ALIBABA CLOUD

阿里云

容器服务

开发指南

文档版本：20211028

阿里云

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
危险	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。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会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，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	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。	单击**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。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	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。	`switch {active|stand}`

目录

1.Swarm SDK参考

1.1. 集群操作SDK

1.1.1. JAVA SDK

您可以基于阿里云 SDK 通过编写代码的方式调用阿里云 API，进而实现对阿里云产品和服务的灵活部署和快速操作。

> **说明** 使用过程中您会需要 AccessKey。您可以在 云账号 AccessKey 管理页面 创建并管理 AccessKey。

环境准备

通过以下地址查询下载最新版本 SDK：https://oss.sonatype.org/#nexus-search;gav~com.aliyun~aliyun-java-sdk-*~~~

从 Github 上下载源代码：https://github.com/aliyun/aliyun-openapi-java-sdk/

Maven 方式

```
<repositories>
    <repository>
      <id>sonatype-nexus-staging</id>
      <name>Sonatype Nexus Staging</name>
      <url>https://oss.sonatype.org/service/local/staging/deploy/maven2/</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </repository>
  </repositories>
  <dependencies>
    <dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-core</artifactId>
      <version>2.3.3</version>
    </dependency>
    <dependency>
      <groupId>com.aliyun</groupId>
      <artifactId>aliyun-java-sdk-cs</artifactId>
      <version>2.0.2</version>
    </dependency>
  </dependencies>
```

Sample

```
public static void main(String[] argc) throws Exception {
    String accessKeyID = "xx";
    String accessKeySecret = "xx";
    String region="cn-shenzhen"; // or other
    DescribeApiVersionRequest describeApiVersionRequest = new DescribeApiVersionRequest();
    IClientProfile profile = DefaultProfile.getProfile(region, accessKeyID, accessKeySecret);
    IAcsClient client = new DefaultAcsClient(profile);
    try {
      HttpResponse httpResponse
          = client.doAction(describeApiVersionRequest);
      System.out.println(httpResponse.getUrl());
      System.out.println(new String(httpResponse.getContent()));
    } catch (ClientException e) {
      e.printStackTrace();
    }
}
```

1.1.2. PHP SDK

您可以基于阿里云 SDK 通过编写代码的方式调用阿里云 API，进而实现对阿里云产品和服务的灵活部署和快速操作。

获取 RAM 子账号 AK 密钥

使用过程中您会需要 AccessKey。为了保证云服务的安全，您需要创建一个能访问容器服务资源的 RAM 子账号，获取该子账号的 AK 密钥，并使用这个 RAM 子账号和 SDK 管理容器服务。

以下是获取 RAM 子账号 AK 密钥的操作步骤：

1. 创建 RAM 用户并创建该用户的 AK 密钥。
2. 给 RAM 用户授权，授予 RAM 子账号管理容器服务资源的权限。

环境准备

PHP 5.3 或更高版本

SDK 下载

从 Github 上下载 PHP SDK 的源代码：https://github.com/AliyunContainerService/aliyun-openapi-php-sdk

Sample

https://github.com/AliyunContainerService/aliyun-openapi-php-sdk/blob/master/aliyun-php-sdk-demo/CsDemo.php

2.通过 CLI 使用容器服务

2.1. 概述

阿里云命令行工具是用 Go 语言编写的，基于阿里云 OpenAPI 打造的，用于管理阿里云资源的工具。通过下载和配置该工具，您可以在一个命令行方式下使用多个阿里云产品。

容器服务是 RESTful 风格的 API。目前容器服务支持 Swarm 和 Kubernetes 两种调度方式，下面是目前容器服务开放的 API 列表。

> **说明** 关于阿里云容器服务已支持的 API，参见 容器服务 API 参考。

API 接口	解释	适用范围
查看所有集群实例	查看您在容器服务中创建的所有集群（包括 Swarm 和 Kubernetes 集群）。	Swarm 集群和 Kubernetes 集群
查看集群实例	根据集群 ID，查看集群的详细信息。	Swarm 集群和 Kubernetes 集群
创建集群实例	创建一个新的集群实例，并新建指定数量的节点。	Swarm 集群和 Kubernetes 集群
扩容集群	增加集群中节点的数量。	Swarm 集群和 Kubernetes 集群
添加已有实例	添加已有实例到集群。	Swarm 集群和 Kubernetes 集群
移除集群内节点	根据集群 ID，以及节点 IP 从容器集群中移除节点。	Swarm 集群
查看镜像列表	查看目前所支持的地域下支持的镜像列表。	Swarm 集群
重置节点	重置集群中的某个节点。	Swarm 集群
删除集群实例	根据集群 ID，删除集群实例，并释放集群所有节点资源。	Swarm 集群和 Kubernetes 集群
获取集群证书	根据集群 ID，获取集群的证书信息。	Swarm 集群

2.2. 查看所有集群实例

查看您在容器服务中创建的所有集群，包括 Swarm 和 Kubernetes 集群。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群和 Kubernetes 集群。

API请求响应

请求格式

```
aliyun cs GET /clusters
```

响应结果

```
[
  {
    "agent_version": "string",
    "cluster_id": "string",
    "created": "datetime",
    "external_loadbalancer_id": "string",
    "master_url": "string",
    "name": "string",
    "network_mode": "string",
    "region_id": "string",
    "security_group_id": "string",
    "size": "numbers",
    "state": "string",
    "updated": "datetime",
    "vpc_id": "string",
    "vswitch_id": "string"
  }
]
```

2.3. 查看集群实例

根据集群 ID，查看集群的详细信息。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群和 Kubernetes 集群。

API请求响应

请求格式

```
aliyun cs GET /clusters/<cluster_id>
```

响应结果

```
{
  "agent_version": "string",
  "cluster_id": "string",
  "created": "datetime",
  "external_loadbalancer_id": "string",
  "master_url": "string",
  "name": "string",
  "network_mode": "string",
  "region_id": "string",
  "security_group_id": "string",
  "size": "numbers",
  "state": "string",
  "updated": "datetime",
  "vpc_id": "string",
  "vswitch_id": "string"
}
```

2.4. 创建集群实例

创建一个新的集群实例，并新建指定数量的节点。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群和 Kubernetes 集群。

API请求响应

请求格式

```
aliyun cs  POST /clusters --header "Content-Type=application/json" --body "$(cat create.json)"
```

参数说明：

- `--header` 需要指定 Content-Type 为 *application/json*。
- `--body` 是要发送给服务端的 body 内容，可以从本地文件读取，需要是有效的 JSON 格式。`create.json` 的内容如下所示。

swarm 集群

```
{
"password": "ECS机器SSH密码",
"region_id": "地域ID",
"instance_type": "实例规格",
"name": "集群名称",
"size": 节点数量,
"network_mode": "网络类型，目前仅支持vpc",
"vpc_id": "VPC示例ID",
"vswitch_id": "VPC下的交换机ID",
"subnet_cidr": "容器CIDR",
"data_disk_category": "数据盘类型",
"data_disk_size": 数据盘大小,
"need_slb": 是否默认创建SLB,
"io_optimized":"是否IO优化，VPC下目前默认为IO优化",
"ecs_image_id": "镜像ID",
"release_eip_flag": "是否需要在集群配置完成后释放EIP"
}
```

Kubernetes集群(单可用区)

```
{
"disable_rollback": "失败是否回滚",
"name": "集群名称",
"timeout_mins": 集群创建超时时间,
"cluster_type": "Kubernetes",
"region_id": "地域",
"vpcid": "VPC ID",
"zoneid": "可用区",
"vswitchid": "交换机ID",
"container_cidr": "容器POD CIDR",
"service_cidr": "服务CIDR",
"ssh_flags": "是否开放公网SSH登录",
"cloud_monitor_flags":"是否安装云监控插件",
"login_password": "节点SSH登录密码，和key_pair二选一",
"key_pair":"keypair名称，和login_password 二选一",
"master_instance_charge_type":"Master实例付费类型，PostPaid|PrePaid",
"master_period_unit":"包年包月单位，Month,Year，只有在PrePaid下生效",
"master_period":"包年包月时长，只有在PrePaid下生效",
"master_auto_renew":"Master节点是否自动续费",
"master_auto_renew_period":"Master节点续费周期",
"master_instance_type": "Master实例规格",
"master_system_disk_category": "Master系统盘类型",
"master_system_disk_size":"Master节点系统盘大小",
"master_data_disk":"Master节点是否挂载数据盘",
"master_data_disk_category":"Master节点数据盘类型",
"master_data_disk_size":"Master节点数据盘大小",
"worker_instance_charge_type":"Worker节点付费类型PrePaid|PostPaid",
"worker_period_unit":"包年包月单位，Month,Year，只有在PrePaid下生效",
"worker_period":"包年包月时长，只有在PrePaid下生效",
"worker_auto_renew":"Worker节点自动续费true|false",
"worker_auto_renew_period":"Worker节点续费周期",
"worker_instance_type": "Worker实例规格",
"worker_system_disk_category": "Worker系统盘类型",
"worker_system_disk_size": "Worker节点系统盘大小",
"worker_data_disk":"Worker节点是否挂载数据盘",
"worker_data_disk_category":"Worker节点数据盘类型",
"worker_data_disk_size":"Worker节点数据盘大小",
"num_of_nodes": "Worker节点数",
"snat_entry": 是否配置SNATEntry,
"public_slb":"是否创建公网API Server对应的SLB"
}
```

Kubernetes集群(多可用区)

```
{
"disable_rollback":"失败是否回滚",
"name": "集群名称",
"timeout_mins": 集群创建超时时间,
"cluster_type": "Kubernetes",
"region_id": "地域"
"multi_az": true,
"vpcid": "VPC ID ",
"container_cidr": "容器 CIDR",
"service_cidr": "服务 CIDR",
"vswitch_id_a": "第一个可用区交换机ID",
"vswitch_id_b": "第二个可用区交换机ID",
"vswitch_id_c": "第三个可用区交换机ID",
"master_instance_type_a": "第一个可用区Master节点实例规格",
"master_instance_type_b": "第二个可用区Master节点实例规格",
"master_instance_type_c": "第三个可用区Master节点实例规格",
"master_instance_charge_type":"Master实例付费类型，PostPaid|PrePaid",
"master_period_unit":"包年包月单位，Month,Year，只有在PrePaid下生效",
"master_period":"包年包月时长，只有在PrePaid下生效",
"master_auto_renew":"Master节点是否自动续费",
"master_auto_renew_period":"Master节点续费周期",
"master_system_disk_category": "Master节点系统盘类型",
"master_system_disk_size": "Master节点系统盘大小",
"master_data_disk":"Master节点是否挂载数据盘",
"master_data_disk_category":"Master节点数据盘类型",
"master_data_disk_size":"Master节点数据盘大小",
"worker_instance_type_a": "第一个可用区Worker节点实例规格",
"worker_instance_type_b": "第二个可用区Worker节点实例规格",
"worker_instance_type_c": "第三个可用区Worker节点实例规格",
"worker_instance_charge_type":"Worker节点付费类型PrePaid|PostPaid",
"worker_period_unit":"包年包月单位，Month,Year，只有在PrePaid下生效",
"worker_period":"包年包月时长，只有在PrePaid下生效",
"worker_auto_renew":"Worker节点自动续费true|false",
"worker_auto_renew_period":"Worker节点续费周期",
"worker_system_disk_category": "Worker节点系统盘类型",
"worker_system_disk_size": "Worker节点系统盘大小",
"worker_data_disk":"Worker节点是否挂载数据盘",
"worker_data_disk_category":"Worker节点数据盘类型",
"worker_data_disk_size":"Worker节点数据盘大小",
"num_of_nodes_a": "第一个可用区Worker节点数",
"num_of_nodes_b": "第二个可用区Worker节点数",
"num_of_nodes_c": "第三个可用区Worker节点数",
"ssh_flags": "是否开放公网SSH 登录",
"login_password": "SSH 登录密码",
"cloud_monitor_flags":"是否安装云监控插件",
"public_slb":"是否创建公网API Server对应的SLB"
}
```

托管Kubernetes集群

```
{
"disable_rollback": "失败是否回滚",
"name": "集群名称",
"timeout_mins": 集群创建超时时间,
"cluster_type": "ManagedKubernetes",
"region_id": "地域，目前仅支持北京和杭州地域(cn-beijing,cn-hangzhou)",
"vpcid": "VPC ID",
"zoneid": "可用区",
"vswitchid": "交换机ID",
"container_cidr": "容器POD CIDR",
"service_cidr": "服务CIDR",
"cloud_monitor_flags":"是否安装云监控插件",
"login_password": "节点SSH登录密码，和key_pair二选一",
"key_pair":"keypair名称，和login_password 二选一",
"worker_instance_charge_type":"Worker节点付费类型PrePaid|PostPaid",
"worker_period_unit":"包年包月单位，Month,Year，只有在PrePaid下生效",
"worker_period":"包年包月时长，只有在PrePaid下生效",
"worker_auto_renew":"Worker节点自动续费true|false",
"worker_auto_renew_period":"Worker节点续费周期",
"worker_instance_type": "Worker实例规格",
"worker_system_disk_category": "Worker系统盘类型",
"worker_system_disk_size": "Worker节点系统盘大小",
"worker_data_disk":"是否挂载数据盘 true|false",
"worker_data_disk_category":"数据盘类型",
"worker_data_disk_size":"数据盘大小",
"num_of_nodes": "Worker节点数",
"snat_entry": 是否配置SNATEntry,
}ntry": 是否配置SNATEntry,
        }
```

响应结果

```
{
  "cluster_id": "c61cf530524474386a7ab5a1c192a0d57",
  "request_id": "348D4C9C-9105-4A1B-A86E-B58F0F875575",
  "task_id": "T-5ad724ab94a2b109e8000004"
}
```

2.5. 扩容集群

增加集群中节点的数量。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群和 Kubernetes 集群。

API 请求响应

请求格式

```
aliyun cs  PUT /clusters/<cluster_id> --header "Content-Type=application/json" --body "$(cat scale.json)"
```

参数说明：

- `--header` 需要指定 Content-Type 为 *application/json*。
- `--body` 是要发送给服务端的 body 内容，可以从本地文件读取，需要是有效的 JSON 格式。`scale.json` 的内容如下所示。

