarm集群的yaml文件进行自动转换。但转换后的内容仍需您核对检查。

  Kompose工具可以从Github上获取：*https://github.com/AliyunContainerService/kompose*。

  Kompose工具下载地址：

  - Mac下载地址：*http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/swarm/kompose-darwin-amd64*
  - Linux下载地址：*http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/swarm/kompose-linux-amd64*
  - Window下载地址：*http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/swarm/kompose-windows-amd64.exe*

  说明：

  目前Kompose工具对阿里云的一些定制标签还不支持，如表*1*所示，阿里云容器服务团队会持续研发逐步覆盖各个标签的支持。

  表 1-1: Kompose工具不支持的标签

标签	参考
external	*external*
dns_options	*dns_options*
oom_kill_disable	*oom_kill_disable*
affinity:service	服务部署约束*#affinity:service#*

- 您也可以手动改写yaml文件。

本文将基于容器服务Swarm的yaml文件，介绍介绍Kubernetes的yaml文件如何与之对应。文章中的yaml示例，仅作为示例，具体部署请依据具体情况添加及修改相关内容。

容器服务Swarm与Kubernetes集群的yaml文件比对

容器服务Swarm集群

容器服务Swarm集群的yaml文件*wordpress-swarm.yaml*如下，注释中的阿拉伯数字与容器服务Kubernetes集群的yaml文件的注释对应。

```
web:  #---1
  image: registry.aliyuncs.com/acs-sample/wordpress:4.5  #---2
  ports:  #---3
    - '80'
  environment:  #---4
```

```
    WORDPRESS_AUTH_KEY: changeme    #---5
    WORDPRESS_SECURE_AUTH_KEY: changeme  #---5
    WORDPRESS_LOGGED_IN_KEY: changeme  #---5
    WORDPRESS_NONCE_KEY: changeme    #---5
    WORDPRESS_AUTH_SALT: changeme    #---5
    WORDPRESS_SECURE_AUTH_SALT: changeme  #---5
    WORDPRESS_LOGGED_IN_SALT: changeme  #---5
    WORDPRESS_NONCE_SALT: changeme    #---5
    WORDPRESS_NONCE_AA: changeme    #---5
  restart: always  #---6
  links:   #---7
    - 'db:mysql'
  labels:   #---8
    aliyun.logs: /var/log  #---9
    aliyun.probe.url: http://container/license.txt  #---10
    aliyun.probe.initial_delay_seconds: '10'    #---10
    aliyun.routing.port_80: http://wordpress    #---11
    aliyun.scale: '3'        #---12
db:    #---1
  image: registry.aliyuncs.com/acs-sample/mysql:5.7  #---2
  environment:  #---4
    MYSQL_ROOT_PASSWORD: password  #---5
  restart: always  #---6
  labels:  #---8
    aliyun.logs: /var/log/mysql  #---9
```

容器服务Kubernetes集群

通过容器服务Swarm集群的*wordpress-swarm.yaml*文件部署的wordpress应用，在容器服务Kubernetes集群中对应2个服务：web和db。

在容器服务Kubernetes集群上需要2个部署（deployment）和2个服务（service）。2个Deployment创建2个Service，2个服务分别暴露2个应用的访问方式。

容器服务Swarm集群中的web应用对应Kubernetes集群的deployment和service如下：

说明：

以下yaml文件的内容仅作为示例说明与容器服务Swarm集群*wordpress-swarm.yaml*的对应关系，不可用作实际部署。

- *wordpress-kubernetes-web-deployment.yaml*内容如下：

```
apiVersion: apps/v1      # api版本
kind: Deployment  # 创建资源的类型
metadata:
  name: wordpress       #---1
  labels:    #---8 在这里的label只能做标识作用
    app: wordpress
spec: #资源创建详细内容
  replicas: 2         #---12 设定实例（副本）个数
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
strategy:
type: Recreate
```

```
  template:   #模板定义POD的详细信息
    metadata:
      labels:  #与前面保持一致
        app: wordpress
        tier: frontend
    spec:    #定义pod中container的详细信息
      containers:    #
      - image: wordpress:4   #---2  对应于镜像及版本
        name: wordpress
        env:    #---4 环境变量设置，kubernetes上configmap，secret都可以通过env的方式使用
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql  #---7 通过名称指向需要访问的mysql，该名称与mysql service的名称相对应。
        - name: WORDPRESS_DB_PASSWORD    #---5 密码在这里使用，但kubernetes提供了secret进行密码封装
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-wordpress
        ports:   #---3 容器内应用暴露的port
        - containerPort: 80
          name: wordpress
livenessProbe:        #add health check    ---10 健康检查
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 30
          timeoutSeconds: 5
          periodSeconds: 5
        readinessProbe:       #add health check    ---10 健康检查
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 5
          timeoutSeconds: 1
          periodSeconds: 5
        volumeMounts:  #使用存储卷，将存储卷真正挂到容器内部
        - name: wordpress-pvc
          mountPath: /var/www/html
      volumes:   #获取存储卷，需先进行PV和PVC的创建
      - name: wordpress-pvc
        persistentVolumeClaim:
          claimName: wordpress-pv-claim
```

- *wordpress-kubernetes-web-service.yaml*内容如下：

```
apiVersion: v1   #版本号
kind: Service    #创建资源类型，在这里为service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80   #服务端口号
  selector:  #通过label进行应用的关联
    app: wordpress
    tier: frontend
```

```
  type: LoadBalancer  #---11 定义访问方式，此处为LoadBalancer类型的
service，会自动创建SLB
```

容器服务Swarm集群中的db应用对应Kubernetes集群的deployment和service如下：

说明：

以下yaml文件的内容仅作为示例说明与容器服务Swarm集群*wordpress-swarm.yaml*的对应关系，不可用作实际部署。

- *wordpress-kubernetes-db-deployment.yaml*内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-mysql
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: wordpress-mysql-pvc
          mountPath: /var/lib/mysql
      volumes:
      - name: wordpress-mysql-pvc
        persistentVolumeClaim:
          claimName: wordpress-mysql-pv-claim
```

- *wordpress-kubernetes-db-service.yaml*内容如下：

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
```

```
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
      clusterIP: None
```

1.9 网络比对

本文介绍容器服务Swarm集群与Kubernetes集群的网络比对。

容器服务Swarm集群

容器服务Swarm集群使用的网络有两种：

- VPC网络
- 经典网络

容器服务Kubernetes集群

容器服务Kubernetes集群的网络为VPC网络，可参考*VPC下 Kubernetes* 的网络地址段规划。

- 若容器服务Swarm集群使用的是VPC网络，在创建Kubernetes集群时，请选择与Swarm集群相同的VPC网络，以便Kubernetes集群与Swarm集群的网络是联通的。
- 若容器服务Swarm集群使用的是经典网络，在创建Kubernetes集群时，由于Kubernetes集群只能为VPC网络，请先打通Swarm集群与Kubernetes集群的网络，可参考迁移方案概述。

  Swarm集群与Kubernetes集群网络联通后，如果Swarm集群下存在OSS、NAS或RDS等存储产品或数据库，会获得VPC网络可访问的IP地址，即在Kubernetes集群的PVC网络中可通过此IP地址对Swarm集群的存储产品或数据库进行访问。

1.10 日志及监控比对

本文介绍容器服务Swarm集群与Kubernetes集群的日志及监控功能的比对。

日志

容器服务Swarm集群

通过标签实现日志等功能。

容器服务Kubernetes集群

Kubernetes集群的日志功能，可在以下情况配置及使用：

- 创建Kubernetes集群：

  在创建Kubernetes集群页面，勾选使用SLS，会在集群中自动配置日志服务插件。您可以使用已有的Project，也可以创建新的Project。

  集群创建成功后，也可以手动安装日志服务组件，请参见已创建 Kubernetes 集群#手动安装日志服务组件。

- 创建应用时配置日志服务，请参见创建应用时配置日志服务。
- 创建应用后使用日志服务，请参见容器文本日志及容器标准输出。

监控

容器服务Swarm集群和Kubernetes集群，均在创建集群页面，勾选在ECS节点上安装云监控插件，即可在云监控管理控制台查看所创建ECS实例的监控信息。

容器服务Swarm集群

默认情况，不勾选。

容器服务Kubernetes集群

默认情况，勾选。

容器服务Kubernetes集群与云监控的集成使用，请参考与云监控集成与使用。

1.11 应用访问比对

本文介绍容器服务Swarm集群与Kubernetes集群的应用访问比对，包括集群内部应用间访问及从集群外部访问应用。

集群内部应用间访问

容器服务Swarm集群

集群内部可以通过links标签，将需要被访问的服务名称设置到容器的环境变量中。

例如：使用yaml文件创建应用比对中，wordpress应用的web服务与mysql关联，在容器启动后，通过mysql这个服务名称即可完成服务的访问。

```
links:   #---7
    - 'db:mysql'
```

容器服务Kubernetes集群

在容器服务Kubernetes集群内部，可通过Service的ClusterIP或服务名称进行应用间访问。推荐使用服务名称进行应用间的访问。

在创建应用时，可以将需要被访问服务的服务名称以环境变量的方式使用。

例如：使用yaml文件创建应用比对中，wordpress在调用mysql服务时，就是通过环境变量的方式实现。

```
spec:
      containers:
      - image: wordpress:4
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql    #---7 通过名称指向需要访问的mysql，该名称与mysql service的名称相对应。
        - name: WORDPRESS_DB_PASSWORD
```

从集群外部访问应用

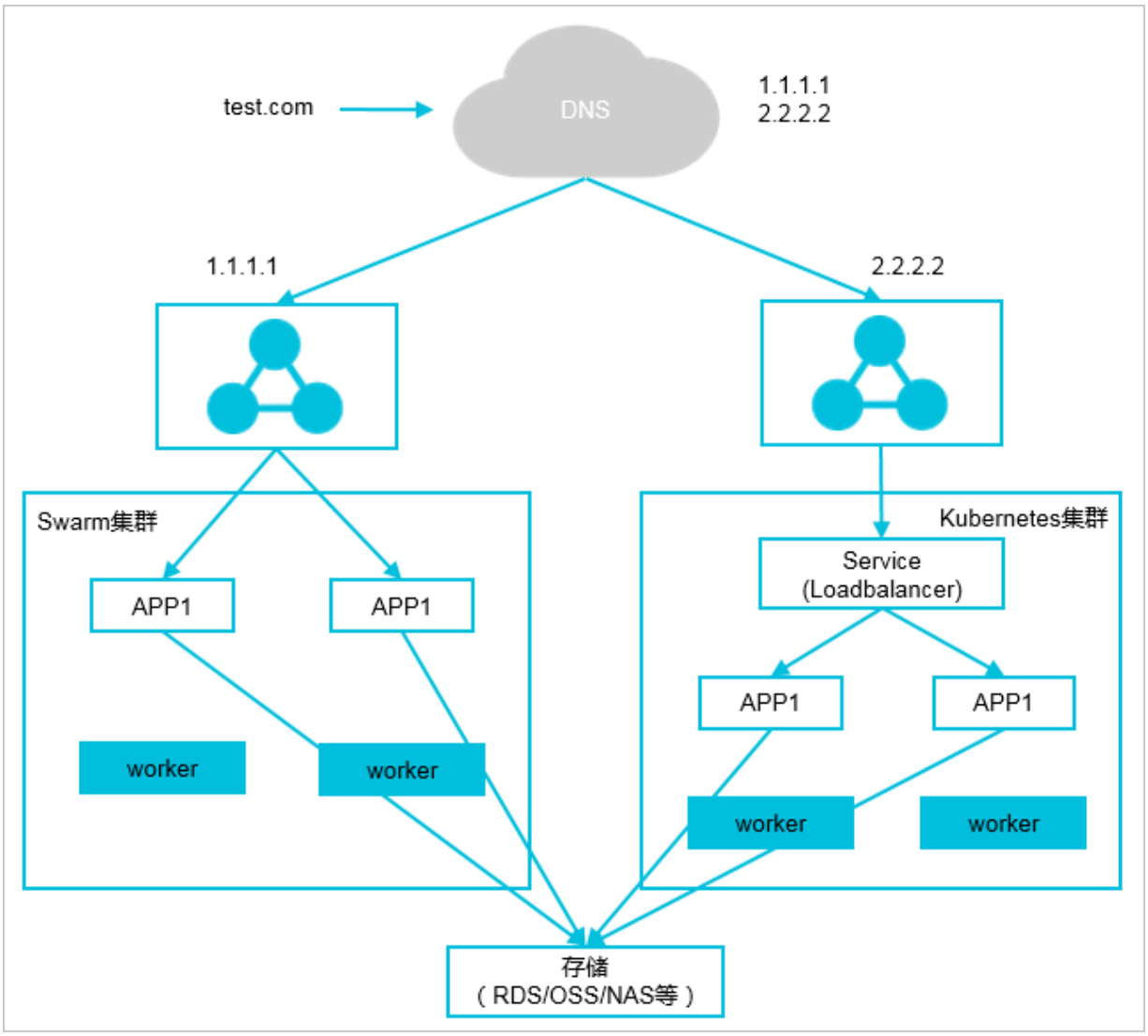
通过域名访问应用

说明:

- 无论经典网络还是VPC，请确保网络的连通。
- DNS具有负载均衡的能力，可将流量分发到不同的后端IP。
- 通过域名作为应用的访问入口，可以不停机地将业务从Swarm集群迁移到Kubernetes集群。

简单路由（域名绑定到容器服务Swarm集群默认的SLB上）

为实现应用从容器服务Swarm集群迁移到Kubernetes集群，需要先在容器服务Kubernetes集群上创建应用，测试可用后，再进行应用迁移。



迁移方法

- 在容器服务Kubernetes集群上通过以下步骤创建应用
  - 在容器服务Kubernetes集群上创建与Swarm集群相同类型的应用。
  - 在容器服务Kubernetes集群上为应用创建Loadbalancer类型的服务（Service）。
  - Loadbalancer类型的服务会创建SLB，在本例中为2.2.2.2。
  - 在DNS中`test.com`域名的后端IP增加2.2.2.2这个IP地址。
- 验证Kubernetes集群的应用可用

  通过IP地址2.2.2.2可以正常访问应用，说明容器服务Kubernetes集群上的应用可用。
- 应用迁移

  将DNS中`test.com`域名的后端IP地址1.1.1.1删除。

后续流量经过DNS后全部流向Kubernetes集群上的应用。

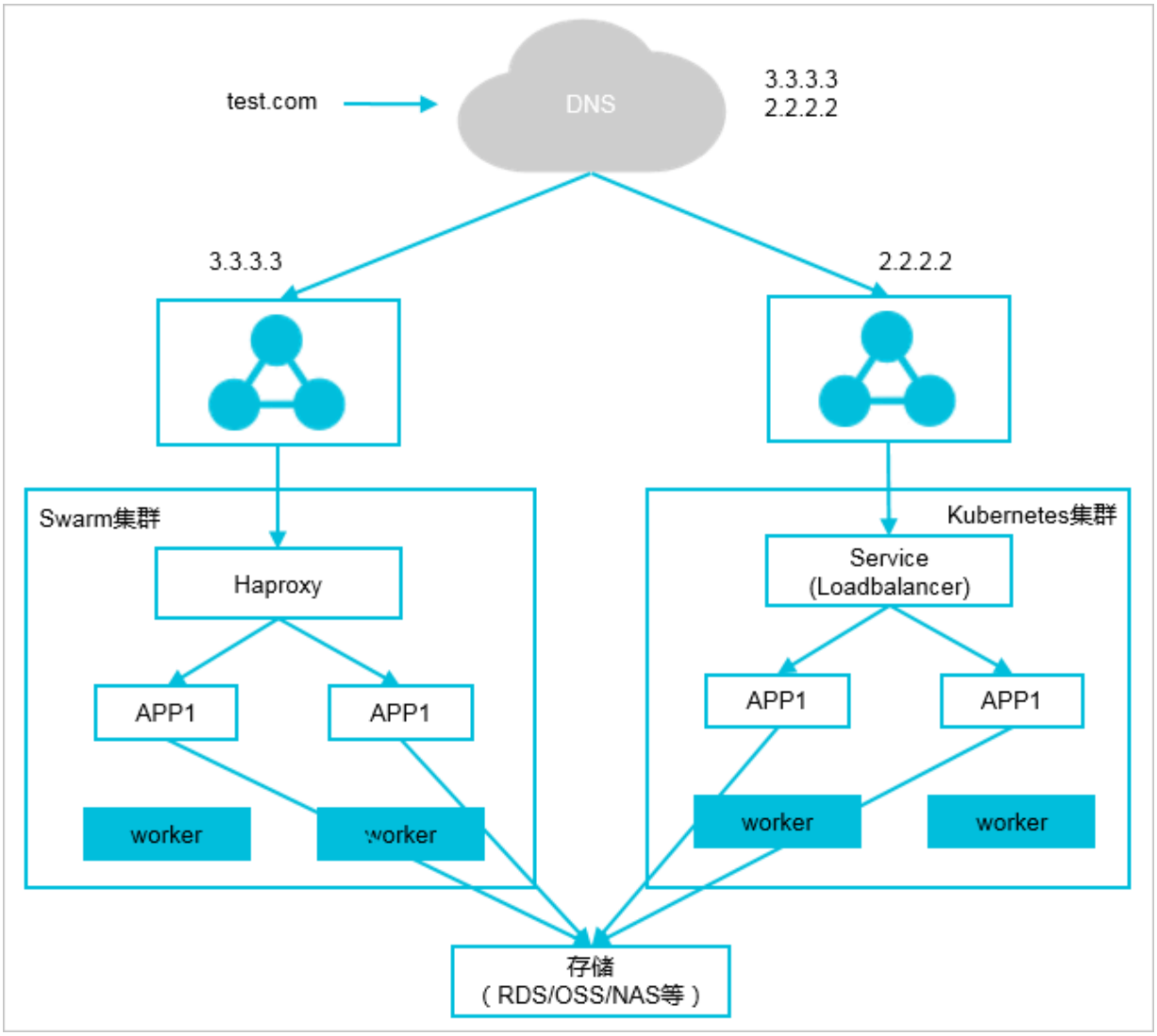
简单路由（域名绑定到应用在容器服务Swarm集群上自建的SLB）

域名绑定到容器服务Swarm集群上自建的SLB与绑定到默认的SLB的区别是：

- SLB不是集群默认的，而是自建的。
- DNS默认为云解析DNS，如果用户使用自己的域名，需要自行解析。

迁移方法

与域名绑定到容器服务Swarm集群默认的SLB的方法相同：在容器服务Kubernetes集群上创建应用，测试可用后，再进行应用迁移。



通过<HostIP>:<port>访问应用

如果您通过<HostIP>:<port>的方式访问应用，那么在原容器服务Swarm集群的基础上无法做到不停机迁移业务，请选择业务量小的时候进行业务的迁移。

迁移方法

1. 在容器服务Kubernetes集群上创建应用，并通过NodePort类型的服务完成应用对外访问方式的暴露，请参考端口映射。
2. 记录该<NodePort>，将Swarm集群的<port>替换为Kubernetes集群的<NodePort>。

   说明:

   此步骤需要逐个停止并修改应用实例。
3. 将Kubernetes集群内的worker节点，挂载到Swarm集群的SLB下。
4. 当有流量时，会有部分流量进入Kubernetes集群的应用，待Kubernetes集群的应用验证可用后，将SLB中Swarm集群的节点删除，完成集群的迁移。

通过负载均衡访问应用

如果您通过负载均衡的方式访问应用，那么在原容器服务Swarm集群的基础上无法做到不停机迁移业务，请选择业务量小的时候进行业务的迁移。

迁移方法

容器Kubernetes集群LoadBalancer的使用方法与容器服务Swarm的负载均衡一样，请参考负载均衡路由配置。

2 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GN4型GPU 云服务器集群

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。

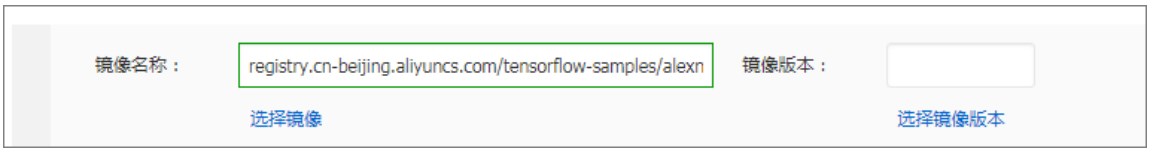
操作步骤

1. 登录容器服务管理控制台。
2. 在Swarm菜单下，单击左侧导航栏中的镜像与方案 > 镜像。
3. 在搜索框中输入 `alexNet_benchmark` 并单击全局搜索。
4. 单击 `registry.cn-beijing.aliyuncs.com/tensorflow-samples/alexnet_benchmark:1.0.0-devel-gpu` 右边的创建应用。