Swarm集群

```
{
    "password": "ECS 机器 SSH 密码",
    "instance_type": "实例规格",
    "size": 节点数量,
    "data_disk_category": "数据盘类型",
    "data_disk_size": 数据盘大小,
    "io_optimized":"是否 IO 优化，VPC 下目前默认为 IO 优化",
    "ecs_image_id": "镜像 ID",
    "release_eip_flag": "是否需要在集群配置完成后释放 EIP"
}
```

Kubernetes集群

```
{ "disable_rollback": "失败是否回滚",
"timeout_mins": 集群创建超时时间,
"worker_instance_type": "Worker实例规格",
"login_password": "节点SSH登录密码",
"num_of_nodes": "Worker节点数"
}
```

响应结果

```
{
  "cluster_id": "c61cf530524474386a7ab5a1c192a0d57",
  "request_id": "348D4C9C-9105-4A1B-A86E-B58F0F875575",
  "task_id": "T-5ad724ab94a2b109e8000004"
}
```

2.6. 添加已有实例

添加已有实例到集群。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群和 Kubernetes 集群。

API 请求响应

请求格式

```
aliyun cs  POST /clusters/<cluster_id>/attach --header "Content-Type=application/json" --body "$(cat attach.json)"
```

参数说明：

- `--header` 需要指定 Content-Type 为 *application/json*。
- `--body` 是要发送给服务端的 body 内容，可以从本地文件读取，需要是有效的 JSON 格式。`attach.json` 的内容如下所示。

```
{
    "password": "ECS 机器 SSH 密码",
    "instances": "ECS 示例数组",
    "ecs_image_id": "镜像 ID",
    "release_eip_flag": "是否需要在集群配置完成后释放 EIP"
}
```

响应结果

```
{
  "list": [
    {
      "code": "200",
      "instanceId": "i-2zee3oiwcyoz7kwdo8bt",
      "message": "successful"
    },
    {
      "code": "200",
      "instanceId": "i-2ze0lgm3y6iylcbtcypf",
      "message": "successful"
    }
  ],
  "task_id": "T-5a544aff80282e39ea000039"
}
```

2.7. 移除集群内节点

根据集群 ID，以及节点 IP 从容器集群中移除节点。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群。

API 请求响应

请求格式

```
aliyun cs DELETE /clusters/<cluster_id>/ip/<ip>?releaseInstance=true
```

参数说明：

- releaseInstance 设置移除节点同时是否释放 ECS。

响应结果

无。

2.8. 查看镜像列表

查看目前所支持的地域下支持的镜像列表。具体的 API 描述，参见 容器服务 API 参考。

API请求响应

请求格式

```
aliyun cs GET /images
```

响应结果

```
{
  "<RegionID>": {
    "items": [
      {
        "text": "<ImageName>",
        "value": "<ImageID>"
      },
      {
        "text": "<ImageName>",
        "value": "<ImageID>"
      }
    ]
  }
}
```

2.9. 重置节点

重置集群中的某个节点。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群。

API请求响应

请求格式

```
aliyun cs  POST /clusters/<cluster_id>/instances/<instance_id>/reset --header "Content-Type=application/json" --body "$(cat reset.json)"
```

参数说明：

- `--header` 需要指定 Content-Type 为 *application/json*。
- `--body` 是要发送给服务端的 body 内容，可以从本地文件读取，需要是有效的 JSON 格式。`reset.json` 的内容如下所示。

```
{
    "password": "ECS 机器 SSH 密码",
    "ecs_image_id": "镜像 ID",
    "release_eip_flag": "是否需要在集群配置完成后释放 EIP"
}
```

响应结果

```
{
  "cluster_id": "c61cf530524474386a7ab5a1c192a0d57",
  "request_id": "348D4C9C-9105-4A1B-A86E-B58F0F875575",
  "task_id": "T-5ad724ab94a2b109e8000004"
}
```

2.10. 删除集群实例

根据集群 ID，删除集群实例，并释放集群所有节点资源。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群和 Kubernetes 集群。

API 请求响应

请求格式

```
aliyun cs DELETE /clusters/<cluster_id>
```

响应结果

无。

2.11. 获取集群证书

根据集群 ID，获取集群的证书信息。具体的 API 描述，参见 容器服务 API 参考。

适用范围

Swarm 集群。

API 请求响应

请求格式

```
aliyun cs GET /clusters/<cluster_id>/certs
```

响应结果

```
{
  "ca": "string",
  "cert": "string",
  "key": "string"
}
```

3.开发者工具

3.1. ECS Driver 简介

Docker Machine 是一个可以帮助开发者在自己本地电脑上或云服务提供商的平台上创建 Docker 运行环境的的应用工具。阿里云 ECS Driver 是对 Docker Machine 的扩展，大大简化了在阿里云环境下 Docker 运行环境的部署，极大地方便了开发者管理分布式 Docker 主机。

通过 Docker Machine 和阿里云 ECS Driver 您可以：

- 快速创建

  通过简单的安装配置，您就可以在阿里云上自动创建 Docker 主机环境，或是 Docker 集群。

- 统一体验

  可以统一通过 Docker Machine 工具管理云端和本地 Docker 主机。

- 无缝集成

  和阿里云容器 Hub，阿里云容器服务无缝集成，提供完整的容器开发体验，是您开发和运维过程中的必备利器。

如果你希望对 ECS Driver 进行定制修改，可以从 GitHub 直接获取源代码并编译包含 ECS Driver 的 Docker Machine。

地址：https://github.com/denverdino/docker-machine-driver-aliyunecs

3.2. 安装方法

安装 Docker Machine

最简单的办法是安装 Docker 开发工具箱，这是 Docker 的工具包，包含了 Docker Engine，Docker Machine 等一系列工具。

如果您需要单独安装 Docker Machine，可以参考官方的安装文档 Install Docker Machine。

获取 ECS Driver

您可以直接下载阿里云 ECS Driver 编译结果。以下版本可以配合 Docker Machine 0.5 及更高版本使用。

- Windows 64 bit：docker-machine-driver-aliyunecs_windows-amd64.tgz
- Linux 64 bit：docker-machine-driver-aliyunecs_linux-amd64.tgz
- Mac OSX 64 bit：docker-machine-driver-aliyunecs_darwin-amd64.tgz

安装 ECS Driver

Docker Machine 要求 ECS Driver 的 binary 在系统路径（以 Mac OSX 为例，就是 `$PATH` 包含的路径）下，将文件名改为 *docker-machine-driver-aliyunecs* 并添加执行权限即可，这样 Docker Machine 可以直接访问到。

Mac OSX 下安装命令其实就是将 ECS Driver 的 binary 拷贝到了 */usr/local/bin* 目录下。Mac OSX 下的命令如下所示。

```
mkdir docker-machine
# Download and unzip Aliyun ECS driver
curl -L https://docker-machine-drivers.oss-cn-beijing.aliyuncs.com/docker-machine-driver-aliyunecs_darwin-amd64.tgz > driver-aliyunecs.tgz && tar xzvf driver-aliyunecs.tgz -C docker-machine && rm driver-aliyunecs.tgz
mv docker-machine/bin/* /usr/local/bin
mv /usr/local/bin/docker-machine-driver-aliyunecs.darwin-amd64 /usr/local/bin/docker-machine-driver-aliyunecs && chmod +x /usr/local/bin/docker-machine-driver-aliyunecs
```

3.3. 使用方法

配置环境变量

通过配置环境变量，可以简化 Docker Machine 命令里每次都要用到的参数。环境变量的参考配置如下所示。

```
export DEBUG=true
export ECS_ACCESS_KEY_ID=<your_access_key_id>
export ECS_ACCESS_KEY_SECRET=<your_access_key_secret>
export ECS_REGION=<your_ecs_region>
export ECS_SSH_PASSWORD=<your_ssh_password>
export MACHINE_DOCKER_INSTALL_URL= http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/internet
# Optional for VPC only
export ECS_VPC_ID=<your_vpc_id>
export ECS_VSWITCH_ID=<your_vswitchid>
```

说明

- `ECS_REGION` 是您的 ECS 实例的地域，例如，cn-beijing，cn-hangzhou，cn-qingdao。
- `ECS_SSH_PASSWORD` 是您之前设置的 ECS 机器 SSH 登录的密码。密码可以包含 8~30 个字符，必须同时包含以下三种字符（大/小写字母、数字和特殊符号），且不支持反斜杠（\）和双引号（”）。详细信息参见 ECS 实例的 相关文档。
- 由于 Docker Engine 的官方 repo 在国内访问非常不稳定，您需要将 `MACHINE_DOCKER_INSTALL_URL` 环境变量配置到如上所示的阿里云镜像站点；否则，在创建机器时可能会出现 `Error Creating machine: Error running provisioning: error installing docker` 之类的错误。在阿里云内网，可以使用 http://acs-public-mirror.oss-cn-hangzhou.aliyuncs.com/docker-engine/intranet 作为 Docker Engine 的安装镜像。
- 如果没有 VPC 相关配置，请不要配置 `ECS_VPC_ID` 和 `ECS_VSWITCH_ID` 这两个环境变量。如果您有 VPC 相关账号，请确保您当前的 `ECS_REGION` 支持 VPC 配置。

使用 ECS Driver 创建或删除 ECS 机器

配置过环境变量后，大部分参数使用环境变量默认就可以了。

您可以使用以下命令创建一台带有 Docker 环境的 ECS 实例。

```
docker-machine create -d aliyunecs dev1
```

示例输出结果如下所示。

```
$ docker-machine create -d aliyunecs dev1
Running pre-create checks...
Creating machine...
Waiting for machine to be running, this may take a few minutes...
Machine is running, waiting for SSH to be available...
Detecting operating system of created instance...
Provisioning created instance...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
To see how to connect Docker to this machine, run: docker-machine env dev1
```

您可以使用 **docker-machine ls** 命令查看创建的包含 Docker 环境的机器的情况。

```
$ docker-machine ls
NAME   ACTIVE   DRIVER    STATE    URL          SWARM
dev1   -    aliyunecs  Running    tcp://1.2.7.2:9376
```

您可以使用 **docker-machine kill** 命令停止此机器，并查看机器状态。

```
$ docker-machine kill dev1
$ docker-machine ls
NAME  ACTIVE  DRIVER   STATE   URL             SWARM
dev1  -    aliyunecs  Stopped  tcp://1.2.7.2:9376
```

您可以使用 **docker-machine rm** 命令删除此机器（也可以不停止机器直接删除机器），并查看机器信息。

```
$ docker-machine rm dev1
Successfully removed dev1
$ docker-machine ls
NAME  ACTIVE  DRIVER  STATE  URL  SWARM
```

更多 **docker-machine** 命令可以参考官方 machine subcommands reference。

3.4. 支持的命令参数

可以使用如下命令来显示所有可能的配置参数。

```
docker-machine create -d aliyunecs
```

ECS Driver 支持的部分命令参数如下所示（参考 说明文档）。

- --aliyunecs-access-key-id：必配参数。阿里云 ECS API 的 AccessKey ID。（阿里云 AccessKey ID 默认使用环境变量里的ECS_ACCESS_KEY_ID。）
- --aliyunecs-access-key-secret：必配参数。阿里云 ECS API 的 AccessKey Secret。（阿里云 AccessKey Secret 默认使用环境变量里的ECS_ACCESS_KEY_SECRET。）
- --aliyunecs-disk-size： /var/lib/docker 目录下的数据盘大小（单位：GB）。
- --aliyunecs-tag: ECS 实例的标签（tag）。

- --aliyunecs-vpc-id: 只有网络模式为 VPC 的时候，才需要配置该参数。用来创建 ECS 实例的 VPC ID。
- --aliyunecs-vswitch-id: 只有网络模式为 VPC 的时候，才需要配置该参数。用来创建 ECS 实例的 VSwitch ID。
- --aliyunecs-zone: 用来创建 ECS 实例的可用区。

以上配置参数需要对阿里云的 ECS 产品 API 有一定的了解，可以查看 ECS 产品 API 文档。

您还可以扩展阅读一下 Alibaba Cloud CLI，使用此 CLI 可以方便地用命令行与 ECS 直接交互。

4.Swarm API参考

4.1. 简介

您可以使用本文档介绍的 API 对容器服务进行相关操作。

> **说明** 请确保在使用这些接口前，您已充分了解了容器服务说明、使用协议和收费方式。

术语表

术语	中文	说明
Cluster	集群	您的容器集群，一个集群可以部署多个应用。
Node	节点	您的容器集群中的某一个节点，目前只支持 ECS 实例。
Project	应用	一个复杂的应用可以由多个服务组合而成，最简单的应用可能只包含一个容器。
Service	服务	一组基于相同镜像和配置定义的容器，作为一个可伸缩的微服务。
Container	容器	Docker 容器运行时实例。

4.2. API 概述

容器服务的 API 主要分为3个部分：

- 集群管理接口
- 应用管理接口
- 触发器

集群管理接口

容器服务提供了一些用于集群管理的接口，例如创建集群、删除集群等。

集群管理接口列表：

API	描述
GetClusterList	查看所有集群实例
CreateCluster	创建集群实例
DeleteCluster	删除集群实例
GetClusterById	查看集群实例

API	描述
GetClusterCerts	获取集群证书
UpdateClusterSizeById	更新集群节点数量

应用管理接口

应用管理提供了 Docker Remote API 兼容接口。您可以像访问单个 Docker Engine 一样，操作您的 Docker 集群。

应用管理接口列表：

API	描述
List Projects	查看应用实例列表
Create project	创建应用实例
Retrieve project	查看应用实例
Start project	启动应用实例
Stop project	停止应用实例
Kill project	终止应用实例
Update project	更新应用实例
Delete project	删除应用实例
List Services	查看服务实例列表
Retrieve service	查看服务实例
Start service	启动服务实例
Stop service	停止服务实例
Kill service	终止服务实例
Scale service	伸缩服务实例
Create volume	创建数据卷
View volume	查看数据卷
List volumes	查看数据卷列表
Delete volume	删除数据卷