5. 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。

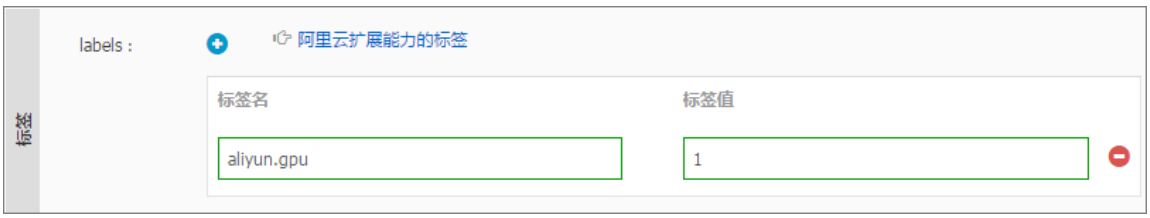
6. 配置应用。

a. 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`。



b. 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`。

c. 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`。

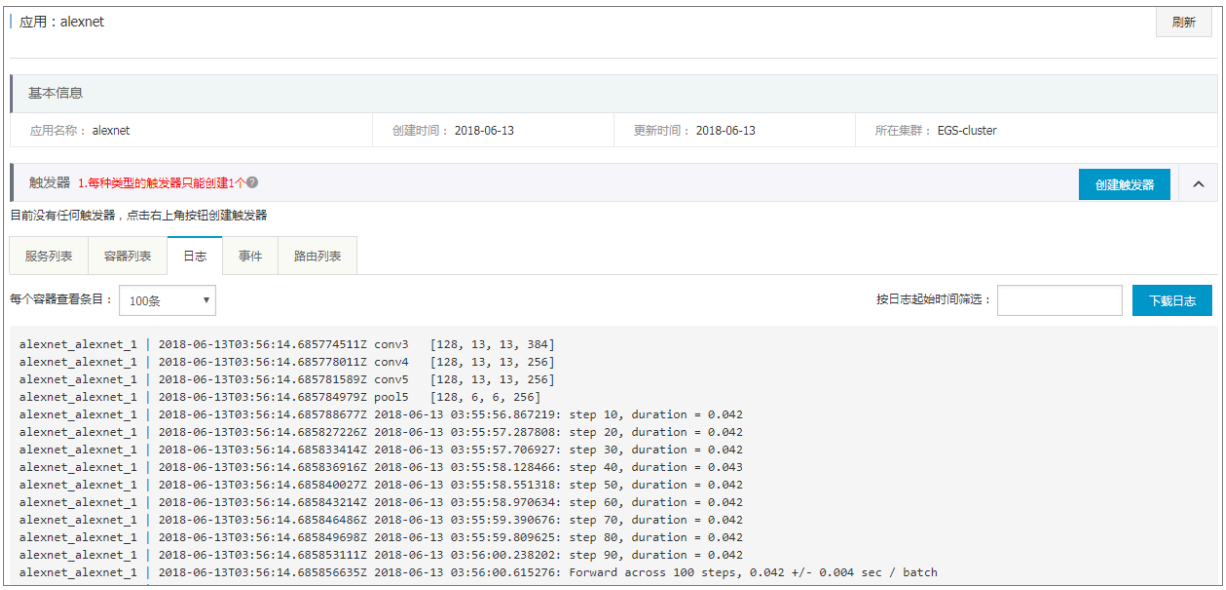


7. 完成应用配置后，单击创建创建应用。

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。



3 节点重启操作最佳实践

直接重启节点可能会导致集群出现异常。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：

1. 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。

2. 验证业务容器配置有效性

   对于 swarm 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。

3. 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。

4. 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。

5. 重启节点

   在控制台或系统内部，正常重启节点。

6. 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。

4 使用 OSSFS 数据卷实现 WordPress 附件共享

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。

> **说明:**
> 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

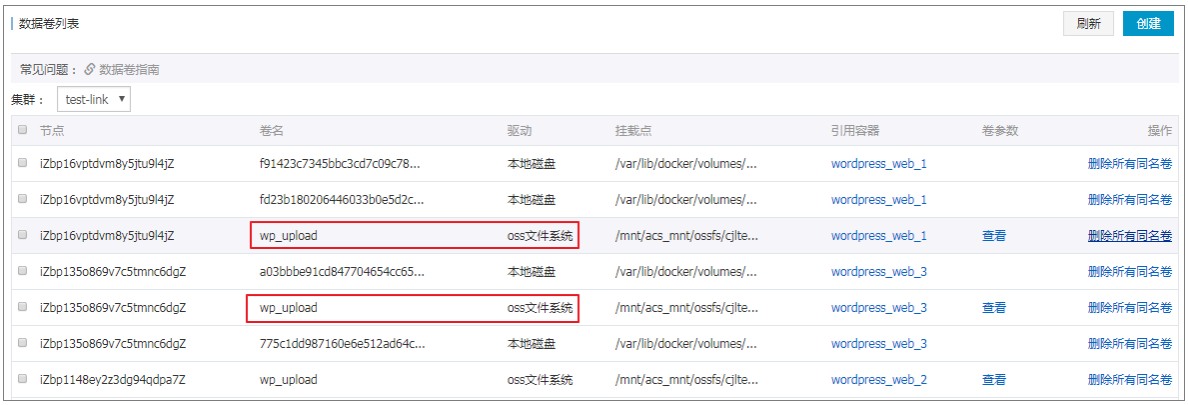
1. 创建 OSSFS 数据卷。

   a. 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。

   b. 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。



2. 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。

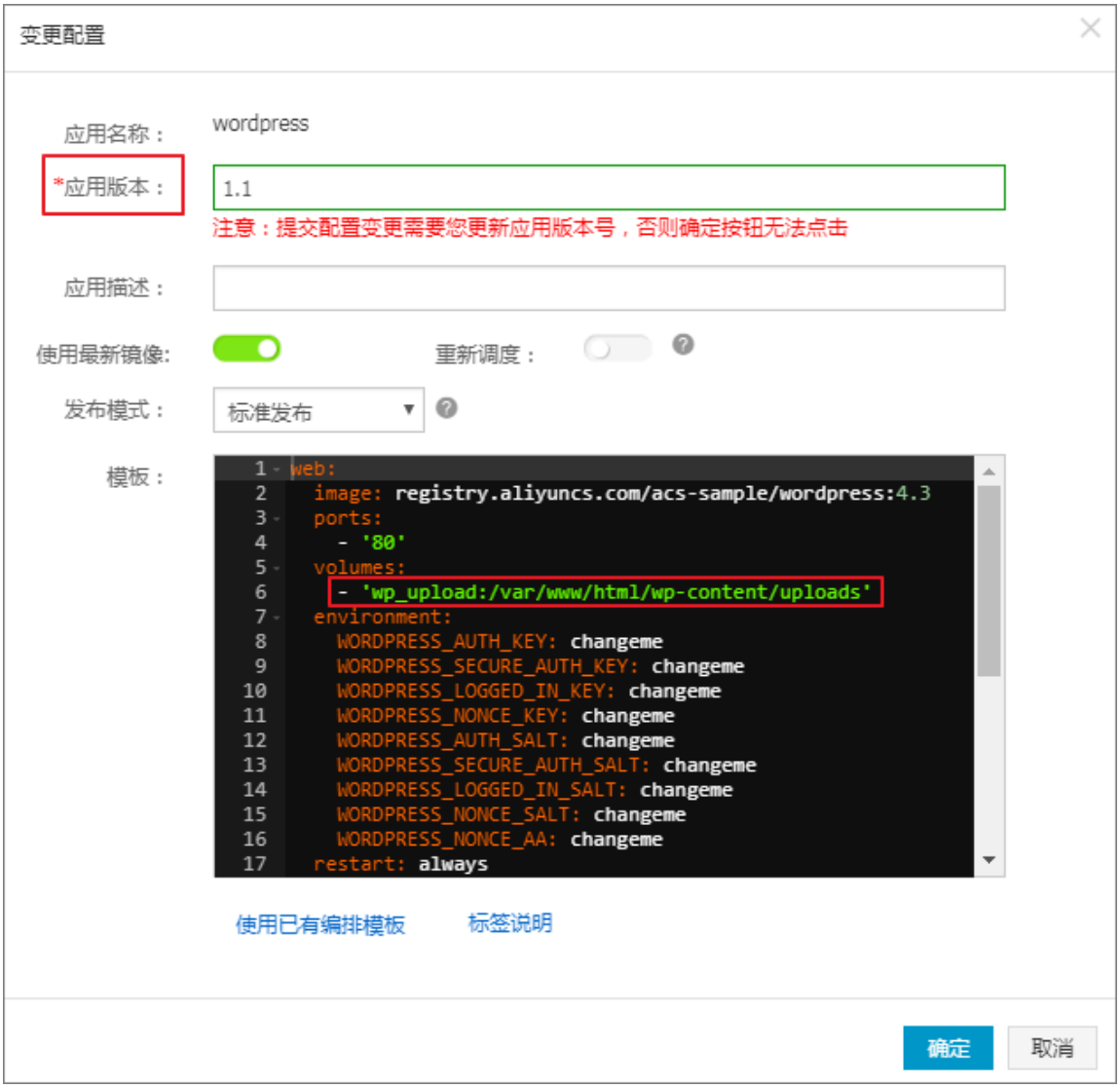
   a. 在 容器服务管理控制台，在 Swarm 菜单下，单击左侧导航栏中的应用。

   b. 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。

   c. 在模板中添加 OSSFS 数据卷到 WordPress 目录的映射。

   说明:

您必须修改 应用版本，否则无法重新部署应用。



d. 单击确定，重新部署应用。

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

5 使用 Docker Compose 测试集群网络连通性

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

1. 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。

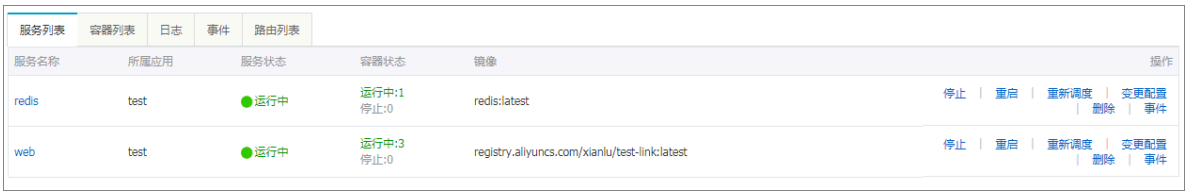
    说明:

创建集群时，请创建负载均衡实例。

2. 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。

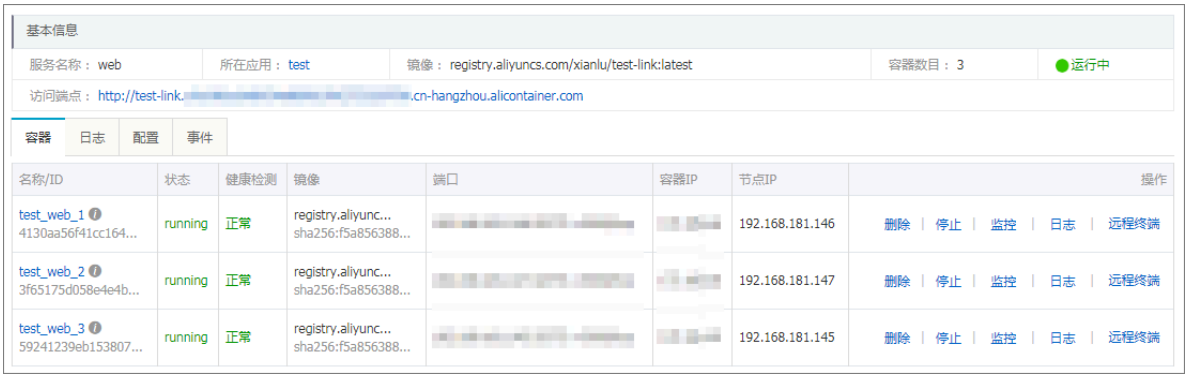
   有关如何创建应用的详细信息，参见 创建应用。

3. 在应用列表页面，单击应用的名称，查看创建的服务。



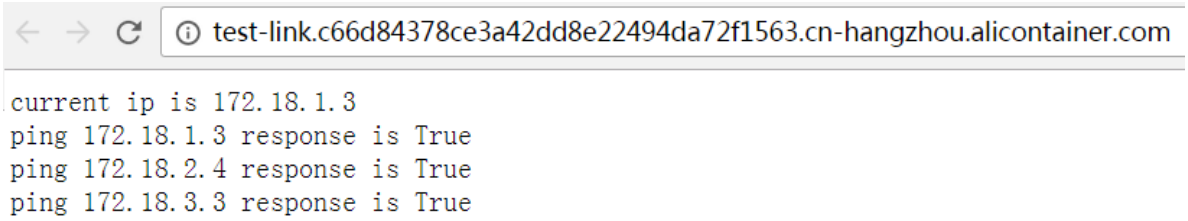
4. 单击 web 服务的名称，进入服务详情页面。

   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。

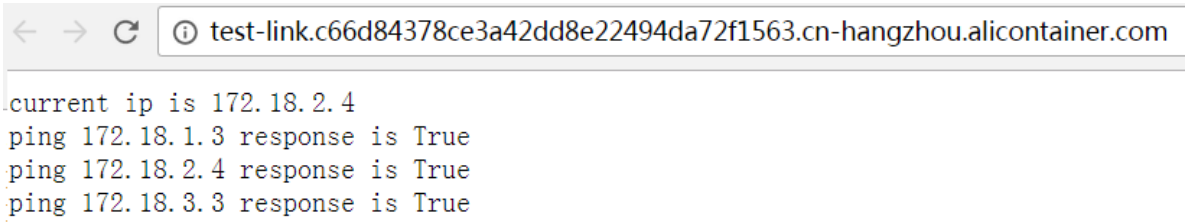


5. 访问 web 服务的访问端点。

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。

6 日志

6.1 容器服务中使用 ELK

本文档介绍如何在容器服务里使用 ELK。

背景信息

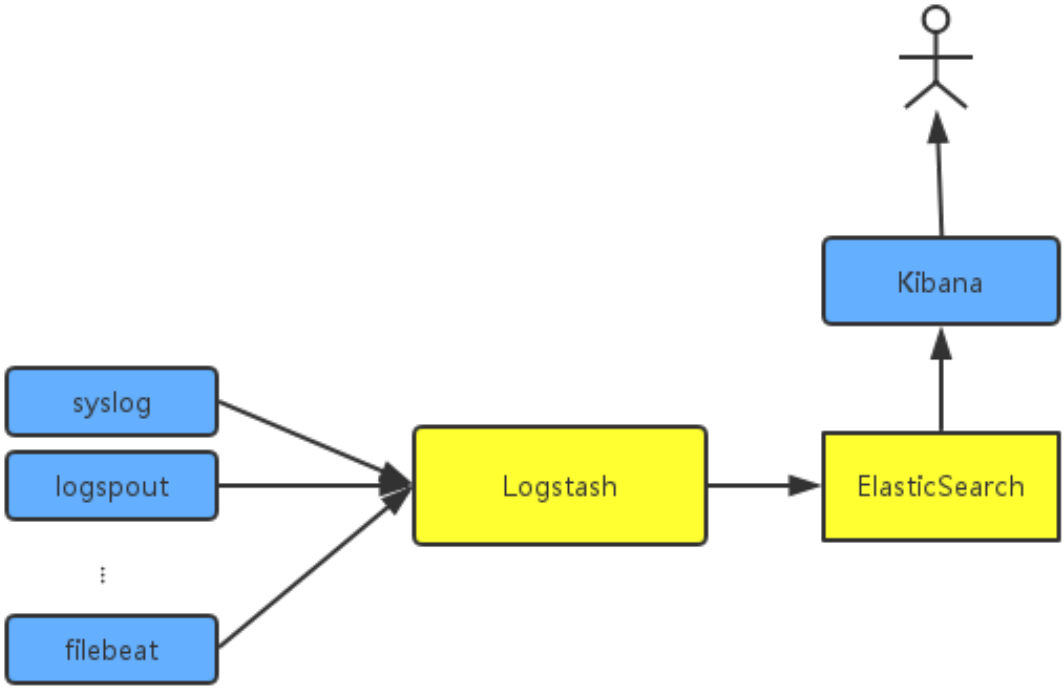
日志是 IT 系统的重要组成部分，记录了系统在什么时候发生了什么事情。我们可以根据日志排查系统故障，也可以做统计分析。

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

> **说明:**
> 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。

1. 创建应用前，登录 负载均衡管理控制台。
2. 创建一个公网类型的负载均衡实例。

3. 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

1. 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。

   > 说明:
   > 集群必须和上边创建的负载均衡实例位于同一地域。

2. 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。

3. 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。

   说明:

   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 }
```

```
}
```

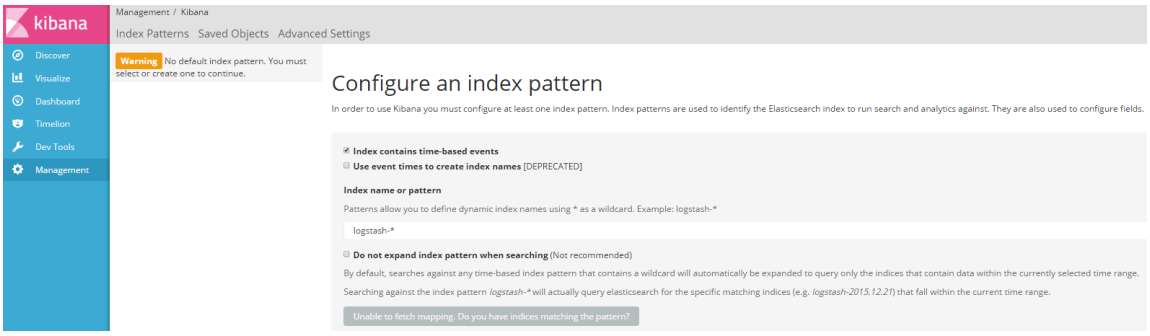
4. 配置 kibana index。

   a. 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。

   b. 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。

1. 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。

   说明:
   集群必须和上边创建的负载均衡实例位于同一地域。

2. 使用下面的编排模板创建应用wordpress。

   说明:
   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}`。

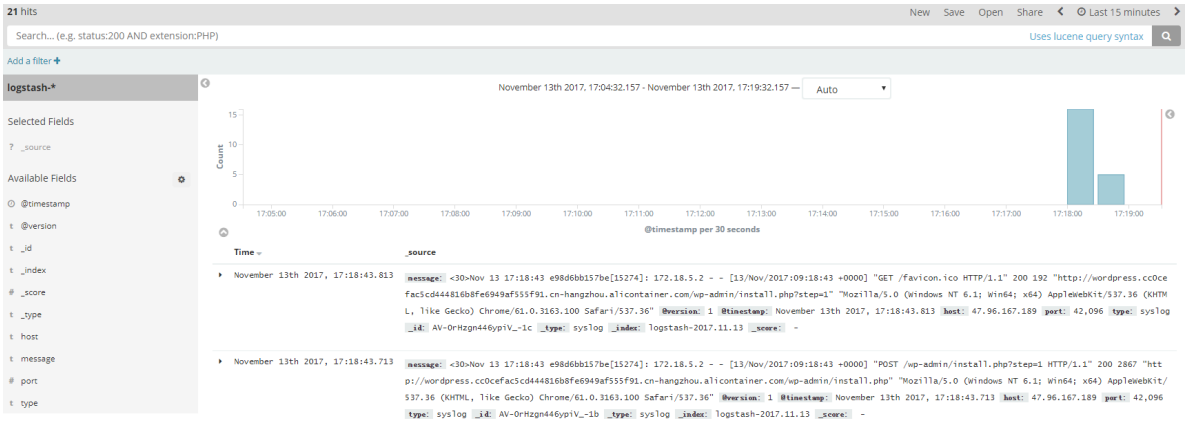
```
version: '2'
```

```
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。

3. 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。



6.2 Docker 日志收集新方案：log-pilot

本文档介绍一款新的 Docker 日志收集工具：log-pilot。log-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 log-pilot 实例，就可以收集机器上所有 Docker 应用日志。(注意：只支持Linux版本的Docker，不支持Windows/Mac版)。

log-pilot 具有如下特性：

- 一个单独的 log 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 log 进程。
- 支持文件日志和 stdout。docker log dirver 亦或 logspout 只能处理 stdout，log-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，log-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，log-pilot 都能把日志投递到正确的地点。
- 开源。log-pilot 完全开源，您可以从 *Git*项目地址 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 log-pilot，再启动一个 tomcat 容器，让 log-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 log-pilot。

> **说明:**
> 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 \
    --privileged  \
    registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.9.5-
filebeat
```

您会看到 log-pilot 的启动日志。

```
root@c                                    3-node1:/# 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:0.9.5-filebeat
Unable to find image 'registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.9.5-filebeat' locally
0.9.5-filebeat: Pulling from acs-sample/log-pilot
a073c86ecf9e: Pull complete
7ba3e804adbd: Pull complete
7bff0b2064d3: Pull complete
ee47809ba289: Pull complete
070d1b641126: Pull complete
Digest: sha256:427b5d81168a5f6584f063a814709618d7b81ed34f961dcd58d223314602b987
Status: Downloaded newer image for registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.9.5-filebeat
enable pilot: filebeat
use default output
DEBU[0000] 72c3eb36e84c2a52f4b309b6c700401e62f8357ba757c06406aa6f3d4aabc519 has not log config, skip
DEBU[0000] b13742566befe366d6676e2d96f8f5641d5947aaf0e481683613d6cedb268066 has not log config, skip
DEBU[0000] fdbaad5815ea4630e56ffb6e3e02e04a0a33a8a2fb26c7635c470b10c440d454 has not log config, skip
DEBU[0000] 84b359b1f8800330748903eb7e091c1020b9732aa714e70a971ad7c9beb1eb15 has not log config, skip
DEBU[0000] 729e8ecad43f02105142bfbd447613766b3661236554b7048b58be13f0c57b6a has not log config, skip
DEBU[0000] b9fac7428bb6e75bd6ccbdabe5a76f5e7d348e5a004cb160e9063d52da34c927 has not log config, skip
DEBU[0000] ce02ee3db38462779f55a2d05c413365465d636e271a9c0881acc32e36981d16 has not log config, skip
DEBU[0000] 857e94936233dc02c5dae6c866ddfb0f3a2877a487079c0b7c2085bfb43fc947 has not log config, skip
INFO[0000] starting filebeat
INFO[0000] filebeat started: 33
INFO[0000] Reload gorouting is ready
INFO[0000] filebeat watcher start
```

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
```