触发器

触发器是容器服务提供的简单快捷地进行持续部署的 API。更多详细信息参见 说明。

相关文档

- API 资源导航
- API Explorer
- API 错误中心

4.3. 更新历史

发布时间	更新	说明
2015-12-15	第一版确定	提供了集群管理的基本接口
2016-02-04	增加应用相关接口	提供了应用管理的基本接口

4.4. 状态表

状态	说明
创建中（Launching）	集群正在申请相应的云资源
运行中（Running）	集群正在运行
创建失败（Failed）	集群申请云资源失败
启动中（Starting）	集群正在启动
停止中（Stopping）	集群正在停止运行
停止（Stopped）	集群停止运行
重启中（Restarting）	集群正在重启
升级中（Updating）	集群升级服务端版本
集群规模变更中（Scaling）	变更集群的节点数量
删除中（Deleting）	正在删除集群
已删除（Deleted）	集群删除成功

4.5. 集群API调用方式

4.5.1. 概述

对容器服务 API 接口的调用是通过向容器服务 API 的服务端地址发送 HTTP 请求，并按照接口说明在请求中加入相应请求参数来完成的。根据请求的处理情况，系统会返回处理结果。

1. 请求结构
2. 公共参数

3. 返回结果
4. 签名机制

4.5.2. 请求结构

服务地址

阿里云容器服务的 Open API 接入地址为 cs.aliyuncs.com。

通信协议

支持通过 HTTP 或 HTTPS 通道进行请求通信。为了获得更高的安全性，推荐您使用 HTTPS 通道发送请求。

请求方法

使用 HTTP 的 PUT、 POST、 GET、 DELETE 等 HTTP Method 发送不同的请求。

请求参数

每个请求都需要包含公共请求参数和指定操作所特有的请求参数。

请求编码

请求及返回结果都使用 UTF-8 字符集进行编码。

4.5.3. 公共参数

公共请求头部

公共请求参数是指每个接口都需要使用到的请求参数。

参数名称	说明	选项
Authorization	用于验证请求合法性的认证信息，采用 `AccessKeyId:Signature` 的形式。	Required
Content-Length	RFC 2616 中定义的 HTTP 请求内容长度。	Required
Content-Type	RFC 2616 中定义的 HTTP 请求内容类型。	Required
Content-MD5	HTTP 协议消息体的 128-bit MD5 散列值转换成 BASE64 编码的结果。为了防止所有请求被篡改，建议所有请求都附加该信息。	Required
Date	请求的构造时间，目前只支持 GMT 格式。如果与 MNS 的服务器时间前后差异超过 15 分钟将返回本次请求非法。	Required

参数名称	说明	选项
Host	访问 Host 值，例如：*diku.aliyuncs.com*。	Required
Accept	客户端需要的返回值类型，支持 *application/json* 和 *application/xml*。	Required
x-acs-version	API 版本号。目前版本号为 *2015-12-15*。	Required
x-acs-region-id	地域（Region）指的是 ECS 实例所在的物理位置。更多详细信息参见 地域概念 和 ECS查询可用地域列表。	Required
x-acs-signature-nonce	唯一随机数，用于防止网络重放攻击。您在不同请求间要使用不同的随机数值。	Required
x-acs-signature-method	用户签名方式，目前只支持 *HMAC-SHA1*。	Required

示例

```
GET /clusters HTTP/1.1
Host: cs.aliyuncs.com
Accept: application/json
User-Agent: cs-sdk-python/0.0.1 (Darwin/15.2.0/x86_64;2.7.10)
x-acs-signature-nonce: f63659d4-10ac-483b-99da-ea8fde61eae3
Authorization: acs <yourAccessKeyId>:<yourSignature>
x-acs-signature-version: 1.0
Date: Wed, 16 Dec 2015 11:18:47 GMT
x-acs-signature-method: HMAC-SHA1
Content-Type: application/json;charset=utf-8
X-Acs-Region-Id: cn-beijing
Content-Length: 0
```

公共返回头部

您发送的每次接口调用请求，无论成功与否，系统都会返回一个唯一识别码 RequestId。

示例

XML 示例：

```
<?xml version="1.0" encoding="UTF-8"?>
<!—结果的根结点-->
<接口名称+Response>
|<!—返回请求标签-->
|<RequestId>4C467B38-3910-447D-87BC-AC049166F216</RequestId>
|<!—返回结果数据-->
</接口名称+Response>
```

JSON 示例：

```
{
  "RequestId": "4C467B38-3910-447D-87BC-AC049166F216"
  /* 返回结果数据 */
}
```

4.5.4. 返回结果

调用 API 服务后返回数据采用统一格式。返回的 HTTP 状态码为 2xx，代表调用成功；返回的 HTTP 状态码为 4xx 或 5xx，代表调用失败。调用成功返回的数据格式主要有 XML 和 JSON 两种，外部系统可以在请求时传入参数来制定返回的数据格式，默认为 XML 格式。

为了便于您查看，本文档中的返回示例做了格式化处理，实际返回结果是没有换行、缩进等处理的。

4.5.5. 签名机制

签名机制说明

Access Key ID 和 Access Key Secret 由阿里云官方颁发给访问者（可以通过阿里云官方网站申请和管理），其中 Access Key ID 用于标识访问者的身份；Access Key Secret 是用于加密签名字符串和服务器端验证签名字符串的密钥，必须严格保密，只有阿里云和用户知道。

容器服务会对每个访问的请求进行验证，每个向容器服务提交的请求，都需要在请求中包含签名（Signature）信息。容器服务通过使用 Access Key ID 和 Access Key Secret 进行对称加密的方法来验证请求的发送者身份。如果计算出来的验证码和提供的一样即认为该请求是有效的；否则，容器服务将拒绝处理这次请求，并返回 HTTP 403 错误。

用户可以在 HTTP 请求中增加授权（Authorization）的 Head 来包含签名信息，表明这个消息已被授权。

容器服务要求将签名包含在 HTTP Header 中，格式为 Authorization: acs [Access Key Id]:[Signature] 。

Signature 的计算方法如下：

```
Signature = base64(hmac-sha1(VERB + "\n"
        + ACCEPT + "\n" +
        + Content-MD5 + "\n"
        + Content-Type + "\n"
        + Date + "\n"
        + CanonicalizedHeaders + "\n"
        + CanonicalizedResource))
```

- VERB 表示 HTTP 的 Method。比如示例中的 PUT 。
- Accept 客户端需要的返回值类型，支持 application/json 和 application/xml。
- Content-MD5 表示请求内容数据的 MD5 值。
- Content-Type 表示请求内容的类型。
- Date 表示此次操作的时间，不能为空，目前只支持 GMT 格式。如果请求时间与 CAS 服务器时间相差超过 1 5分钟，CAS 会判定此请求不合法，并返回 400 错误。错误信息及错误码详见本文档第 5 部分。比如示例中的 Thu, 17 Mar 2012 18:49:58 GMT 。
- CanonicalizedHeaders 表示 HTTP 中以 x-acs- 开始的字段组合。
- CanonicalizedResource 表示 HTTP 所请求资源的 URI (统一资源标识符)。比如示例中的 /clusters?name=my-clusters&resource=new 。

> **说明** CanonicalizedHeaders（即以 x-acs- 开头的 header）在签名验证前需要符合以下规范：

1. 将所有以 x-acs- 为前缀的 HTTP 请求头的名字转换成小写字母。比如将 X-ACS-Meta-Name: TaoBao 转换为 x-acs-meta-name: TaoBao 。阿里云规范请求头的名字是大小写不敏感的，建议全部使用小写。
2. 如果一个公共请求头的值部分过长，则需要处理其中的 \t 、 \n 、 \r 、 \f 分隔符，将其替换为英文半角的空格。
3. 将上一步得到的所有 HTTP 阿里云规范头按照字典序进行升序排列。
4. 删除请求头和内容之间分隔符两端出现的任何空格。比如将 x-acs-meta-name: TaoBao,Alipay 转换为 x-acs-meta-name:TaoBao,Alipay 。
5. 将所有的头和内容用 \n 分隔符分隔拼成最后的 CanonicalizedHeaders 。

> **说明** CanonicalizedResource 的格式规范：CanonicalizedResource 表示客户想要访问资源的规范描述，需要将子资源和 qurey 一同按照字典序，从小到大排列并以 & 为分隔符生成子资源字符串（ ? 后的所有参数）。

```
http://cs.aliyuncs.com/clusters?name=my-clusters&resource=new
```

CanonicalizedResource 应该为：

```
/clusters?name=my-clusters&resource=new
```

签名示例

示例概述

您可以通过该示例，了解加签的步骤。

示例使用的 accessKeyId 和 accessKeySecret 分别为 access_key_id 和 access_key_secret 。推荐您使用自己的 OpenAPI 调用程序，来计算下面这个示例的加签串，您自己的加签结果和示例结果。

请求的示例如下：

```
POST http://cs.aliyuncs.com/clusters?param1=value1&param2=value2 HTTP/1.1
Accept-Encoding: identity
Content-Length: 210
Content-MD5: 6U4ALMkKSj0PYbeQSHqgmA==
x-acs-version: 2015-12-15
Accept: application/json
User-Agent: cs-sdk-python/0.0.1 (Darwin/15.2.0/x86_64;2.7.10)
x-acs-signature-nonce: fbf6909a-93a5-45d3-8b1c-3e03a7916799
x-acs-signature-version: 1.0
Date: Wed, 16 Dec 2015 12:20:18 GMT
x-acs-signature-method: HMAC-SHA1
Content-Type: application/json;charset=utf-8
X-Acs-Region-Id: cn-beijing
Authorization: acs access_key_id:/ZmVlMDNkNDA1ZTQyMWViYWY1MTRhZGVjODgxMDM4YzRiMzEzNTg0ZA==
{"password": "Just$test","instance_type": "ecs.m2.medium","name": "my-test-cluster-97082734","size": 1, "network_mode": "classic","data_disk_category": "cloud","data_disk_size": 10,"ecs_image_id": "m-253llee3l"}
```

请求构造过程

计算 `Content-Length` 和 `Content-MD5`

`Content-Length`：body 内容的长度。

> **说明** 示例 body 首位没有空格或换行符

```
body: {"password": "Just$test","instance_type": "ecs.m2.medium","name": "my-test-cluster-97082734","size": 1,"network_mode": "classic","data_disk_category": "cloud","data_disk_size": 10,"ecs_image_id": "m-253llee3l"}
Content-Length: 210
```

`Content-MD5`：MD5 的计算过程。

```
body: {"password": "Just$test","instance_type": "ecs.m2.medium","name": "my-test-cluster-97082734","size": 1,"network_mode": "classic","data_disk_category": "cloud","data_disk_size": 10,"ecs_image_id": "m-253llee3l"}
# 计算 body 的 md5 值
md5(body): e94e002cc90a4a3d0f61b790487aa098
# 将 md5 值转化成字节数组。将 md5 中的每两个十六进制位合并，转化为一个字节。
# 例如：e9 -> 11111111111111111111111111101001 -> -23
bytes(md5(body)): {[-23], [78], [0], [44], [-55], [10], [74], [61], [15], [97], [-73], [-112], [72], [122], [-96], [-104]}
# 将得到的字节数组做一个 base64 转换
base64(bytes(md5(body))): 6U4ALMkKSj0PYbeQSHqgmA==
Content-MD5: 6U4ALMkKSj0PYbeQSHqgmA==
```

处理 `CanonicalizedHeaders`

```
# 将所有以‘x-acs-’开头的头部列出来
x-acs-version: 2015-12-15
x-acs-signature-nonce: ca480402-7689-43ba-acc4-4d2013d9d8d4
x-acs-signature-version: 1.0
x-acs-signature-method: HMAC-SHA1
X-Acs-Region-Id: cn-beijing
# 将请求名字变成小写，去掉每一行首尾的空格，并按照字典序进行排序。删除请求头和内容之间分隔符两端出现的任何空格。
# 注意：最后一行没有换行符。
x-acs-region-id:cn-beijing
x-acs-signature-method:HMAC-SHA1
x-acs-signature-nonce:fbf6909a-93a5-45d3-8b1c-3e03a7916799
x-acs-signature-version:1.0
x-acs-version:2015-12-15
```

计算 `CanonicalizedResource`

示例得到的 CanonicalizedResource，长度应该为 27。

> **说明** 第一行行尾有一个 `\n` 的换行符。

```
/clusters?param1=value1&param2=value2
```

计算 `Signature`

组装 `SignatureString` 。示例中的加签字符串的长度为 307。除最后一行外，每一行行尾均有一个 `\n` 的换行符。

```
POST
application/json
6U4ALMkKSj0PYbeQSHqgmA==
application/json;charset=utf-8
Wed, 16 Dec 2015 12:20:18 GMT
x-acs-region-id:cn-beijing
x-acs-signature-method:HMAC-SHA1
x-acs-signature-nonce:fbf6909a-93a5-45d3-8b1c-3e03a7916799
x-acs-signature-version:1.0
x-acs-version:2015-12-15
/clusters?param1=value1&param2=value2
```

计算 `Signature`

```
# 使用 accessKeySecret 来对加签字符串进行加密，其中示例使用的 accessKeySecret 是 access_key_secret。
hmac-sha1(SignatureString): fee03d405e421ebaf514adec881038c4b313584d
# 类似于 Content-MD5 的计算方式，将得到的加密串转化成字节数组。
# 将得到的字符数组做一个 base64 转换。得到最后的签名串。
base64(bytes(hmac-sha1(SignatureString))): ZmVlMDNkNDA1ZTQyMWViYWY1MTRhZGVjODgxMDM4YzRiMzEzNTg0ZA==
Signature: ZmVlMDNkNDA1ZTQyMWViYWY1MTRhZGVjODgxMDM4YzRiMzEzNTg0ZA==
```