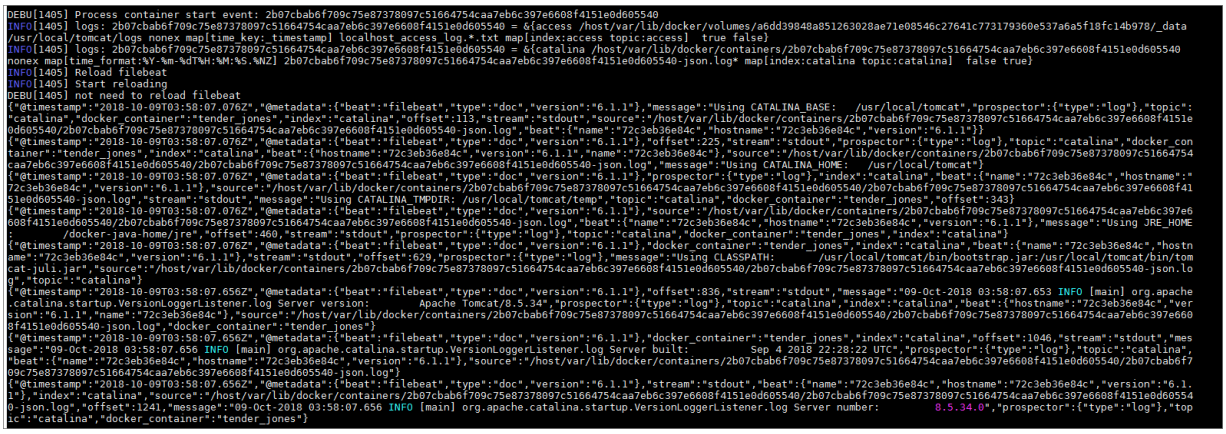
说明：

- `aliyun.logs.catalina=stdout` 告诉 log-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。

说明：

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 log-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。

log-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 log-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 log-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 log-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 log-pilot，先关掉，然后使用下面的命令启动。

说明:

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 \
    --privileged \
    -e FLUENTD_OUTPUT=elasticsearch \
    -e ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST} \
    -e ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT} \
    registry.cn-hangzhou.aliyuncs.com/acs-sample/log-pilot:0.9.5-
filebeat
```

相比前面启动 log-pilot 的方式，这里增加了三个环境变量：

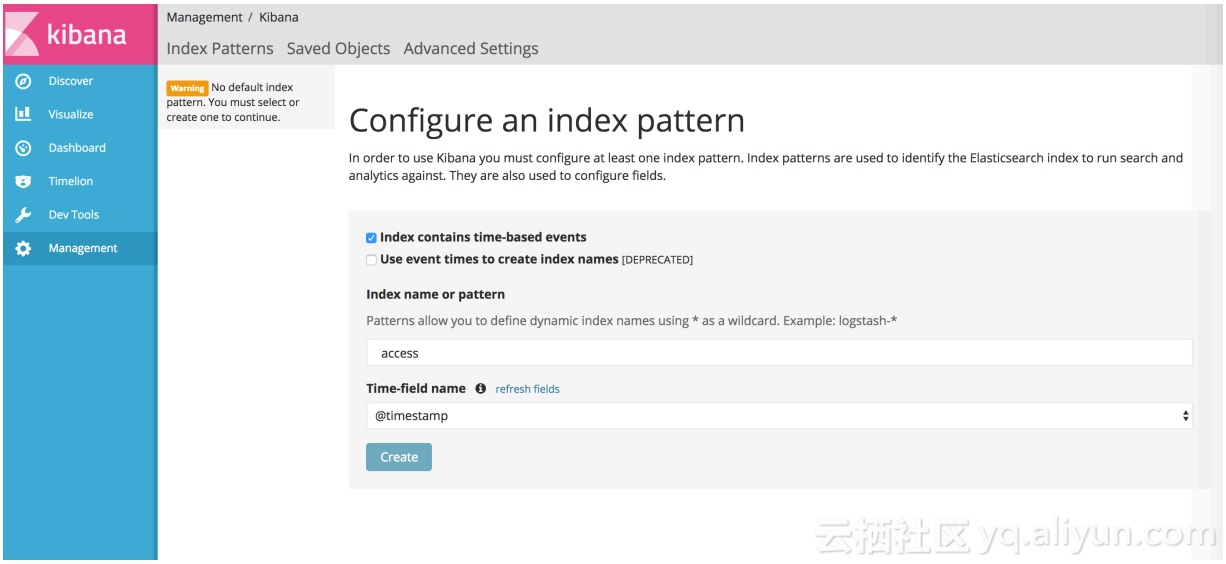
- `FLUENTD_OUTPUT=elasticsearch`：把日志发送到 ElasticSearch。
- `ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST}`：ElasticSearch 的域名。
- `ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT}`：ElasticSearch 的端口号。

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。

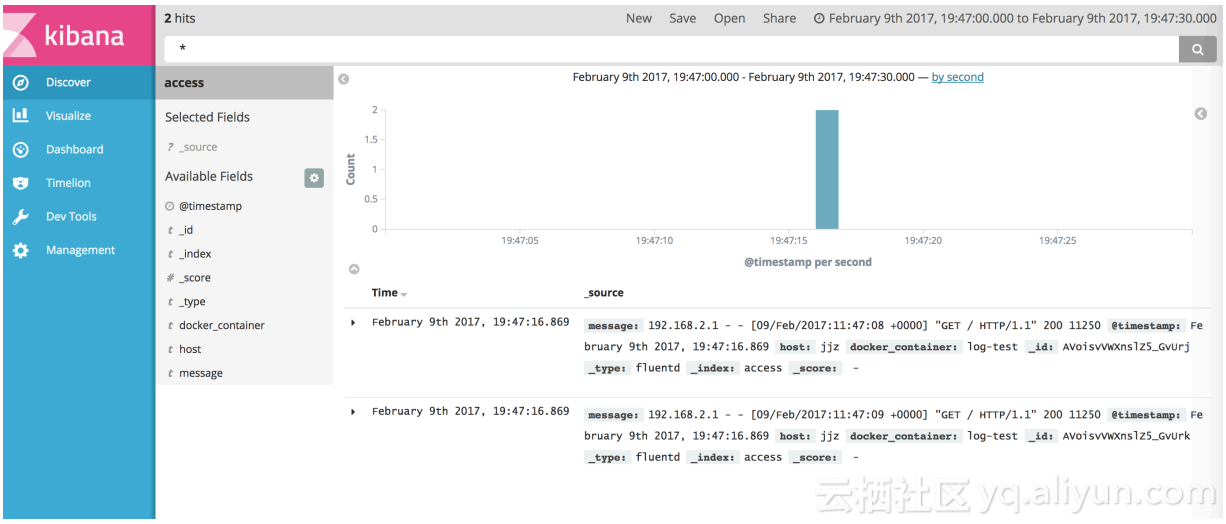
打开 kibana，此时您还看不到新日志，需要先创建 index。log-pilot 会把日志写到 ElasticSearch 特定的 index下，规则如下：

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 log-pilot

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

要在容器服务里运行 log-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/log-pilot:0.9.5-
filebeat
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - /:/host
  privileged: true
  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 log-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 log-pilot

对于大部分用户来说，log-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/log-pilot* 提交 issue。
- 直接改代码，再提 PR。

7 Docker 容器健康检查机制

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。

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。

> 说明：
> 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/healthcheck*。

说明：

- 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。

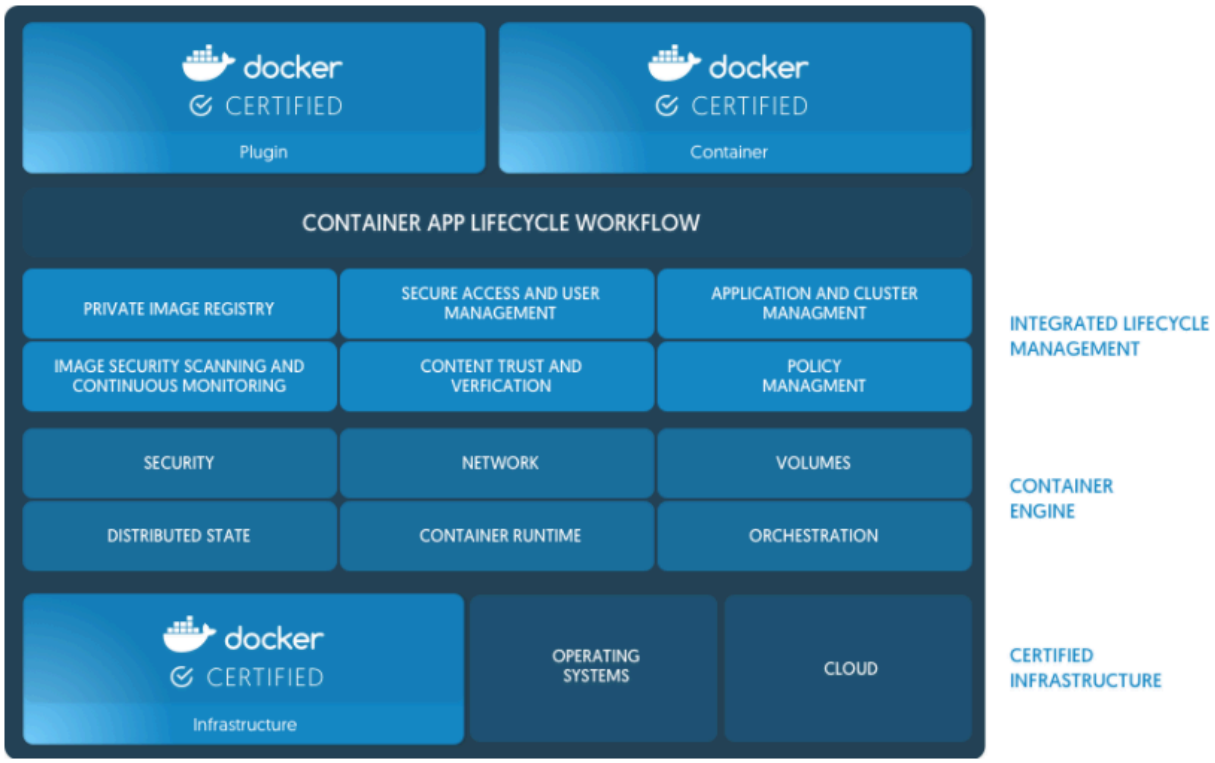
8 一键部署 Docker Datacenter

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。

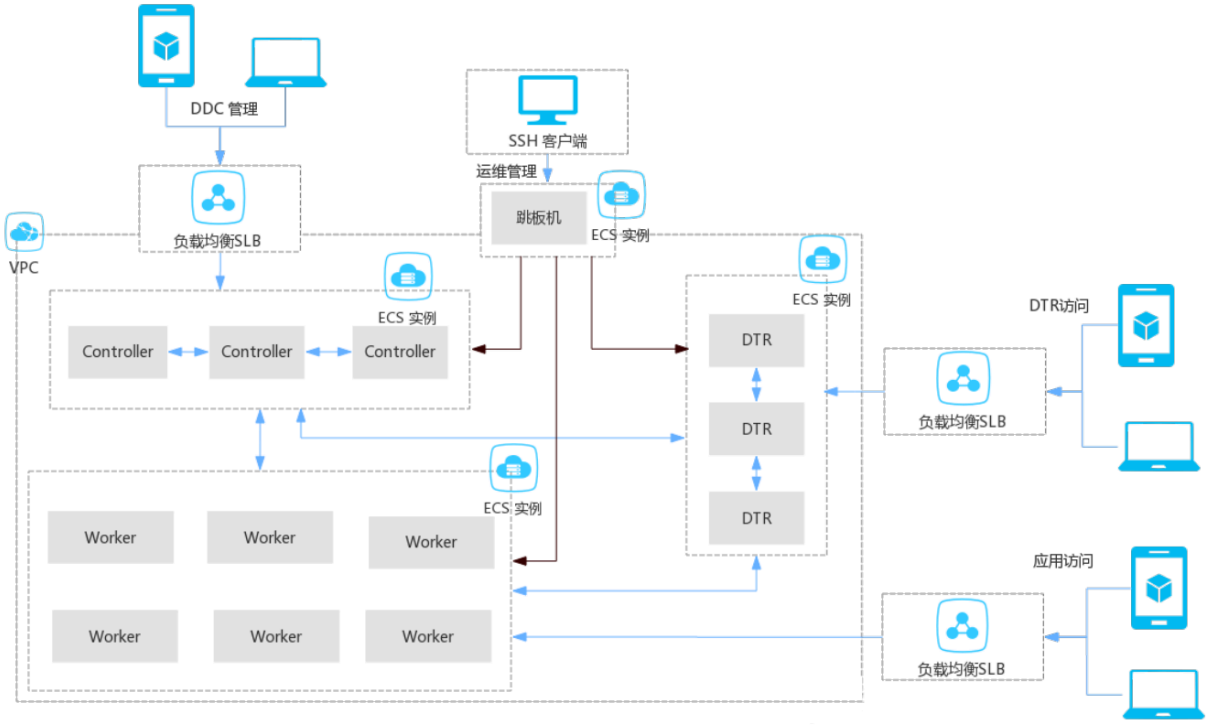
DDC 在 Docker 官网的地址为 https://www.docker.com/products/docker-datacenter。



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。

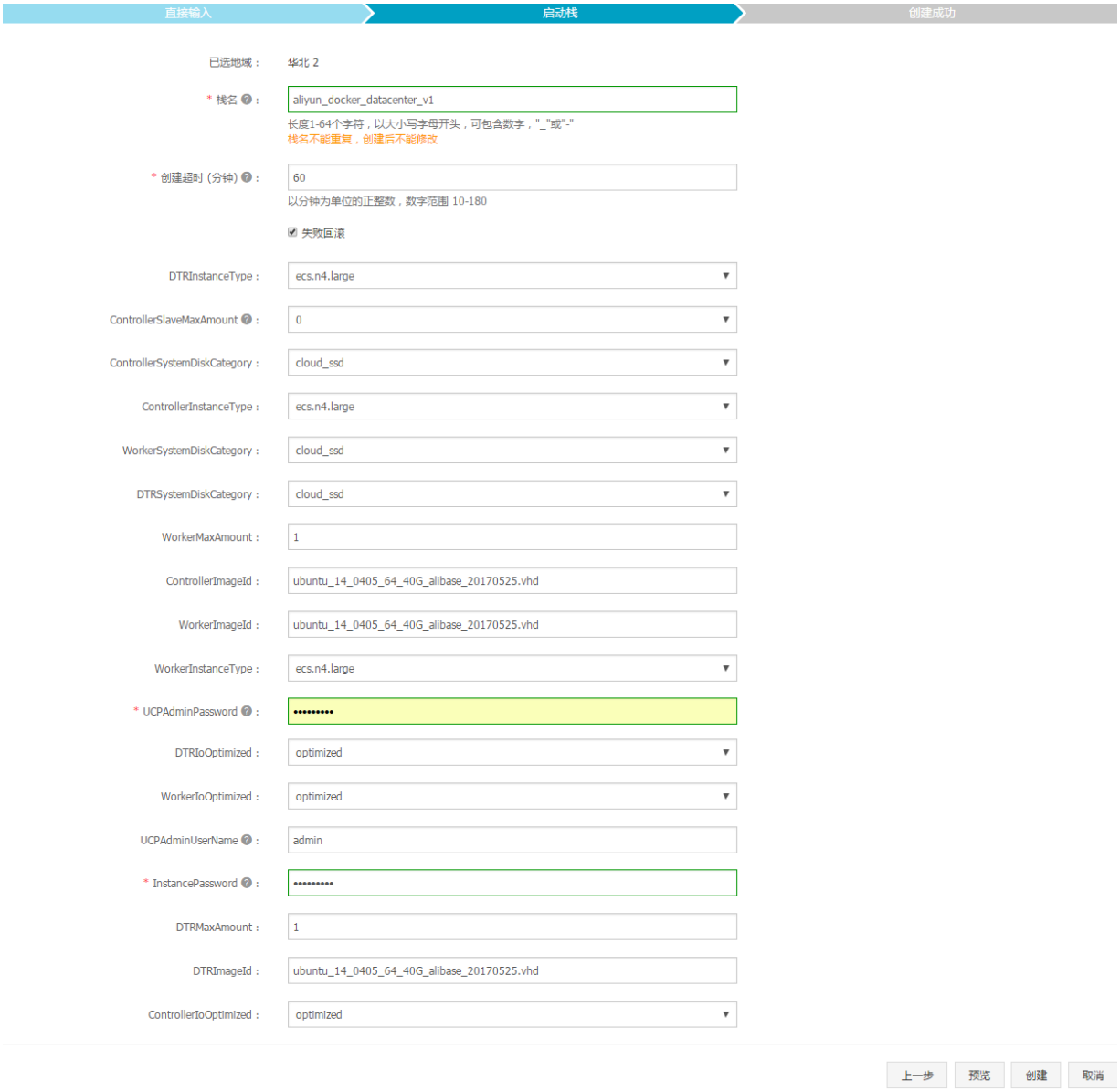
DDC 一键部署

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。

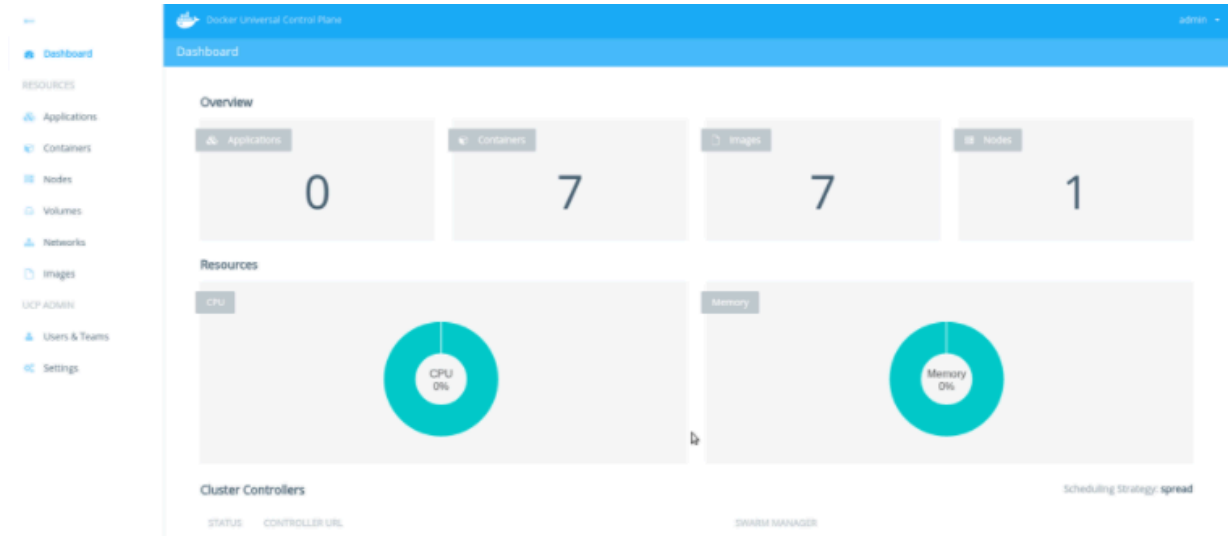


DDC 访问

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。

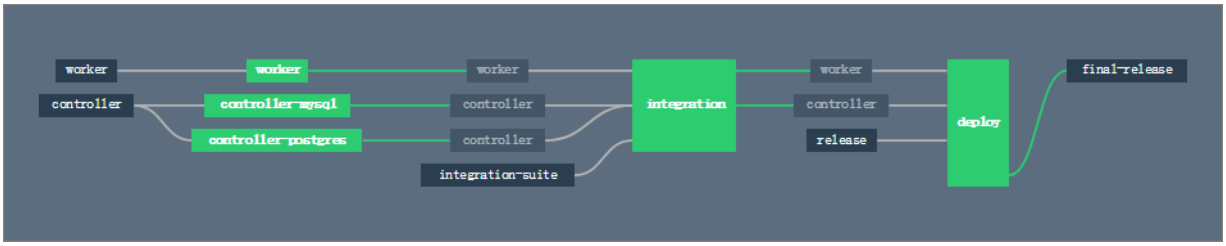
在浏览器中输入 UCP 的地址就会显示 UCP 的访问页面，输入在安装 UCP 时创建的管理账号和密码，系统会提示导入 Licence 文件，把准备好的 Licence 导入，即可进入 UCP 的控制界面。


Docker Universal Control Plane
admin
Dashboard
Dashboard
Applications
Containers
Nodes
Volumes
Networks
Images
Users & Teams
Settings
Overview
Applications
0
Containers
7
Images
7
Nodes
1
Resources
CPU
CPU
0%
Memory
Memory
0%
Cluster Controllers
Scheduling Strategy: spread
STATUS
CONTROLLER URL
SWARM MANAGER

9 在容器服务上轻松搭建 Concourse CI

Concourse CI 是一款 CI/CD 工具，它的魅力在于极简设计，被广泛应用于 Cloud Foundry 各个模块的 CI/CD。Concourse CI 官方提供了标准的 Docker 镜像，您可以通过阿里云容器服务快速部署一套 Concourse CI 应用。

如果您还不了解 Concourse CI 这款工具，您可以先了解一下 Concourse 的原理。请参见 *concourse* 官网。



创建 Swarm 集群

登录 容器服务管理控制台 创建一个集群。本示例中以包含 1 个节点，网络类型为 VPC 的 swarm 集群为例进行说明。

关于如何创建集群，请参见 创建集群。

> 说明:
> 在创建容器集群时，请保留 EIP 。因为需要为 Concourse 配置外部 URL 地址，让您从本机访问 concourse 的 web 服务。

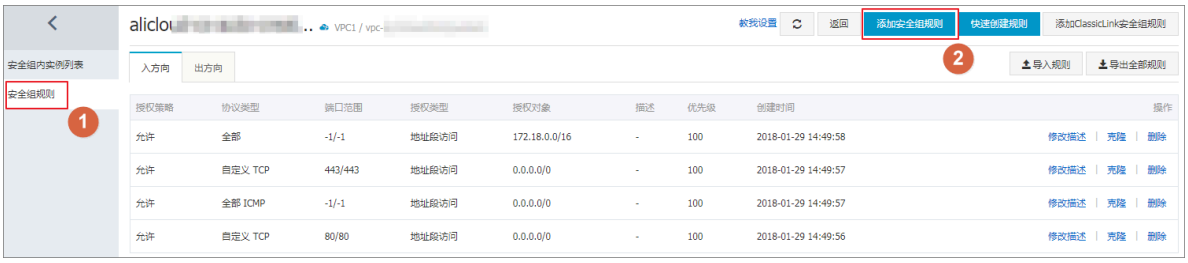
配置安全组规则

Concourse 的组件 ATC 默认监听 8080 端口，因此您需要为集群的安全组配置 8080 端口的入网权限。

1. 在 容器服务管理控制台 的 swarm 集群列表页面，选择前面创建的集群并单击右侧的管理，进入集群详情页面。

2. 在集群的基本信息中，单击安全组的ID，跳转到集群的安全组页面。

3. 单击左侧导航栏中的安全组规则，并单击页面右上角的添加安全组规则。



4. 为安全组配置 8080 端口的入网权限并单击确定。

在 ECS 节点上创建 key

为了安全运行 Concourse，您需要生成 3 个私有 key。

1. 登录到 ECS 节点上，在根目录下，创建目录 *keys/web* 和 *keys/worker* 。您可以执行以下命令，快速创建这两个目录。

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

2. 执行以下命令，生成 3 个私有 key。