完成

经过以上的处理，添加一些其他头部信息，最终构成的 HTTP 请求如下所示。

```
POST http://cs.aliyuncs.com/clusters?param1=value1&param2=value2 HTTP/1.1
Accept-Encoding: identity
Content-Length: 210
Content-MD5: 6U4ALMkKSj0PYbeQSHqgmA==
x-acs-version: 2015-12-15
Accept: application/json
User-Agent: cs-sdk-python/0.0.1 (Darwin/15.2.0/x86_64;2.7.10)
x-acs-signature-nonce: fbf6909a-93a5-45d3-8b1c-3e03a7916799
x-acs-signature-version: 1.0
Date: Wed, 16 Dec 2015 12:20:18 GMT
x-acs-signature-method: HMAC-SHA1
Content-Type: application/json;charset=utf-8
X-Acs-Region-Id: cn-beijing
Authorization: acs access_key_id:/ZmVlMDNkNDA1ZTQyMWViYWY1MTRhZGVjODgxMDM4YzRiMzEzNTg0ZA==
{"password": "Just$test","instance_type": "ecs.m2.medium","name": "my-test-cluster-97082734","size": 1,
"network_mode": "classic","data_disk_category": "cloud","data_disk_size": 10,"ecs_image_id": "m-253llee3l"}
```

4.6. 集群API列表

4.6.1. 获取集群证书

根据集群 ID，获取集群的证书信息。使用 Docker Client 操作集群时，需要使用该证书进行访问。更多详细信息，参见 通过 Docker 工具连接集群。

请求信息

请求行 RequestLine

```
GET /clusters/{cluster_id}/certs HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
cluster_id	string	是	集群 ID

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
{
  "ca": "string",
  "cert": "string",
  "key": "string"
}
```

返回体解析

名称	类型	描述
ca	string	认证机构证书，*ca.pem*
cert	string	用户公钥证书，*cert.pem*
key	string	用户私钥证书，*key.pem*

示例

请求示例

```
GET /clusters/Cccfd68c474454665ace07efce924f75f/certs HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 200 OK
<公共响应头>
{
  "ca": "-----BEGINCERTIFICATE-----ca content-----ENDCERTIFICATE-----\n",
  "cert": "-----BEGINCERTIFICATE-----cert content-----ENDCERTIFICATE-----\n",
  "key": "-----BEGINRSAPRIVATEKEY-----key content-----ENDRSAPRIVATEKEY-----\n"
}
```

4.6.2. 查看所有集群实例

查看您在容器服务中创建的所有集群(包括 Swarm 和 Kubernetes 集群)。

请求信息

请求行 RequestLine

```
GET /clusters HTTP/1.1
```

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
[
  {
    "agent_version": "string",
    "cluster_id": "string",
    "created": "datetime",
    "external_loadbalancer_id": "string",
    "master_url": "string",
    "name": "string",
    "network_mode": "string",
    "region_id": "string",
    "security_group_id": "string",
    "size": "numbers",
    "state": "string",
    "updated": "datetime",
    "vpc_id": "string",
    "vswitch_id": "string"
  }
]
```

返回体解释

Cluster 的格式

名称	类型	描述
agent_version	string	Agent 版本号。
cluster_id	String	集群 ID，集群的唯一标识。
created	string	集群的创建时间。
external_loadbalancer_id	String	集群负载均衡服务的 ID。
master_url	string	集群 Master 地址，您可以通过该地址连接您的集群进行相关操作。更多详细信息，参见 通过 Docker 工具连接集群。
name	string	集群名称，由您在创建集群时指定，在每个用户下唯一。
network_mode	String	集群网络模式（VPC 网络: vpc）。
region_id	String	集群所在地域 ID。
security_group_id	String	安全组 ID。
size	String	节点数。
state	String	集群状态。更多详细信息，参见 集群的生命周期。
updated	string	最后更新时间。

名称	类型	描述
vpc_id	string	VPC ID。
vswitch_id	string	VSwitch ID。

示例

请求示例

```
GET /clusters HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 200 OK
<公共响应头>
[
  {
    "agent_version": "0.5-e56dab3",
    "cluster_id": "c978ca3eaacd3409a9437db07598f1f69",
    "created": "2015-12-11T03:52:40Z",
    "external_loadbalancer_id": "1518f2b7e4c-cn-beijing-btc-a01",
    "master_url": "https://182.92.245.56:17589",
    "name": "my-python-cluster-039de960",
    "network_mode": "vpc",
    "region_id": "cn-beijing",
    "security_group_id": "sg-25yqjuxhz",
    "size": 5,
    "state": "running",
    "updated": "2015-12-15T15:01:58Z",
    "vpc_id": "",
    "vswitch_id": ""
  },
  {
    "agent_version": "0.5-e56dab3",
    "cluster_id": "c1eb19e0093204cbb86c3a80334d2129e",
    "created": "2015-12-15T14:26:58Z",
    "external_loadbalancer_id": "151a6099de1-cn-beijing-btc-a01",
    "master_url": "https://182.92.245.56:11905",
    "name": "my-test-cluster-002b3f3d",
    "network_mode": "vpc",
    "region_id": "cn-beijing",
    "security_group_id": "sg-25rg2ws9f",
    "size": 1,
    "state": "running",
    "updated": "2015-12-15T14:43:55Z",
    "vpc_id": "",
    "vswitch_id": ""
  }
]
```

4.6.3. 创建集群实例

创建一个新的集群实例，并新建指定数量的节点。

请求信息

请求行 RequestLine

```
POST /clusters HTTP/1.1
```

请求行参数 URI Param

无

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

```
{
  "password": "ECS实例root登录密码",
  "region_id": "RegionID",
  "instance_type": "实例规格",
  "name": "集群名称",
  "size": "节点数",
  "network_mode": "vpc",
  "vpc_id": "VPC_ID",
  "vswitch_id": "交换机ID",
  "subnet_cidr": "容器网段，如172.28.1.0/24",
  "data_disk_category": "系统盘类型",
  "data_disk_size": "系统盘大小",
  "need_slb": "是否需要创建集群的负载均衡",
  "ecs_image_id": "操作系统镜像",
  "io_optimized": "是否IO优化",
  "release_eip_flag": "是否需要在集群配置完成后释放EIP"
  "rds_instances": "RDS 实例 ID"
}
```

请求体解释

名称	类型	必须	描述
name	string	是	集群名称, 集群名称可以使用大小写英文字母、中文、数字、中划线。
size	int	是	集群 ECS 节点数量。
instance_type	string	是	ECS 规格类型代码。更多详细信息，参见 实例规格族。

名称	类型	必须	描述
network_mode	string	是	集群网络模式（VPC 网络： vpc），目前仅支持 VPC 网络。
subnet_cidr	string	是	集群可以使用的网络地址块，例如：*192.168.24.0/22*。只有网络模式为 vpc 的时候，才需要设置该字段。
vpc_id	string	是	VPC 网络 ID。只有网络模式为 vpc 的时候，才需要设置该字段。
vswitch_id	string	是	VPC 网络的交换机 ID。只有网络模式为 vpc 的时候，才需要设置该字段。
password	string	是	root 账号密码。
data_disk_category	string	是	ECS 使用的磁盘类型。更多详细信息，参见 磁盘种类表。
data_disk_size	number	是	节点共享磁盘大小。
ecs_image_id	string	可选	ECS 使用的系统镜像 ID。参见 查询镜像列表 API。
io_optimized	string	可选	根据 ECS 实例规则来确定。取值为 *none* 或者 *optimized*。由于目前仅支持 VPC 网络，建议传入 *optimized*。
need_slb	bool	可选	是否需要创建集群默认的简单路由 SLB，默认为 *true*。
release_eip_flag	bool	可选	配置完集群后是否释放 EIP，默认为 *false*。
rds_instances	array	否	选择是否将 ECS 实例的 IP 添加到 RDS 的白名单里。

ecs_image_id 列表

请参考文档 查看镜像列表 获取 ecs_image_id 列表。如果您需要自定义集群的 ECS 镜像的 ID，需要保证 ECS 镜像满足以下条件：

- 操作系统：Ubuntu、CentOS。

- Linux Kernel version ≥ 3.18，用于支持 overlayfs 以及 overlay network。
- 镜像中删除 */etc/docker/key.json* 文件。

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 Accepted
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
{
  "cluster_id":"string",
  "request_id":"string",
  "task_id":"string"
}
```

示例

请求示例

```
POST /clusters HTTP/1.1
<公共请求头>
{
  "password": "TestPwd124",
  "region_id": "cn-beijing",
  "instance_type": "ecs.n1.small",
  "name": "my-test-cluster",
  "size": 1,
  "network_mode": "vpc",
  "vpc_id":"vpc-xxxx",
  "vswitch_id":"vsw-xxxx",
  "subnet_cidr":"172.28.1.0/24",
  "data_disk_category": "cloud_ssd",
  "data_disk_size": 40,
  "need_slb":true,
  "ecs_image_id":"centos_7_04_64_20G_alibase_201701015",
  "io_optimized":"true",
  "release_eip_flag":false
}
```

返回示例

```
HTTP/1.1 202 Accepted
<公共响应头>
{
  "cluster_id": "cb95aa626a47740afbf6aa099b650d7ce",
  "request_id": "687C5BAA-D103-4993-884B-C35E4314A1E1",
  "task_id": "T-5a54309c80282e39ea00002f"
}
```

4.6.4. 删除集群实例

根据集群 ID，删除集群实例，并释放集群所有节点资源。

请求信息

请求行 RequestLine

```
DELETE /clusters/{cluster_id} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
cluster_id	string	是	集群 ID

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 Accepted
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

无

示例

请求示例

```
DELETE /clusters/Cccfd68c474454665ace07efce924f75f HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 202 Accepted
<公共响应头>
```

4.6.5. 查看集群实例

根据集群 ID，查看集群的详细信息。

请求信息

请求行 RequestLine

```
GET /clusters/{cluster_id} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
cluster_id	string	是	集群 ID

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
{
  "agent_version": "string",
  "cluster_id": "string",
  "created": "datetime",
  "external_loadbalancer_id": "string",
  "master_url": "string",
  "name": "string",
  "network_mode": "string",
  "region_id": "string",
  "security_group_id": "string",
  "size": "numbers",
  "state": "string",
  "updated": "datetime",
  "vpc_id": "string",
  "vswitch_id": "string"
}
```

返回体解释

Cluster 的格式

名称	类型	描述
agent_version	string	Agent 版本号。
cluster_id	String	集群 ID，集群的唯一标识。
created	string	集群的创建时间。
external_loadbalancer_id	String	集群负载均衡服务的 ID。
master_url	string	集群 Master 地址，您可以通过该地址连接您的集群进行相关操作。更多详细信息，参见 通过 Docker 工具连接集群。
name	string	集群名称，由您在创建集群时指定，在每个用户下唯一。
network_mode	String	集群网络模式（经典网络：classic、VPC 网络：vpc）。
region_id	String	集群所在地域 ID。
security_group_id	String	安全组 ID。
size	String	节点数。
state	String	集群状态。更多详细信息，参见 集群的生命周期。
updated	string	最后更新时间。
vpc_id	string	VPC ID。

名称	类型	描述
vswitch_id	string	VSwitch ID。

示例

请求示例

```
GET /clusters/C5b5e80b0b64a4bf6939d2d8fbbc5ded7 HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 200 Ok
<公共响应头>
{
  "agent_version": "0.5-e56dab3",
  "cluster_id": "c978ca3eaacd3409a9437db07598f1f69",
  "created": "2015-12-11T03:52:40Z",
  "external_loadbalancer_id": "1518f2b7e4c-cn-beijing-btc-a01",
  "master_url": "https://182.92.245.56:17589",
  "name": "my-python-cluster-039de960",
  "network_mode": "classic",
  "region_id": "cn-beijing",
  "security_group_id": "sg-25yqjuxhz",
  "size": 5,
  "state": "running",
  "updated": "2015-12-15T15:01:58Z",
  "vpc_id": "",
  "vswitch_id": ""
}
```

4.6.6. 扩容集群

增加集群中节点的数量。

请求信息

请求行 RequestLine

```
PUT /clusters/{cluster_id} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
cluster_id	string	是	集群 ID

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

```
{
  "password": "ECS实例root登录密码",
  "instance_type": "实例规格",
  "size": "扩容到节点数",
  "data_disk_category": "系统盘类型",
  "data_disk_size": "系统盘大小",
  "ecs_image_id": "操作系统镜像",
  "io_optimized": "是否IO优化",
  "release_eip_flag": "是否需要在集群配置完成后释放EIP"
}
```

请求体解析

名称	类型	是否必须	描述
password	String	是	ECS 实例密码。
instance_type	String	是	ECS 规格类型代码。更多详细信息，参见 实例规格族。
size	int	是	节点的总数量，要大于现有节点数量。
data_disk_category	String	否	ECS 使用的磁盘类型。更多详细信息，参见 磁盘种类表。
data_disk_size	Number	否	节点共享磁盘大小（单位：GB）。
ecs_image_id	String	是	ECS 使用的系统镜像 ID。
io_optimized	String	否	根据 ECS 实例规则来确定。取值为 *none* 或者 *optimized*。
release_eip_flag	bool	可选	配置完集群后是否释放 EIP，默认为 *false*。

ecs_image_id 列表

请参考文档 查看镜像列表 获取 ecs_image_id 列表。如果您需要自定义集群的 ECS 镜像的 ID，需要保证 ECS 镜像满足以下条件：

- 操作系统：Ubuntu、Centos。
- Linux Kernel version ≥ 3.18，用于支持 overlayfs 以及 overlay network。
- 镜像中删除 */etc/docker/key.json* 文件。

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
{
  "cluster_id":"string",
  "request_id":"string",
  "task_id":"string"
}
```

示例

请求示例

```
PUT /clusters/Cccfd68c474454665ace07efce924f75f HTTP/1.1
<公共请求头>
{
 "password": "password",
 "instance_type": "ecs.s3.large",
 "size": 2,
 "data_disk_category": "cloud_ssd",
 "data_disk_size": 500,
 "ecs_image_id": "centos_7_2_64_40G_base_20170222.vhd",
 "io_optimized":"optimized",
 "release_eip_flag": false,
}
```

返回示例

```
HTTP/1.1 202 Accepted
<公共响应头>
{
  "cluster_id": "cb95aa626a47740afbf6aa099b650d7ce",
  "request_id": "687C5BAA-D103-4993-884B-C35E4314A1E1",
  "task_id": "T-5a54309c80282e39ea00002f"
}
```

4.6.7. 添加已有实例

添加已有实例到集群。

> **说明** 添加过程中会替换系统盘，需要提前做好数据备份。

请求信息

请求行 RequestLine