```
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 ''
```

3. 将证书拷贝到相应的目录下。

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

部署 Concourse CI

1. 登录 容器服务管理控制台。
2. 在 Swarm 菜单下，单击左侧导航栏中的 配置项，创建配置文件，变量名称为 CONCOURSE_EXTERNAL_URL，变量值为 `http://your-ecs-public-ip:8080`。

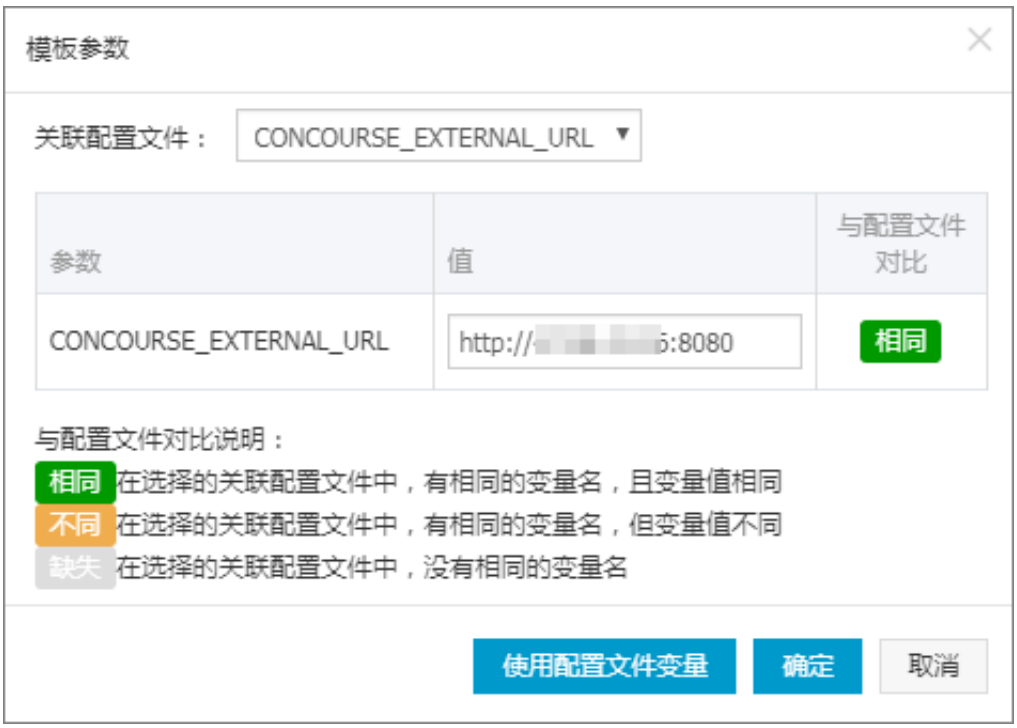
3. 单击左侧导航栏的 应用，选择本示例中使用的集群，并单击 创建应用。
4. 填写应用的基本信息，并选择 使用编排模板创建应用。模板如下。

```
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
conocurse-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
```

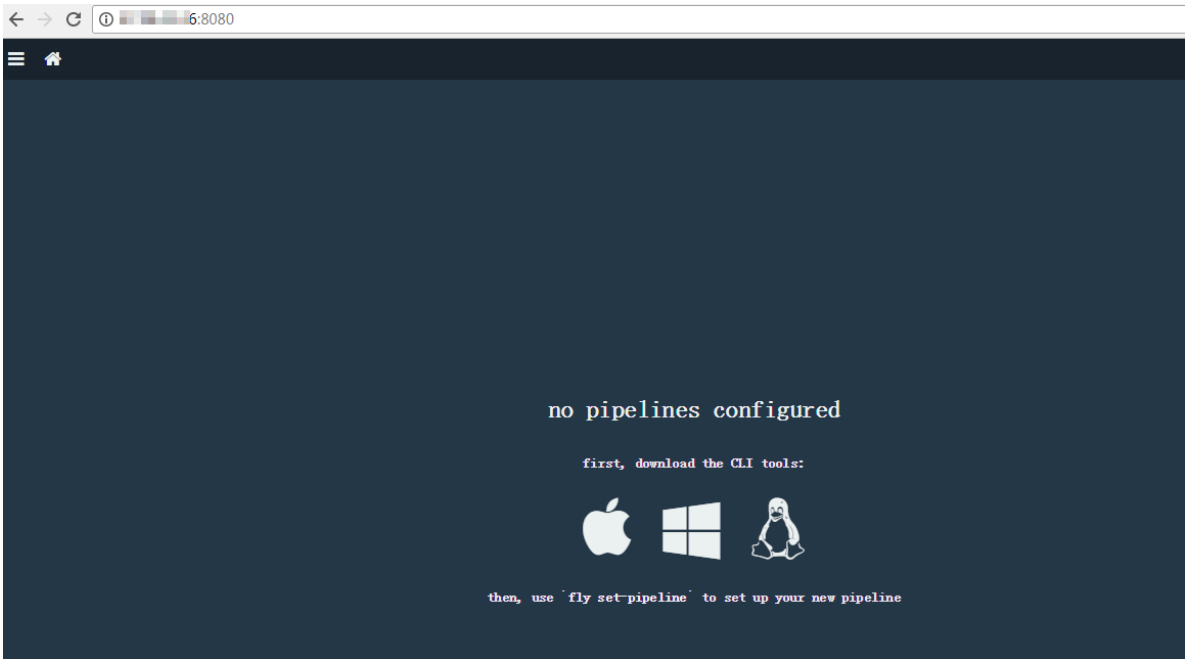
```
    dns: 8.8.8.8
```

5. 单击创建并部署时，会提示您配置模板参数，您需要选择关联配置文件，再单击使用配置文件变量。



应用创建之后会启动如下 3 个服务。

至此，Concourse CI 部署完成，在浏览器输入`http://your-ecs-public-ip:8080`即可进行访问。



运行 CI 任务（Hello world）

1. 在上一步浏览器中下载对应您操作系统的 CLI，然后安装 CLI 客户端，本例以 ECS（Ubuntu16.04）为例。
2. 针对 Linux 和 Mac OS X 系统，首先需要给下载的 FLY CLI 文件添加执行权限，然后安装到系统并添加到 `$PATH`中：

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

3. 安装之后可以查看一下版本。

```
$fly -v
 3.4.0
```

4. 连接 Target，用户名和密码默认是 concourse 和 changeme。

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

5. 将下面的配置模板保存为 `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!"]
```

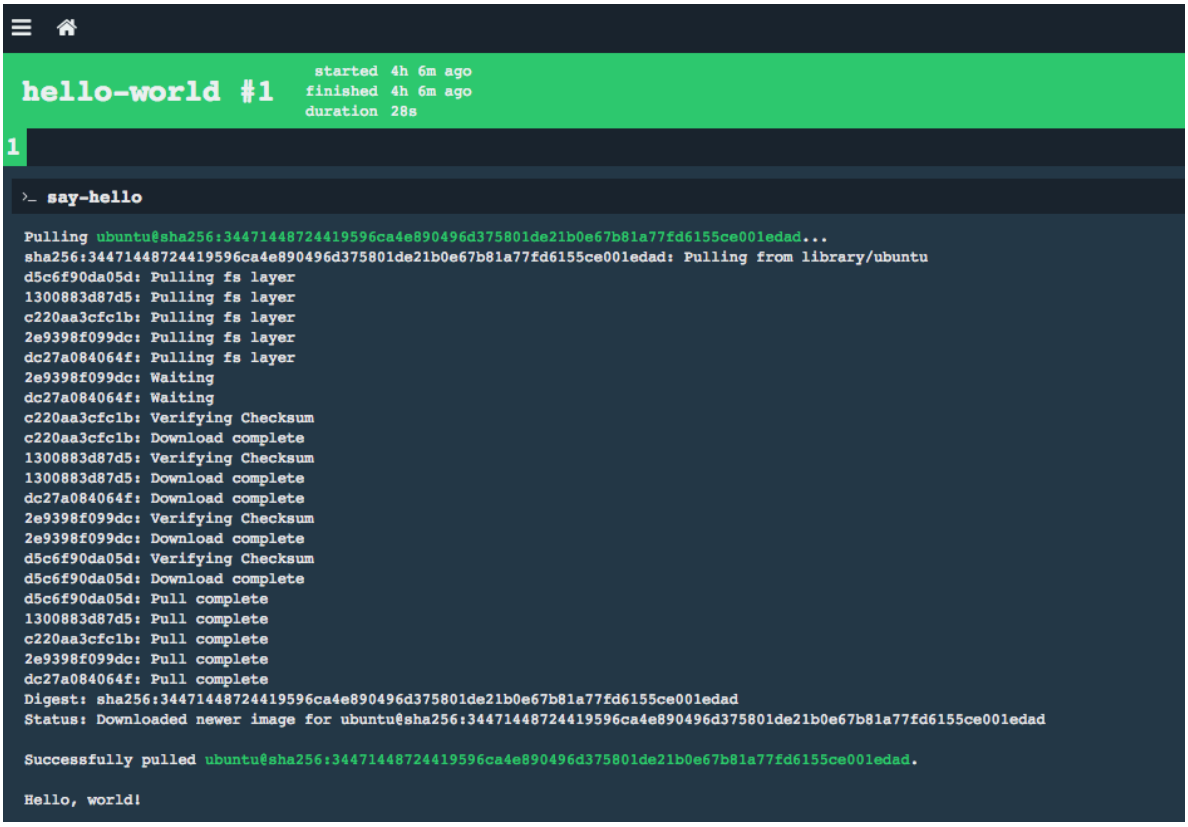
6. 注册任务。

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

7. 启动任务。

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

成功执行的界面如下。



更多关于 Concourse CI 的特性，请参见 *Concourse CI* 项目。

10 Chef实现Dokcer和WebServer自动化部署

Chef 是一个自动化部署框架，结合阿里云容器服务，可实现定制化的自动部署工作。首先参考Chef官网了解一些基本概念，诸如 Cookbook、 Recipe 、Chef Workstation、 Chef Server、 Chef Nodes，以便快速入门。