```
POST /clusters/{cluster_id}/attach HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
cluster_id	string	是	集群 ID

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

```
{
  "password": "ECS实例root登录密码",
  "instances": "要添加的实例数组",
  "ecs_image_id": "操作系统镜像",
  "release_eip_flag": "是否需要在集群配置完成后释放EIP"
}
```

请求体解析

名称	类型	是否必须	描述
password	String	是	ECS 实例密码。
instances	Array	是	已有实例的数组
ecs_image_id	String	是	ECS 使用的系统镜像 ID。
release_eip_flag	bool	可选	配置完集群后是否释放 EIP，默认为 *false*。

ecs_image_id 列表。

请参考文档 查看镜像列表 获取 ecs_image_id 列表。如果您需要自定义集群的 ECS 镜像的 ID，需要保证 ECS 镜像满足以下条件：

- 操作系统：Ubuntu、Centos。
- Linux Kernel version ≥ 3.18，用于支持 overlayfs 以及 overlay network。
- 镜像中删除 */etc/docker/key.json* 文件。

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 OK
{
  "list": [
    {
      "code": "200",
      "instanceId": "i-xxx",
      "message": "successful"
    },
    {
      "code": "200",
      "instanceId": "i-yyy",
      "message": "successful"
    }
  ],
  "task_id": "T-5a544aff80282e39ea000039"
}
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
{
  "list": [
    {
      "code": "200",
      "instanceId": "i-2zee3oiwcyoz7kwdo8bt",
      "message": "successful"
    },
    {
      "code": "200",
      "instanceId": "i-2ze0lgm3y6iylcbtcypf",
      "message": "successful"
    }
  ],
  "task_id": "T-5a544aff80282e39ea000039"
}
```

示例

请求示例

```
POST /clusters/Cccfd68c474454665ace07efce924f75f/attach HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 202 Accepted
<公共响应头>
{
  "list": [
    {
      "code": "200",
      "instanceId": "i-2zee3oiwcyoz7kwdo8bt",
      "message": "successful"
    },
    {
      "code": "200",
      "instanceId": "i-2ze0lgm3y6iylcbtcypf",
      "message": "successful"
    }
  ],
  "task_id": "T-5a544aff80282e39ea000039"
}
```

4.6.8. 移除集群内节点

根据集群 ID，以及节点 IP 从容器集群中移除节点。

请求信息

请求行 RequestLine

```
DELETE /clusters/{cluster_id}/ip/{ip}?releaseInstance=true|false HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
cluster_id	string	是	集群 ID
ip	string	是	节点 IP 地址，经典网络为公网 IP，VPC 为内网 IP。
releaseInstance	bool	可选	移除节点同时释放 ECS 实例(仅能释放按量付费)。

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 Accepted
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

无

示例

请求示例

```
DELETE /clusters/Cccfd68c474454665ace07efce924f75f/ip/10.1.0.123?releaseInstance=false HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 202 Accepted
<公共响应头>
```

4.6.9. 查看镜像列表

查看目前所支持的地域下支持的镜像列表。

请求信息

请求行 RequestLine

```
GET /images HTTP/1.1
```

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
{
  "<RegionID>": {
    "items": [
      {
        "text": "<ImageName>",
        "value": "<ImageID>"
      },
      {
        "text": "<ImageName>",
        "value": "<ImageID>"
      }
    ]
  }
}
```

返回体解释

Image 的格式

名称	类型	描述
text	string	镜像名称。
value	String	镜像 ID。

示例

请求示例

```
GET /images HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 200 OK
<公共响应头>
{
  "ap-northeast-1": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "ap-southeast-1": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
```

```
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "ap-southeast-2": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "ap-southeast-3": {
    "items": [
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      },
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      }
    ]
  },
  "cn-beijing": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "cn-beijing-hpc": {
    "items": [
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      },
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      }
    ]
```

```
    ]
  },
  "cn-hangzhou": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "cn-hongkong": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "cn-huhehaote": {
    "items": [
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      },
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      }
    ]
  },
  "cn-qingdao": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "cn-shanghai": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
```

```
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "cn-shenzhen": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "cn-zhangjiakou": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "eu-central-1": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  },
  "us-east-1": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
```

```
  },
  "us-west-1": {
    "items": [
      {
        "text": "CentOS 7.4 64位",
        "value": "centos_7_04_64_20G_alibase_201701015.vhd"
      },
      {
        "text": "Ubuntu 16.04 64位",
        "value": "ubuntu_16_0402_64_20G_alibase_20170818.vhd"
      }
    ]
  }
}
```

4.6.10. 重置节点

重置集群中的某个节点。

> **说明** 重置过程中会替换系统盘，需要提前做好数据备份。

请求信息

请求行 RequestLine

```
POST /clusters/{cluster_id}/instances/{instance_id}/reset HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
cluster_id	string	是	集群 ID
instance_id	string	是	ECS实例 ID

特有请求头 RequestHead

无，请参考 公共请求头部。

请求体 RequestBody

```
{
  "password": "ECS实例root登录密码",
  "ecs_image_id": "操作系统镜像",
  "release_eip_flag": "是否需要在集群配置完成后释放EIP"
}
```

请求体解析

名称	类型	是否必须	描述
password	String	是	ECS 实例密码。
ecs_image_id	String	是	ECS 使用的系统镜像 ID。
release_eip_flag	bool	可选	配置完集群后是否释放 EIP，默认为 *false*。

ecs_image_id 列表

请参考文档 查看镜像列表 获取 ecs_image_id 列表。如果您需要自定义集群的 ECS 镜像的 ID，需要保证 ECS 镜像满足以下条件：

- 操作系统：Ubuntu、Centos。
- Linux Kernel version ≥ 3.18，用于支持 overlayfs 以及 overlay network。
- 镜像中删除 */etc/docker/key.json* 文件。

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 OK
```

特有返回头 ResponseHead

无，请参考 公共返回头部。

返回体 ResponseBody

```
{
  "cluster_id": "string",
  "request_id": "string",
  "task_id": "string"
}
```

示例

请求示例

```
POST /clusters/Cccfd68c474454665ace07efce924f75f/instances/i-xx/reset HTTP/1.1
<公共请求头>
```

返回示例

```
HTTP/1.1 202 Accepted
<公共响应头>
{
  "cluster_id": "c2ac959c94acc4e86aca4e68bdf7c1987",
  "request_id": "B145E765-2800-40E5-9167-9E999574ABF8",
  "task_id": "T-5a544d2b9645f75f2e00003d"
}
```

4.7. 应用API调用方式

4.7.1. 调用方式

应用管理 REST API 需要指向集群的接入点地址，并通过自签名证书的 HTTPS 请求和集群进行交互。

获取集群 Endpoint 和证书

控制台方式

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 **集群**。
3. 选择需要查看的集群并单击 **管理**。

4. 您可以查看集群的 endpoint 并单击 **下载证书** 下载集群证书。

通过 API 访问，您需要将截图里命令中的 tcp 相应改为 https。

编程方式获取

您需要先通过集群管理的 API 获取：

1. 获取集群的 `master_url` 字段值。更多详细信息，参见 查看集群信息。
2. 获取集群的证书。更多详细信息，参见 查看集群证书。

API 返回结果：

```
{
  "ca": "string", ##认证机构证书，ca.pem
  "cert": "string", ##用户公钥证书，cert.pem
  "key": "string"  ##用户私钥证书，key.pem
}
```

推荐将返回结果的三个 string 的内容保存为一个目录下的三个文件 *ca.pem*、*cert.pem*、*key.pem*。大部分的工具或编程框架都是以文件的方式加载 https 证书.

调用应用管理的 API

假设您的集群名称为 `ClusterName`，并且已经将上面三个证书存储到 *~/.docker/aliyun/ClusterName* 目录下。上面获得的 `master_url` 地址为 `https://123.123.123.123:1234`。

应用 API 列表

详见 应用API列表。

下面以查看应用列表接口为例（`context path` 为 `/projects/`）。

curl 方式

```
# 提示: 请注意你的 curl 版本，您可能需要升级你的 curl.
curl --insecure --cert ~/.docker/aliyun/ClusterName/cert.pem --key ~/.docker/aliyun/ClusterName/key.pem h
ttps://123.123.123.123:1234/projects/
```

PHP 方式

```
<?php
   $ch = curl_init();
   curl_setopt($ch, CURLOPT_URL, "https://123.123.123.123:1234/projects/");
   curl_setopt($ch, CURLOPT_SSLKEY, "~/.docker/aliyun/ClusterName/key.pem");
   curl_setopt($ch, CURLOPT_CAINFO, "~/.docker/aliyun/ClusterName/ca.pem");
   curl_setopt($ch, CURLOPT_SSLCERT, "~/.docker/aliyun/ClusterName/cert.pem");
   $result=curl_exec($ch);
   echo $result;
   curl_close($ch);
?>
```

Python 方式

```
import requests
res = requests.get('https://123.123.123.123:1234/projects/', verify='~/.docker/aliyun/ClusterName/ca.pem', c
ert=('~/.docker/aliyun/ClusterName/cert.pem', '~/.docker/aliyun/ClusterName/key.pem'))
print res.content
```

JAVA 方式

添加 Maven 依赖：

```
<dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.5.1</version>
    </dependency>
    <dependency>
      <groupId>org.bouncycastle</groupId>
      <artifactId>bcpkix-jdk15on</artifactId>
      <version>1.52</version>
    </dependency>
```

代码示例：

```
import java.nio.file.Path;
    import java.nio.charset.Charset;
    import java.nio.file.Files;
```

```
import java.nio.file.Paths;
import java.security.KeyFactory;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.spec.PKCS8EncodedKeySpec;
import javax.net.ssl.SSLContext;
import org.bouncycastle.openssl.PEMKeyPair;
import org.bouncycastle.openssl.PEMParser;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.conn.ssl.SSLConnectionSocketFactory;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.ssl.SSLContexts;
import org.apache.http.util.EntityUtils;
public class Test {
  public static void main(String[] argc) throws Exception {
    final char[] KEY_STORE_PASSWORD = "".toCharArray();
    //获取证书地址
    Path caCertPath = Paths.get("~/.docker/aliyun/ClusterName/ca.pem");
    Path clientCertPath = Paths.get("~/.docker/aliyun/ClusterName/cert.pem");
    Path clientKeyPath = Paths.get("~/.docker/aliyun/ClusterName/key.pem");
    final CertificateFactory cf = CertificateFactory.getInstance("X.509");
    final Certificate caCert = cf.generateCertificate(Files.newInputStream(caCertPath));
    final Certificate clientCert = cf.generateCertificate(
        Files.newInputStream(clientCertPath));
    final PEMKeyPair clientKeyPair = (PEMKeyPair) new PEMParser(
        Files.newBufferedReader(clientKeyPath,
            Charset.defaultCharset()))
        .readObject();
    final PKCS8EncodedKeySpec spec = new PKCS8EncodedKeySpec(
        clientKeyPair.getPrivateKeyInfo().getEncoded());
    final KeyFactory kf = KeyFactory.getInstance("RSA");
    final PrivateKey clientKey = kf.generatePrivate(spec);
    //设置信任的证书
    final KeyStore trustStore = KeyStore.getInstance(KeyStore.getDefaultType());
    trustStore.load(null, null);
    trustStore.setEntry("ca", new KeyStore.TrustedCertificateEntry(caCert), null);
    //设置私钥
    final KeyStore keyStore = KeyStore.getInstance(KeyStore.getDefaultType());
    keyStore.load(null, null);
    keyStore.setCertificateEntry("client", clientCert);
    keyStore.setKeyEntry("key", clientKey, KEY_STORE_PASSWORD, new Certificate[]{clientCert});
    SSLContext sslContext = SSLContexts.custom()
        .loadTrustMaterial(trustStore, null)
        .loadKeyMaterial(keyStore, KEY_STORE_PASSWORD)
        .build();
    SSLConnectionSocketFactory sslsf = new SSLConnectionSocketFactory(
        sslContext,
        SSLConnectionSocketFactory.getDefaultHostnameVerifier());
    //httpclient连接
    CloseableHttpClient httpclient = HttpClients.custom()
```

```
            .setSSLSocketFactory(sslsf)
            .build();
        try {
          HttpGet httpget = new HttpGet("https://123.123.123.123:1234/projects/");
          CloseableHttpResponse response = httpclient.execute(httpget);
          try {
            System.out.println("----------------------------------------");
            String bodyAsString = EntityUtils.toString(response.getEntity());
            System.out.println(bodyAsString);
          } finally {
            response.close();
          }
        } finally {
          httpclient.close();
        }
      }
    }
```

4.8. 应用API列表

4.8.1. 查看应用实例列表

查看您在容器集群中创建的所有应用实例列表。

请求信息

请求行 RequestLine

```
GET /projects/ HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
q	string	否	应用名称
services	boolean	否	是否包含应用的服务信息，缺省值 *true*
containers	boolean	否	是否包含服务的容器信息，缺省值 *true*

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

```
[
  {
    "name": "string",
    "description": "string",
    "template": "string",
    "version": "string",
    "created": "datetime",
    "updated": "datetime",
    "desired_state": "string",
    "current_state": "string",
    "environment": {
      "key": "value",
      ...
    }
    "services": [
      ...
    ]
  }
]
```

返回体解释

Project 的格式

名称	类型	描述
name	string	应用名称
description	string	应用描述
template	string	应用 Compose 模板
version	string	应用版本
created	datetime	应用创建时间
updated	datetime	应用更新时间
desired_state	string	期望状态 （如果当前状态是中间状态时，期望状态指明变迁终态）
current_state	string	当前状态
environment	map	环境变量 key/value
services	array	服务列表

示例

请求示例

```
GET /projects HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
Content-Type:application/json;charset=UTF-8
[
 {
  "name": "test",
  "description": "This is a test application",
  "template": "...",
  "version": "1.0",
  "environment": {
   "COMPOSE_PROJECT_NAME": "test"
  },
  "created": "2016-02-02T07:45:13.113833319Z",
  "updated": "2016-02-02T07:45:16.03142154Z",
  "desired_state": "running",
  "current_state": "running",
  "services": [
   ...
  ]
 },
 ...
]
```

4.8.2. 创建应用实例

创建一个新的应用实例。

请求信息

请求行 RequestLine

```
POST /projects/ HTTP/1.1
```

请求行参数 URI Param

无

特有请求头 RequestHead

```
Content-Type: application/json
```

请求体 RequestBody

JSON object

```
{
  "name": "string",
  "description": "string",
  "template": "string",
  "version": "string",
  "environment": {
    "key": "value",
    ...
  }
}
```

请求体解释

名称	类型	必须	描述
name	string	是	应用名称。名称为 1~64 个字符，可包含数字，英文字符和连字符（-）,且不能以连字符（-）开头。
description	string	否	应用描述。
template	string	是	字符串格式的应用的 Compose yaml 模板，注意需要按照 JSON 格式进行转义。
version	string	否	应用版本，缺省值为 *1.0*。
environment	map	否	key/value 用于替换 Compose 模板的变量参数。
latest_image	bool	否	创建应用前，是否需要更新镜像。

返回信息

返回行 ResponseLine

```
HTTP/1.1 201 Created
```

特有返回头 ResponseHead

```
Location /projects/<name>
```

示例

请求示例

```
POST /projects HTTP/1.1
Content-Type: application/json
{
 "name": "test",
 "description": "This is a test application",
 "template": "web:\r\n  image: nginx",
 "version": "1.0",
 "environment": {
  "USER": "abc",
  "PWD": "password"
 }
}
```

返回示例

```
HTTP/1.1 201 Created
Location /projects/test
```

4.8.3. 删除应用实例

根据应用名称，删除应用实例。

请求信息

请求行 RequestLine

```
DELETE /projects/{name}?force={force}&v={volume} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用实例名称。
force	string	否	是否强制删除，可选值为 *true*、*1* 或 *false*、*0*，缺省为*false* 。
volume	string	否	是否删除容器卷，可选值为 *true*、*1* 或 *false*、*0*，缺省为 *false*。

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
DELETE /projects/test HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.4. 查看应用实例

根据应用实例名称，查看实例的详细信息。

请求信息

请求行 RequestLine

```
GET /projects/{name} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用实例名称

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

```
{
  "name": "string",
  "description": "string",
  "template": "string",
  "version": "string",
  "created": "datetime",
  "updated": "datetime",
  "desired_state": "string",
  "current_state": "string",
  "environment": {
    "key": "value",
    ...
  }
  "services": [
    ...
  ]
}
```

返回体解释

应用实例的格式

名称	类型	描述
name	string	应用名称
description	string	应用描述
template	string	应用Compose模板
version	string	应用版本
created	datetime	应用创建时间
updated	datetime	应用更新时间
desired_state	string	期望状态 （如果当前状态是中间状态时，期望状态指明变迁终态）
current_state	string	当前状态
environment	map	环境变量key/value
services	array	服务列表

示例

请求示例

```
GET /projects/test HTTP/1.1
```

返回示例

```
HTTP/1.1 200 Ok
Content-Type:application/json;charset=UTF-8
{
 "name": "test",
 "description": "This is a test application",
 "template": "...",
 "version": "1.0",
 "environment": {
  "COMPOSE_PROJECT_NAME": "test"
 },
 "created": "2016-02-02T07:45:13.113833319Z",
 "updated": "2016-02-02T07:45:16.03142154Z",
 "desired_state": "running",
 "current_state": "running",
 "services": [
  ...
 ]
}
```

4.8.5. 启动应用实例

启动一个应用实例，会依照服务之间依赖顺序进行启动操作。

请求信息

请求行 RequestLine

```
POST /projects/{name}/start HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用实例名称

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test/start HTTP/1.1
```

返回示例

HTTP/1.1 200 OK

4.8.6. 终止应用实例

向一个应用实例所包含的所有容器发送信号（Signal），会依照服务之间依赖顺序的逆序进行发送。缺省信号会终止所有容器。

请求信息

请求行 RequestLine

```
POST /projects/{name}/kill?signal={signal} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用实例名称
signal	string	否	缺省为 *KILL*

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test/kill HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.7. 停止应用实例

停止一个应用实例所包含的所有容器，会依照服务之间依赖顺序的逆序进行停止操作。

请求信息

请求行 RequestLine

```
POST /projects/{name}/stop?t={timeout} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用实例名称
timeout	int	否	停止容器的超时时间（以秒为单位），缺省为 *10*

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test/stop HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.8. 重新部署应用

重新部署一个应用实例。

请求信息

请求行 RequestLine

```
POST /projects/{name}/redeploy HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用实例名称

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test/redeploy HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.9. 更新应用配置

更新一个应用实例的配置信息。

请求信息

请求行 RequestLine

```
POST /projects/{name}/update HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用实例名称

特有请求头 RequestHead

```
Content-Type: application/json
```

请求体 RequestBody

JSON object

```
{
  "description": "string",
  "template": "string",
  "version": "string",
  "latest_image": true,
  "environment": {
    "key": "value",
    ...
  }
}
```

请求体解析

名称	类型	必须	描述
description	string	否	更新的应用描述。
template	string	是	更新的 Compose yaml 模板，注意需要按照 JSON 格式进行转义。
version	string	否	更新的应用版本，更新的版本应和原有版本不同，否则会返回 HTTP code 409。
latest_image	bool	否	是否拉取最新镜像
environment	map	环境变量	key/value 用于替换 Compose 模板的环境变量

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 Accepted
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test/update HTTP/1.1
Content-Type: application/json
{
 "description": "This is a test application",
 "template": "web:\r\n  image: nginx",
 "version": "2.0",
 "latest_image": true,
 "environment": {
  "USER": "abc",
  "PWD": "newpwd"
 }
}
```

返回示例

```
HTTP/1.1 202 Accepted
```

4.8.10. 蓝绿应用更新

蓝绿更新一个应用实例的配置信息。

请求信息

请求行 RequestLine

```
POST /projects/{name}/update HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用名称

特有请求头 RequestHead

```
Content-Type: application/json
```

请求体 RequestBody

JSON object

```
{
  "update_method": "string",
  "description": "string",
  "template": "string",
  "version": "string",
  "latest_image": true,
  "environment": {
    "key": "value",
    ...
  }
}
```

请求体解析

名称	类型	必须	描述
update_method	string	是	可选值 *blue-green*。
description	string	否	更新的应用描述。
template	string	是	更新的 Compose yaml 模板，注意需要按照 JSON 格式进行转义。
version	string	是	更新的应用版本，更新的版本应和原有版本不同，否则会返回 HTTP code 409。
latest_image	bool	否	是否拉取最新镜像
environment	map	环境变量	key/value 用于替换 Compose 模板的环境变量

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 Accepted
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test_01/update HTTP/1.1
Content-Type: application/json
{
 "update_method": "blue-green",
 "description": "This is a test_01 application",
 "template": "web:\r\n  image: nginx",
 "version": "2.0",
 "latest_image": true,
 "environment": {
  "USER": "abc",
  "PWD": "newpwd"
    ...
 }
}
```

返回示例

```
HTTP/1.1 202 Accepted
```

4.8.11. 蓝绿发布确认

确认蓝绿发布一个应用实例。

请求信息

请求行 RequestLine

```
POST /projects/{name}/confirm-update?force=(true or false) HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用名称
confirm-update	bool	是	当传递 *True* 的时候，在回滚完成的时候会设置当前应用的路由权重为100%，旧的应用权重为0%；当传递 *False* 的时候，在回滚完成的时候会忽略权重设置，以客户的权重设置为准。

特有请求头 RequestHead

无

请求体 RequestBody

无

请求体解析

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 Accepted
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test_01/confirm-update?force=true HTTP/1.1
```

返回示例

```
HTTP/1.1 202 Accepted
```

4.8.12. 蓝绿发布回滚

回滚一个蓝绿更新的应用实例。

请求信息

请求行 RequestLine

```
POST /projects/{name}/rollback-update?force=(true or false) HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
name	string	是	应用名称
rollback-update	bool	是	目前传递 *true* 即可

特有请求头 RequestHead

无

请求体 RequestBody

无

请求体解析

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 202 Accepted
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /projects/test_01/rollback-update?force=true HTTP/1.1
```

返回示例

```
HTTP/1.1 202 Accepted
```

4.8.13. 查看服务实例列表

查看您在容器集群中创建的所有服务实例列表。

请求信息

请求行 RequestLine

```
GET /services/ HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
q	string	否	服务名称
containers	boolean	否	是否包含服务的容器信息，缺省值 *true*

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

```
[
 {
   "id": "string",
   "name": "string",
   "project": "string",
   "description": "string",
   "created": "datetime",
   "updated": "datetime",
   "desired_state": "string",
   "current_state": "string",
   "definition": {
     "key": "value",
     ...
   },
   "extensions": {
     "key": "value",
     ...
   },
   "containers": {
     "key": "value",
     ...
   }
 },
 ...
]
```

返回体解释

服务实例的格式

名称	类型	描述
id	string	服务 ID
name	string	服务名称
project	string	应用名称
created	datetime	服务创建时间
updated	datetime	服务更新时间
desired_state	string	期望状态 （如果当前状态是中间状态时，期望状态指明变迁终态）

名称	类型	描述
current_state	string	当前状态
definition	map	Compose 中服务定义 key/value
extensions	map	容器服务 Compose 中服务扩展 key/value
containers	map	服务中所包含容器 key （容器 id）/value（属性）

示例

请求示例

```
GET /services/ HTTP/1.1
```

返回示例

```
HTTP/1.1 200 Ok
Content-Type:application/json;charset=UTF-8
[
 {
  "id": "wordpress_db",
  "name": "db",
  "project": "wordpress",
  "definition": {
   "environment": [
    "MYSQL_ROOT_PASSWORD=password"
   ],
   "image": "mysql:5.7",
   "restart": "always"
  },
  "extensions": {
   "scale": 1,
   "logs": [
    "/var/log/mysql"
   ]
  },
  "created": "2016-04-21T13:36:32.440646459Z",
  "updated": "2016-04-21T13:36:33.270308958Z",
  "desired_state": "running",
  "current_state": "running",
  "containers": {
   "5616f05d27516b3502a391fd2ca9d312cabffa5ad431bf261ea81f4ceabd476e": {
    "name": "/wordpress_db_1",
    "node": "10.246.2.3",
    "ip": "10.0.0.2",
    "running": true,
    "status": "running",
    "health": "success"
   }
  }
 }
]
```

4.8.14. 查看服务实例

根据服务实例 ID 查看详细信息。

请求信息

请求行 RequestLine

```
GET /services/{service_id} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
service_id	string	是	服务实例 ID, 格式为 {project_name}_{service_name}

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

```
{
  "id": "string",
  "name": "string",
  "project": "string",
  "description": "string",
  "created": "datetime",
  "updated": "datetime",
  "desired_state": "string",
  "current_state": "string",
  "definition": {
    "key": "value",
    ...
  },
  "extensions": {
    "key": "value",
    ...
  },
  "containers": {
    "key": "value",
    ...
  }
}
```

返回体解释

服务实例的格式

名称	类型	描述
id	string	服务 ID
name	string	服务名称
project	string	应用名称
created	datetime	服务创建时间
updated	datetime	服务更新时间
desired_state	string	期望状态 （如果当前状态是中间状态时，期望状态指明变迁终态）
current_state	string	当前状态
definition	map	Compose 中服务定义 key/value
extensions	map	容器服务 Compose 中服务扩展 key/value
containers	map	服务中所包含容器 key （容器 ID）/value（属性）

示例

请求示例

```
GET /services/wordpress_db HTTP/1.1
```

返回示例

```
HTTP/1.1 200 Ok
Content-Type:application/json;charset=UTF-8
{
 "id": "wordpress_db",
 "name": "db",
 "project": "wordpress",
 "definition": {
  "environment": [
   "MYSQL_ROOT_PASSWORD=password"
  ],
  "image": "mysql:5.7",
  "restart": "always"
 },
 "extensions": {
  "scale": 1,
  "logs": [
   "/var/log/mysql"
  ]
 },
 "created": "2016-04-21T13:36:32.440646459Z",
 "updated": "2016-04-21T13:36:33.270308958Z",
 "desired_state": "running",
 "current_state": "running",
 "containers": {
  "5616f05d27516b3502a391fd2ca9d312cabffa5ad431bf261ea81f4ceabd476e": {
   "name": "/wordpress_db_1",
   "node": "10.246.2.3",
   "ip": "10.0.0.2",
   "running": true,
   "status": "running",
   "health": "success"
  }
 }
}
```

4.8.15. 删除数据卷

删除指定数据卷。

请求信息

请求行 RequestLine

```
Delete /volumes/{name} HTTP/1.1
```

请求行参数 URI Param

无

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 204 No Content
```

返回体 ResponseBody

无

4.8.16. 启动服务实例

启动一个服务实例中所有容器。

请求信息

请求行 RequestLine

```
POST /services/{service_id}/start HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
service_id	string	是	服务实例 ID, 格式为 {project_name}_{service_name}

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /service/test_web/start HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.17. 停止服务实例

停止一个服务实例中所有容器。

请求信息

请求行 RequestLine

```
POST /services/{service_id}/stop?t={timeout} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
service_id	string	是	服务实例 ID, 格式为 {project_name}_{service_name}
timeout	int	否	停止容器的超时时间（以秒为单位），缺省为 *10*

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /services/test_web/stop HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.18. 终止服务实例

向一个服务实例中所有容器发送信号（Signal），缺省信号会终止所有容器。

请求信息

请求行 RequestLine

```
POST /services/{service_id}/kill?signal={signal} HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
service_id	string	是	服务实例 ID, 格式为 {project_name}_{service_name}
signal	string	否	缺省为 *KILL*

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

```
POST /services/test_web/kill HTTP/1.1
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.19. 伸缩服务实例

伸缩一个服务实例中所包含容器到指定数量。

请求信息

请求行 RequestLine

```
POST /services/{service_id}/scale HTTP/1.1
```

请求行参数 URI Param

名称	类型	是否必须	描述
service_id	string	是	服务实例 ID, 格式为 {project_name}_{service_name}

特有请求头 RequestHead

无

请求体 RequestBody

JSON object

```
{
  "type": "string",
  "value": "int"
}
```

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

特有返回头 ResponseHead

无

返回体 ResponseBody

无

示例

请求示例

将 `test_web` 服务定义的容器实例数改变为 3 个。

```
POST /services/test_web/scale HTTP/1.1
Content-Type: application/json
{
  "type": "scale_to",
  "value": 3
}
```