前提条件

- 您已成功创建一个Swarm集群，该集群保留EIP。
- 准备一个本地Linux环境，本示例是Ubuntu 16.04，您可根据本地环境，参考https://downloads.chef.io/chefdk/获取相应的ChefDK。
- 您需要登录Chef官网，注册一个账号，创建一个Organization，本例中为example。

在Linux上安装chef工作站

您需要前往chef官网下载符合本地Linux环境的ChefDK，本例中使用Ubuntu 16.04对应的ChefDK。

首先在*/home*目录下创建一个*chef-repo*文件夹。

```
mkdir /home/chef-repo
```

进入*chef-repo*目录，使用curl命令下载ChefDK程序包，并进行安装。

```
cd  /home/chef-repo
curl  -O https://packages.chef.io/files/stable/chefdk/3.0.36/ubuntu/16
.04/chefdk_3.0.36-1_amd64.deb
dpkg  -i  chefdk_3.0.36-1_amd64.deb
```

然后您需要进行大量的Chef安装配置，您在安装过程中如果遇到问题，可参见Chef官方文档进行排查。

验证Chef

```
chef verify                                                    #验证
ChefDK的组件是否正常
chef --version                                                  #查看chef
版本
```

设置Chef 环境变量

设置Chef相关的环境变量，如：GEM_ROOT、GEM_HOME、GEM_PATH。

```
export GEM_ROOT="/opt/chefdk/embedded/lib/ruby/gems/2.1.0"
export GEM_HOME="/root/.chefdk/gem/ruby/2.1.0"
export GEM_PATH="/root/.chefdk/gem/ruby/2.1.0:/opt/chefdk/embedded/lib/
ruby/gems/2.1.0"
```

此外，如果你的系统上已经安装了ruby，你需要更新与ruby相关的PATH变量。

```
export PATH="/opt/chefdk/bin:/root/.chefdk/gem/ruby/2.1.0/bin:/opt/
chefdk/embedded/bin:/opt/chefdk/bin:/root/.chefdk/gem/ruby/2.1.0/bin:/
opt/chefdk/embedded/bin:/opt/chefdk/bin:/root/.chefdk/gem/ruby/2.1.0/
bin:/opt/chefdk/embedded/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin
:/usr/bin:/root/bin"
```

设置访问Chef的Firewalld规则

为了访问Chef服务器上的Chef Manage GUI，添加以下firewalld规则，开放Chef服务器上的相应端口。

```
firewall-cmd --direct  --add-rule ipv4 \
filter INPUT_direct 0 -i eth0 -p tcp \
 --dport 443 -j ACCEPT

firewall-cmd --direct  --add-rule ipv4 \
filter INPUT_direct 0 -i eth0 -p tcp \
 --dport 80 -j ACCEPT

firewall-cmd --direct  --add-rule ipv4 \
filter INPUT_direct 0 -i eth0 -p tcp \
 --dport 9683 -j ACCEPT

firewall-cmd --reload
```

从Chef Manage GUI下载Starter Kit

登录*Chef Manage GUI*，单击Administration选项，从列表中选择organization。此例中，organization为example，选中organization之后，点击左侧菜单中的Starter Kit，将chef-starter.zip文件下载到本地机器。

将*chef-starter.zip*文件传输到本地Linux下的Chef工作站，并解压到*home/chef-repo*目录下。

```
# cd /home/chef-repo
unzip chef-starter.zip
```

下载Chef服务器的SSL证书

证书会下载*#chef-repo/.chef/trusted_certs*目录中。

```
# cd ~/chef-repo
# knife ssl fetch

WARNING: Certificates from api.chef.io will be fetched and placed in
your trusted_cert
directory (/root/chef-repo/.chef/trusted_certs).

Knife has no means to verify these are the correct certificates. You
should
verify the authenticity of these certificates after downloading.
```

```
Adding certificate for wildcard_opscode_com in /root/chef-repo/.chef/
trusted_certs/wildcard_opscode_com.crt
Adding certificate for DigiCert_SHA2_Secure_Server_CA in /root/chef-
repo/.chef/trusted_certs/DigiCert_SHA2_Secure_Server_CA.crt
```

验证Chef工作站是否安装成功

配置成功后，执行以下命令，可看到我们预先创建的Organization，说明我们已成功连接到工作站。

```
# cd ~/chef-repo

# knife client list
example-validator
```

创建实现Docker自动初始化的CookBook

1. **在Chef工作站上创建一个CookBook。**
   - **在chef-repo/cookbooks目录下，执行以下命令，创建一个名为docker_init的CookBook。**

     ```
     chef generate cookbook docker_init
     ```

   - **进入*chef-repo/cookbooks/docker_init/recipe/*目录，找到default.rb文件，进行配置。该示例用于在Ubuntu中启动最新的Docker版本。**

     ```
     apt_update


     package 'apt-transport-https'
     package 'ca-certificates'
     package 'curl'
     package 'software-properties-common'


     execute 'apt-key' do
     command 'apt-key fingerprint 0EBFCD88'
     end


     execute 'apt-repo' do
     command 'add-apt-repository "deb [arch=amd64] https://download.
     docker.com/linux/ubuntu/dists/xenial/stable/"'


     end


     execute 'apt-repo' do
     command 'apt-get update'
     end


     execute 'apt-repo' do
     command 'apt-get install docker-ce -y --allow-unauthenticated'
     ```

```
end


service 'docker' do
action [:start, :enable]
end
```

2. 校验docker_init这个CookBook是否在本地工作。

```
# chef-client --local-mode --runlist 'recipe[docker_init]'

[2018-06-27T15:54:30+08:00] INFO: Started chef-zero at chefzero://
localhost:1 with repository at /root/chef-repo
One version per cookbook


Starting Chef Client, version 14.1.12
[2018-06-27T15:54:30+08:00] INFO: *** Chef 14.1.12 ***
[2018-06-27T15:54:30+08:00] INFO: Platform: x86_64-linux
[2018-06-27T15:54:30+08:00] INFO: Chef-client pid: 2010
[2018-06-27T15:54:30+08:00] INFO: The plugin path /etc/chef/ohai/
plugins does not exist. Skipping...
[2018-06-27T15:54:31+08:00] INFO: Setting the run_list to [#] from
CLI options
[2018-06-27T15:54:32+08:00] INFO: Run List is [recipe[docker_init]]
[2018-06-27T15:54:32+08:00] INFO: Run List expands to [docker_init]
[2018-06-27T15:54:32+08:00] INFO: Starting Chef Run for yxm
[2018-06-27T15:54:32+08:00] INFO: Running start handlers
[2018-06-27T15:54:32+08:00] INFO: Start handlers complete.
resolving cookbooks for run list: ["docker_init"]
[2018-06-27T15:54:32+08:00] INFO: Loading cookbooks [docker_init@0.1
.0]
Synchronizing Cookbooks:
- docker_init (0.1.0)
Installing Cookbook Gems:
Compiling Cookbooks...
Converging 10 resources
Recipe: docker_init::default
* apt_update[] action periodic[2018-06-27T15:54:32+08:00] INFO:
Processing apt_update[] action periodic (docker_init::default line 9
)
....
---- End output of add-apt-repository "deb [arch=amd64] https://
download.docker.com/linux/ubuntu/dists/xenial/stable/" ----
Ran add-apt-repository "deb [arch=amd64] https://download.docker.com
/linux/ubuntu/dists/xenial/stable/" returned 1
```

执行以下命令，检查本地安装的Docker是否升级到最新版本。

```
# docker --version
Docker version 17.06.2-ce, build 2e0fd6f
```

3. 将该CookBook上传到Chef Server。
   - 在Chef工作站，执行以下命令，将名为docker_init的CookBook上传到Chef Server。

     ```
     knife cookbook upload docker_init
     ```

   - 执行以下命令，检验该CookBook是否被成功上传。

     ```
     # knife cookbook list
     docker_init 0.1.0
     ```

4. 将该cookbook导入到阿里云Swarm集群的节点。
   - 在Chef工作站执行以下命令，将docker_init导入充当Chef Node的Swarm集群的节点。

     说明:

     将ADDRESS替换为Swarm集群的ECS节点的EIP， USER是ECS节点登录用户，一般为root，PASSWORD是ECS节点登录密码。若Swarm集群中有多个节点，需要对每个ECS节点执行该命令。

     ```
     # knife bootstrap ADDRESS --ssh-user USER --ssh-password 'PASSWORD
     ' --sudo --use-sudo-password --node-name node1-ubuntu --run-list '
     recipe[docker_init]'

     Creating new client for node1-ubuntu
     Creating new node for node1-ubuntu
     Connecting to 121.196.219.18
     ...
     https://download.docker.com/linux/ubuntu/dists/xenial/stable/"
      ----
     121.196.219.18 Ran add-apt-repository "deb [arch=amd64] https://
     download.docker.com/linux/ubuntu/dists/xenial/stable/" returned 1
     ```

   - 登录每个ECS节点，检查每个节点上安装的Docker是否已更新到最新版本。执行`docker -- version`命令进行检验。

至此，通过Chef自动部署系统，实现对阿里云容器集群Docker的版本更新。

创建自动化部署Web Server的CookBook

1. 在Chef工作站创建一个新的CookBook。
   - 在chef-repo/cookbooks目录下，执行以下命令，创建名为web_init的CookBook。

     ```
     chef generate cookbook web_init
     ```

   - 进入*chef-repo/cookbooks/web_init/recipe/*目录，找到default.rb文件，进行配置。

     ```
     execute 'apt-repo' do
     command 'apt-get -y install apache2 --allow-unauthenticated'
     end


     service 'apache2' do
     action [:start, :enable]
     end


     file '/var/www/html/index.html' do
     content '
     hello world
     '
     end


     service 'iptables' do
     action :stop
     end
     ```

2. 检验该CookBook是否在本地工作。
   - 执行`curl http://localhost:80`命令，检查web_init是否在本机工作。
   - 在Chef 工作站，将web_init这个CookBook上传到Chef Server。

     ```
     knife cookbook upload web_init
     ```

3. 将该cookbook导入到阿里云Swarm集群的节点。

   在Chef工作站执行以下命令，将web_init导入充当Chef Node的Swarm集群的节点。

    **说明:**

将ADDRESS替换为Swarm集群的ECS节点的EIP， USER是ECS节点登录用户，一般为root，PASSWORD是ECS节点登录密码。若Swarm集群中有多个节点，需要对每个ECS节点执行该命令。

```
knife bootstrap ADDRESS --ssh-user USER --ssh-password 'PASSWORD' --sudo --use-sudo-password --node-name node1-ubuntu --run-list 'recipe[web_init]'
```

4. 在阿里云Swarm集群中，检查Web Server是否成功启动。登录阿里云Swarm集群的节点。
   - 执行`systemctl status apache2.service`，检查apache2 是否正常运行。
   - 在浏览器中访问`http://ADDRESS:80`，查看浏览器屏幕是否输出`hello world`。

   说明:
   ADDRESS是指节点的EIP。