返回示例

```
HTTP/1.1 200 OK
```

4.8.20. 创建数据卷

在集群中创建一个数据卷。

请求信息

请求行 RequestLine

```
POST /volumes/create HTTP/1.1
```

请求行参数 URI Param

无

特有请求头 RequestHead

```
Content-Type: application/json
```

请求体 RequestBody

JSON object

```
{
  "name": "***",
  "driver": "***",
  "driverOpts": {
    "para1": "value",
    "para2": "value",
    ...
  }
}
```

请求体解释

名称	类型	必须	描述
name	string	是	数据卷名称。名称为 1~64 个字符，可包含数字，英文字符和连字符（-），且不能以连字符（-）开头。

名称	类型	必须	描述
driver	string	是	数据卷类型。当前支持 ossfs，nas 类型。
driverOpts	DriverOptions	是	数据卷配置参数选项，不同数据卷类型的参数不相同；OSS 数据卷为 OSSOpts，NAS 数据卷类型为 NASOpts。

各类型数据卷的定义如下所示：

OSSOpts：

名称	类型	必须	描述
bucket	string	是	OSS 存储的 bucket 名称，可以从 OSS 控制台获取。
ak_id	string	是	用户访问 OSS 资源所需的 Access Key ID。 参见 如何获取 Access ID 和 Access Key。
ak_secret	string	是	用户访问 OSS 资源所需的 Access Key Secret。
url	string	是	OSS bucket 所提供的域名，可以从 OSS 控制台获取。
no_stat_cache	string	是	文件缓存，如果需要在不同机器间同步同一个文件的修改，请关闭缓存。
other_opts	string	是	连接 OSS 的配置参数，详见 FAQ。

NASOpts：

名称	类型	必须	描述
diskid	string	是	NAS 实例的磁盘 ID。
host	string	是	NAS 实例的接入点域名，详见 NAS 使用文档。
path	string	是	NAS 路径下的子目录，详见 NAS 使用文档。

名称	类型	必须	描述
mode	string	是	配置数据卷的访问权限。

返回信息

返回行 ResponseLine

```
HTTP/1.1 201 Created
```

返回体 ResponseBody

JSON object

```
{
"Name": "volume",
"Driver": "***",
"Mountpoint": "/mnt/acs_mnt/**/***",
"Labels": null,
"Scope": ""
}
```

返回体解释

名称	类型	描述
Name	string	数据卷名称。
Driver	string	数据卷的驱动类型：ossfs，nas等。
Mountpoint	string	数据卷在主机上的挂载点：*/mnt/acs_mnt/**/***。
Labels	map[string]string	数据卷的元数据信息。
Scope	string	描述数据卷管理范围。*global* 表示集群级别；*local* 表示主机内部。

OSS 数据卷示例

请求示例

```
{
  "name": "ossvolume",
  "driver": "ossfs",
  "driverOpts": {
    "bucket": "aliyun-docker",
    "ak_id": "****",
    "ak_secret": "*****",
    "url": "oss-cn-hangzhou.aliyuncs.com",
    "no_stat_cache": "true",
    "other_opts": "-o allow_other -o default_permission=666"
 }
}
```

返回示例

```
{
  "Name": "ossvolume",
  "Driver": "ossfs",
  "Mountpoint": "/mnt/acs_mnt/ossfs/aliyun-docker",
  "Labels": null,
  "Scope": ""
}
```

NAS 数据卷示例

请求示例

```
{
  "name": "nasvolume",
  "driver": "nas",
  "driverOpts": {
    "diskid": "1234556",
    "host": "1234556-gpp53.cn-hangzhou.nas.aliyuncs.com",
    "path": "/abc",
    "mode": "755"
 }
}
```

返回示例

```
{
  "Name": "nasvolume",
  "Driver": "nas",
  "Mountpoint": "/mnt/acs_mnt/nas/nasvolume",
  "Labels": null,
  "Scope": ""
}
```

4.8.21. 查看数据卷

查看一个数据卷的详细信息。

请求信息

请求行 RequestLine

```
Get /volumes/{name} HTTP/1.1
```

请求行参数 URI Param

无

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

返回体 ResponseBody

JSON object

```
{
  "Name": "volume",
  "Driver": "***",
  "Mountpoint": "/mnt/acs_mnt/**/***",
  "Labels": null,
  "Scope": "",
  "Node": "***",
  "Refs": []
}
```

名称	类型	描述
Name	string	数据卷名称。
Driver	string	数据卷的驱动类型：ossfs、nas等。
Mountpoint	string	数据卷在主机上的挂载点：*/mnt/acs_mnt/**/****。
Labels	map[string]string	数据卷的元数据信息。
Scope	string	描述数据卷管理范围。*global* 表示集群级别；*local* 表示主机内部。
Node	string	数据卷所在节点的主机名。

名称	类型	描述
Refs	[]VolumeRef	使用这个数据卷的容器列表信息，详见 VolumeRef 定义。

VolumeRef 数据类型：

名称	类型	描述
Name	string	容器名
ID	string	容器 ID

示例

返回示例

```
{
  "Name": "volume",
  "Driver": "ossfs",
  "Mountpoint": "/mnt/acs_mnt/ossfs/bucket",
  "Labels": null,
  "Scope": "",
  "Node": "asd903233s23q-node1",
  "Refs": []
}
```

4.8.22. 查看数据卷列表

查看当前集群所有数据卷。

请求信息

请求行 RequestLine

```
Get /volumes HTTP/1.1
```

请求行参数 URI Param

无

特有请求头 RequestHead

无

请求体 RequestBody

无

返回信息

返回行 ResponseLine

```
HTTP/1.1 200 OK
```

返回体 ResponseBody

JSON object

```
{
  "Name": "volume",
  "Driver": "***",
  "Mountpoint": "/mnt/acs_mnt/**/***",
  "Labels": null,
  "Scope": "",
  "Node": "***",
  "Refs": []
}
```

名称	类型	描述
Name	string	数据卷名称。
Driver	string	数据卷的驱动类型：ossfs、nas 等。
Mountpoint	string	数据卷在主机上的挂载点：*/mnt/acs_mnt/**/****。
Labels	map[string]string	数据卷的元数据信息。
Scope	string	描述数据卷管理范围。global 表示集群级别；local 表示主机内部。
Node	string	数据卷所在节点的主机名。
Refs	[]VolumeRef	使用这个数据卷的容器列表信息，详见 VolumeRef 定义。

VolumeRef 数据类型：

名称	类型	描述
Name	string	容器名
ID	string	容器 ID

示例

返回示例

```
{
 "Volumes": [
  {
   "Name": "docker2",
   "Driver": "ossfs",
   "Mountpoint": "/mnt/acs_mnt/ossfs/bucket2",
   "Labels": null,
   "Scope": "",
   "Node": "b347bcf0e68b",
   "Refs": []
  },
  {
   "Name": "docker1",
   "Driver": "ossfs",
   "Mountpoint": "/mnt/acs_mnt/ossfs/bucket1",
   "Labels": null,
   "Scope": "",
   "Node": "b347bcf0e68b",
   "Refs": []
  }
 ]
}
```

4.9. 触发器

4.9.1. 触发器简介

触发器是容器服务提供的简单快捷地进行重新部署和资源伸缩的 API。

由于标准的 API 需要保证安全性，因此需要严格的鉴权。但是对于需要与其他三方系统（例如 Jenkins 或者其他持续集成的 CI/CD 系统）集成的场景来说，所需要的权限是有限的，可能仅仅需要消息通知。因此为了保证安全性与便捷性，带有部分鉴权策略，并且可以灵活调用的 API 被广泛应用于持续集成持续交付的场景中。

容器服务目前提供重新部署触发器和资源伸缩触发器。

- 重新部署触发器

  您可以与自己的监控系统进行集成，当发现系统异常时进行重新部署；您也可以与容器 Hub 进行集成，当容器新镜像构建完成后，可以自动进行部署等。

- 资源伸缩触发器

  您可以通过调用资源伸缩触发器来实现容器伸缩。

4.9.2. 重新部署触发器

本例中，将与容器镜像服务、阿里云 code 进行集成，当容器新镜像构建完成后，可以自动进行部署等。

前提条件

- 准备一个可用的集群，参见 创建集群。
- 准备一个代码仓库，推荐使用阿里云容器镜像服务，支持阿里云code、github、Bitbucket、私有 GitLab 和本地仓库等，您可以在 alicode 上新建一个代码库。

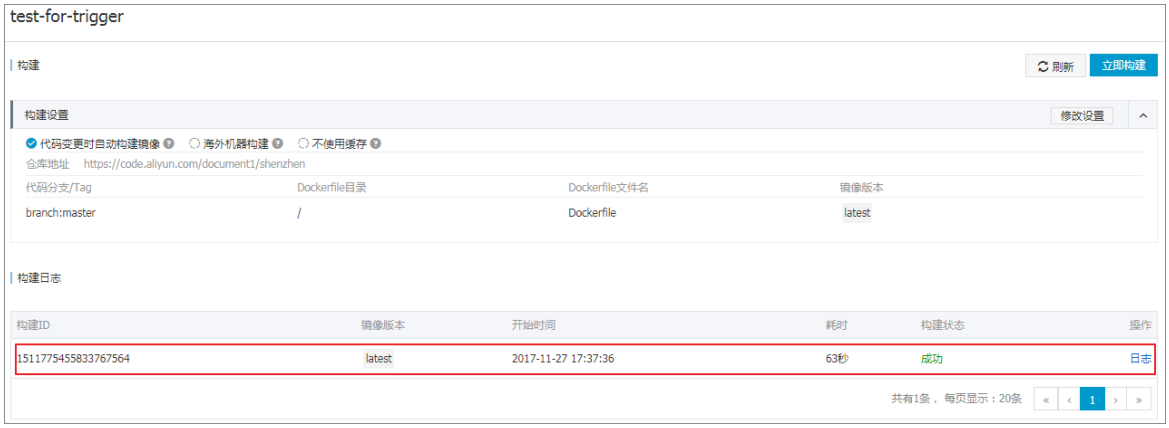
本例中，为了方便操作，我们准备了一个已有的代码库，您可以把它派生成自己的代码库，只需要打开 重新部署触发器示例，单击 **派生**（fork project）。如果您看不到该按钮，请先登录。

操作步骤

自动创建 Docker 容器镜像

1. 登录 容器镜像服务管理控制台。
2. 点击右上角的 **创建镜像仓库** 按钮，在接下来的页面中填入相关信息。
3. 设置代码源处，已经绑定了你的alicode账号，选择准备好的代码库。
4. 勾选 **代码变更时自动构建镜像**，选择 **Branch:master**，Dockerfile目录填写 “/ “，镜像版本填写 “latest” 。

5. 镜像仓库创建成功后，单击页面右上角的 **立即构建**，可以从构建记录里看到构建结果。



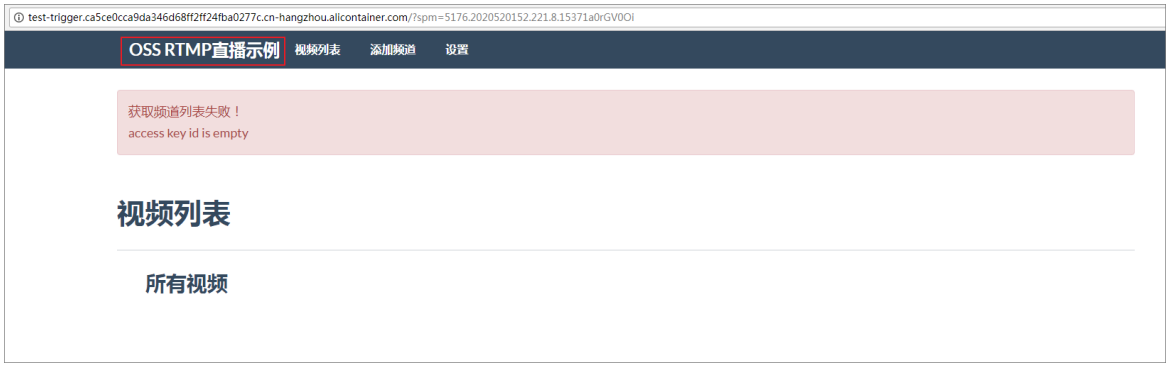
创建应用

1. 找到上面构建的镜像仓库，进入对应的管理页面。
2. 在左侧导航栏，单击 **基本信息**，然后单击 **部署应用**。您也可以记录镜像的地址，在容器服务管理控制台中，根据集群网络类型，输入正确的镜像地址，创建应用，参见 创建应用。

3. 自动转到容器服务管理控制台的创建应用界面。需要勾选 **检查最新 Docker 镜像**。

4. 在应用配置页，镜像名称已经输入，接下来配置 **简单路由配置** 和 **数据卷**。

5. 回到应用列表页面，等应用就绪后，通过应用的路由地址访问部署好的应用。该应用是一个示例，请注意红框内的内容。



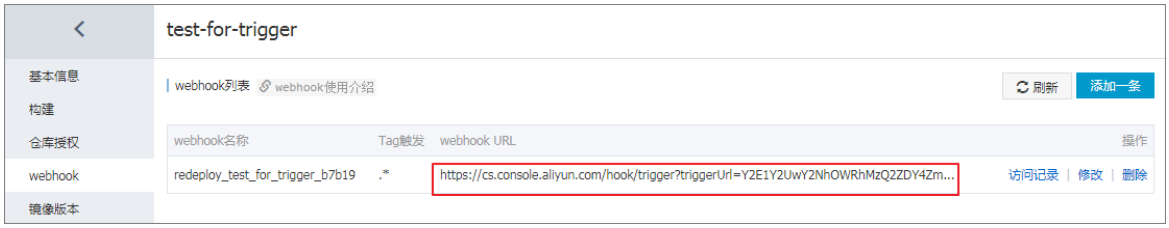
创建触发器并实现重新部署

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏中的 **应用**。
3. 选择目标应用并单击应用的名称。
4. 在应用详情页面，单击 **创建触发器**。

5. 在 **触发器行为** 下拉框中选择 **重新部署**，并单击 **确定**。

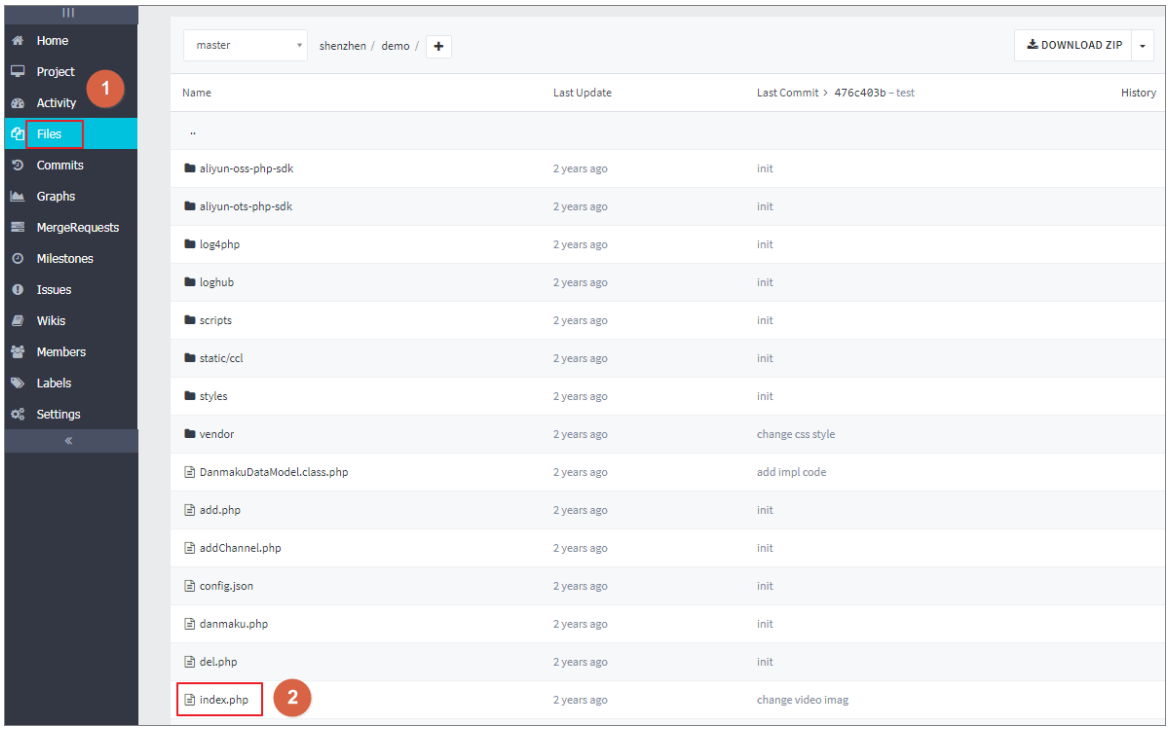
   当您对应用使用的镜像有写权限时，您可以勾选 **关联到镜像更新**。勾选后，当容器新镜像构建完成后，可以使用最新镜像自动对应用进行重新部署。

6. 勾选 **关联到镜像更新**，本示例中，该操作会触发在关联镜像 test-for-trigger 的 Webhook 上自动创建一条 Webhook。
7. 登录 容器镜像控制台。进入目标仓库管理页，单击左侧导航栏的 **Webhook**，查看是否自动创建一条 Webhook 规则。

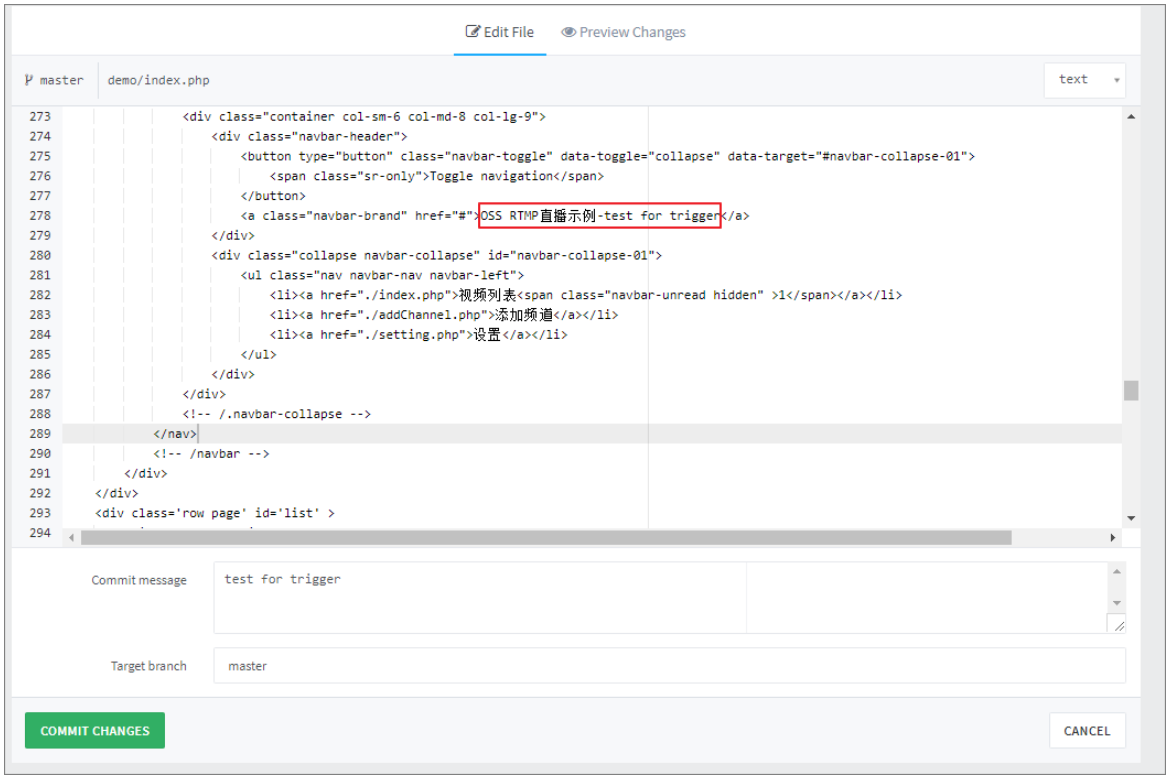


8. 在 阿里云code 中修改代码，会自动构建镜像。

   打开 阿里云code 代码库，单击左边导航栏的 **文件**，找到“demo/index.php”文件，单击编辑。



9. 找到278行，将“OSS RTMP直播示例”改成“OSS RTMP直播示例-test for trigger”，填入提交信息，并提交修改。



10. 提交后，会自动触发镜像构建，构建成功后会触发自动部署。回到应用列表页面，当应用状态变为“就绪”后，访问路由地址，可以看到修改已经自动生效。从而实现代码变更自动触发应用自动部署的功能。

后续操作

您可以通过三方集成系统进行触发，使用 GET 或者 POST 都可以进行触发，例如使用 curl 命令触发。

调用重新部署触发器：

```
curl 'https://cs.console.aliyun.com/hook/trigger?triggerUrl=Y2E1Y2UwY2NhOWRhMzQ2ZDY4ZmYyZmYyNGZiYTAyNzdjfHRlc3QtZm9yLXRyaWdnZXItZGVmYXVsdHxzY2FsaW5nfDE5dXAyajhoOXR1NHN8&secret=48556533517a6f5677747259347565 51fff0c4ccee34046cceef2c97ebacfd62'
```

4.9.3. 资源伸缩触发器

现在有很多客户很关心应用的自动弹性伸缩，有些客户也有自己的监控框架，并希望能跟阿里云容器服务进行集成。阿里云容器服务提供了资源弹性伸缩触发器，并能够跟监控框架集成来实现自定义的服务自动弹性伸缩。

阿里云容器服务会自动采集容器的监控数据，并可以通过集成将监控数据发送到三方的监控框架中。有了监控数据，我们可以在监控框架中定义自己的报警规则，当指标发生报警的时候调用阿里云容器服务提供的触发器来进行容器的扩容或者缩容。 下面用Influxdb，Kapacitor来介绍怎样通过触发器跟监控框架集成实现自定义弹性伸缩。

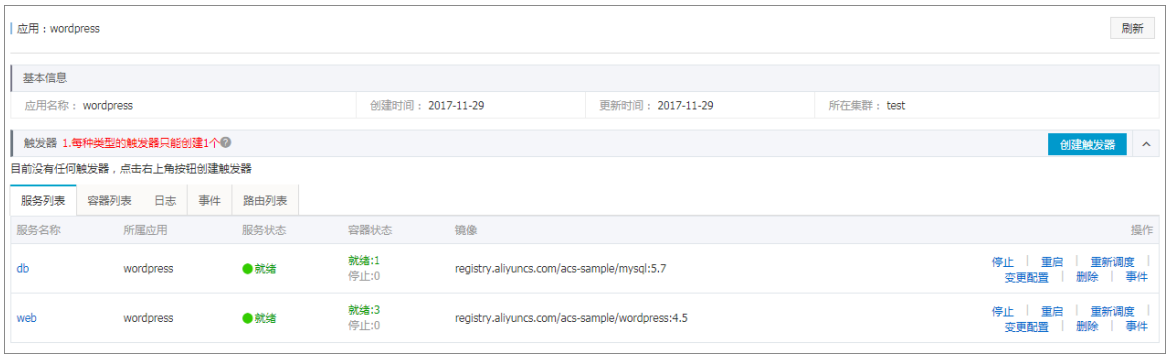
前提条件

- 准备一个可用的集群，参见 创建集群。
- 创建一个 wordpress 示例应用，参见 创建应用。

操作步骤

生成 wordpress 资源伸缩触发器

1. 登录 容器服务管理控制台。
2. 选择创建好的 wordpress 应用，单击应用名称，进入详情页。



3. 单击 **创建触发器**，在弹出的对话框中，选择资源伸缩， 并在 **服务** 下拉框中选择需要设置资源伸缩触发器的服务。

> **说明** 您需要将所对应集群的 Agent 升级到最新版本，才能使用资源伸缩触发器。

4. 此时生成的触发器地址即为 API 的地址。

> **说明** 该 API 支持添加 type 和 step 两个参数，本示例中，调用 `sale out` 的时候会添加参数 &type=scale_out&step=5。

部署监控应用

我们创建一个监控应用，包括监控时序数据库Influxdb、 监控报警框架 Kapacitor 以及界面展现 Grafana。将 wordpress 资源伸缩触发器写到应用配置文件中，达到根据应用负载，资源自动伸缩的目的。

我们使用一个编排模板来部署：

```yaml
version: '2'
services:
 influxdb:
  image: influxdb:0.13
  ports:
   - "8083:8083"
   - "8086:8086"
  container_name: "influxdb"
  labels:
    aliyun.monitoring.addon.influxdb: "http://influxdb:8086"
 grafana:
  image: grafana/grafana:3.0.3-1463994644
  ports:
   - "3000:3000"
  links:
   - influxdb
 kapacitor:
  image: kapacitor:0.13
  ports:
   - "9092:9092"
  volumes:
      - /etc/acs/:/etc/acs/
  environment:
   - KAPACITOR_INFLUXDB_0_URLS_0=http://influxdb:8086
  command: kapacitord -config /etc/kapacitor/kapacitor.conf
```

创建好的应用，如下图所示。

配置Kapacitor报警规则

创建报警规则文件。

在 Kapacitor 中配置报警规则，并当报警时调用扩容触发器 URL。

通过 Web 远程终端或者 Docker Exec 进入 Kapacitor 容器，增加报警规则，比如我们对CPU指标设置报警规则，创建/etc/acs/cpu.tick 文件，内容如下。

```
stream
  // Select just the cpu measurement from our example database.

  |from()
    .measurement('docker_container_cpu')

  |groupBy('aliyun.cluster', 'aliyun.service.id')
  |alert()
    .crit(lambda: "aliyun.cluster"=='xxxxx' AND "aliyun.service.id"=='xxxxx' AND "usage_percent" > 70)

    .post('https://cs.console.aliyun.com/hook/trigger?
    triggerUrl=Y2E1Y2UwY2NhOWRhMzQ2ZDY4ZmYyZmYyNGZiYTAyNzdjfHdvcmRwcmVzc3xzY2FsaW5nfDE5
dXM1cHZjdjQ4cnN8=&secret=xxx&&type=scale_out&step=5')

    .log('/tmp/alerts.log')
```

这里对监控的CPU指标 `docker_container_cpu` 按集群及服务进行聚合然后判断当 usage_percent>70 的时候进行服务扩容。类似，我们也可以增加一个缩容的报警规则。

定义报警规则并启用

在Kapacitor容器中执行如下命令定义并启用报警规则。

```
kapacitor define cpu_alert -type stream  -tick cpu_alert.tick -dbrp telegraf.default;
kapacitor enable cpu_alert
```

这样当CPU的使用率超过 70% 的时候，会自动调用扩容触发器进行容器的扩容。

后续操作

您可以通过三方集成系统进行触发，使用 GET 或者 POST 都可以进行触发，例如使用 **curl** 命令触发。

调用资源伸缩触发器：

> **说明** 调用资源伸缩触发器时，需要在触发器 URL 中手动添加以下参数：

参数名称	必填	语义	可选值
type	是	伸缩类型	缩容：*scale_in*；扩容：*scale_out*
step	是	伸缩数量	正整数，1~100

例如，调用下面的触发器会执行扩容五个容器的操作。

```
curl 'https://cs.console.aliyun.com/hook/trigger?triggerUrl=Y2E1Y2UwY2NhOWRhMzQ2ZDY4ZmYyZmYyNGZiYTAyNzdjfHdvcmRwcmVzc3xzY2FsaW5nfDE5dXM1cHZjdjQ4cnN8=&secret=xxx&&type=scale_out&step=5'
```

4.10. 附录

4.10.1. Docker Registry 相关 API

有关 Docker Registry 相关接口，您可以参考以下官方帮助文档：

Registry API 文档：https://docs.docker.com/registry/spec/api/

Registry API 加签介绍：https://docs.docker.com/registry/spec/auth/