阿里云
容器服务Kubernetes版

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 集群

1.1 Kubernetes集群选型及高可靠推荐配置

1.1.1 ECS选型

本文介绍构建Kubernetes集群时推荐的ECS选型。

集群规划

目前在创建Kubernetes集群时，存在使用很多小规格ECS的情况，这样做存在以下弊端：

- 小规格Woker ECS的网络资源受限。
- 如果一个容器基本可以占用一个小规格ECS，此ECS的剩余资源就无法利用（构建新的容器或者是恢复失败的容器），在小规格ECS较多的情况下，存在资源浪费。

使用大规格ECS的优势：

- 网络带宽大，对于大带宽类的应用，资源利用率高。
- 容器在一台ECS内建立通信的比例增大，减少网络传输。
- 拉取镜像的效率更高。因为镜像只需要拉取一次就可以被多个容器使用。而对于小规格的ECS拉取镜像的次数就会增多。若需要联动ECS伸缩集群，则需要花费更多的时间，反而达不到立即响应的目的。

选择Master节点规格

通过容器服务创建的Kubernetes集群，Master节点上运行着etcd、kube-apiserver、kube-controller等核心组件，对于Kubernetes集群的稳定性有着至关重要的影响，对于生产环境的集群，必须慎重选择Master规格。Master规格跟集群规模有关，集群规模越大，所需要的Master规格也越高。

说明：
您可从多个角度衡量集群规模：节点数量、Pod数量、部署频率、访问量。这里简单的认为集群规模就是集群里的节点数量。

对于常见的集群规模，可以参考如下的方式选择Master节点的规格（对于测试环境，规格可以小一些。下面的选择能尽量保证Master负载维持在一个较低的水平上）：

节点规模	Master规格
1-5个节点	4核8G(不建议2核4G)

节点规模	Master规格
6-20个节点	4核16G
21-100个节点	8核32G
100-200个节点	16核64G

选择Worker节点规格

- 确定整个集群的日常使用的总核数以及可用度的容忍度。

  例如：集群总的核数有160核，可以容忍10%的错误。那么最小选择10台16核ECS，并且高峰运行的负荷不要超过160*90%=144核。如果容忍度是20%，那么最小选择5台32核ECS，并且高峰运行的负荷不要超过160*80%=128核。这样就算有一台ECS出现故障，剩余ECS仍可以支持现有业务正常运行。
- 确定CPU：Memory比例。对于使用内存比较多的应用例如java类应用，建议考虑使用1:8的机型。

选用裸金属神龙服务器

以下两种场景，建议选用裸金属神龙服务器：

- 集群日常规模能够达到1000核。一台神龙服务器至少96核，这样可以通过10～11台神龙服务器即可构建一个集群。
- 快速扩大较多容器。例如：电商类大促，为应对流量尖峰，可以考虑使用神龙服务来作为新增节点，这样增加一台神龙服务器就可以支持很多个容器运行。

使用神龙服务器构建集群，存在以下优势：

- 超强网络： 配备RDMA（Remote Direct Memory Access）技术。通过Terway容器网络，充分发挥硬件性能，跨宿主机容器带宽超过9Gbit/s。
- 计算性能零抖动：自研芯片取代Hypervisor，无虚拟化开销，无资源抢占。
- 安全：物理级别加密，支持Intel SGX加密，可信计算环境，支持区块链等应用。

1.1.2 高可靠推荐配置

为了保证应用可以稳定可靠的运行在Kubernetes里，本文介绍构建Kubernetes集群时的推荐配置。

磁盘类型及大小

磁盘类型

- 推荐选择SSD盘。

- 对于Worker节点，创建集群时推荐选择挂载数据盘。这个盘是专门提供给`/var/lib/docker`存放本地镜像。可避免后续如果镜像太多根磁盘容量不够的问题。在运行一段时间后，本地会存在很多无用的镜像。比较快捷的方式就是，先下线这台机器，重新构建这个磁盘，然后再上线。

磁盘大小

Kubernetes节点需要的磁盘空间也不小，Docker镜像、系统日志、应用日志都保存在磁盘上。创建Kubernetes集群的时候，要考虑每个节点上要部署的Pod数量，每个Pod的日志大小、镜像大小、临时数据，再加上一些系统预留的值。

Kubernetes集群中，ECS操作系统占用3G左右的磁盘空间，建议预留ECS操作系统8G的空间。剩余的磁盘空间由Kubernetes资源对象使用。

是否立即构建Worker节点

创建集群时，节点类型若选择：

- 按量付费，可在创建集群时，构建Worker节点。
- 包年包月，创建集群时，可先不构建Worker节点，根据后续需求，单独购买ECS添加进集群。

网络选择

- 如果需要连接外部的一些服务，如RDS等，则需要考虑复用原有的VPC，而不是创建一个新的VPC。因为VPC间是隔离的，您可以通过创建一个新的交换机，把运行Kubernetes的机器都放在这个交换机下，从而便于管理。
- 在Kubernetes集群创建时提供两种网络插件：Terway和Flannel，可参考如何选择Kubernetes集群网络插件#Terway和Flannel。
- Pod网络CIDR不能设置太小，如果太小，可支持的节点数量就会受限。这个值的设置需要与高级选项中的节点Pod数量综合考虑。例如：Pod网络CIDR的网段是/16，那么就有256*256个地址，如果每个节点Pod数量是128，则最多可以支持512个节点。

使用多可用区

阿里云支持多Region（地域），每个Region下又有不同的可用区。可用区是指在同一地域内，电力和网络互相独立的物理区域。多可用区能够实现跨区域的容灾能力。同时也会带来额外的网络延时。创建Kubernetes集群时，您可选择创建多可用区Kubernetes集群。参见创建多可用区Kubernetes 集群。

声明每个Pod的resource

在使用Kubernetes集群时，经常会遇到：在一个节点上调度了太多的Pod，导致节点负载太高，没法正常对外提供服务的问题。

为避免上述问题，在Kubernetes中部署Pod时，您可以指定这个Pod需要Request及Limit的资源，Kubernetes在部署这个Pod的时候，就会根据Pod的需求找一个具有充足空闲资源的节点部署这个Pod。下面的例子中，声明Nginx这个Pod需要1核CPU，1024M的内存，运行中实际使用不能超过2核CPU和4096M内存。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    resources: # 资源声明
      requests:
        memory: "1024Mi"
        cpu: "1000m"
      limits:
        memory: "4096Mi"
        cpu: "2000m"
```

Kubernetes采用静态资源调度方式，对于每个节点上的剩余资源，它是这样计算的：节点剩余资源=节点总资源-已经分配出去的资源，并不是实际使用的资源。如果您自己手动运行一个很耗资源的程序，Kubernetes并不能感知到。

另外所有Pod上都要声明resources。对于没有声明resources的Pod，它被调度到某个节点后，Kubernetes也不会在对应节点上扣掉这个Pod使用的资源。可能会导致节点上调度过去太多的Pod。

日常运维

- 日志

  创建集群时，请勾选日志服务。

- 监控

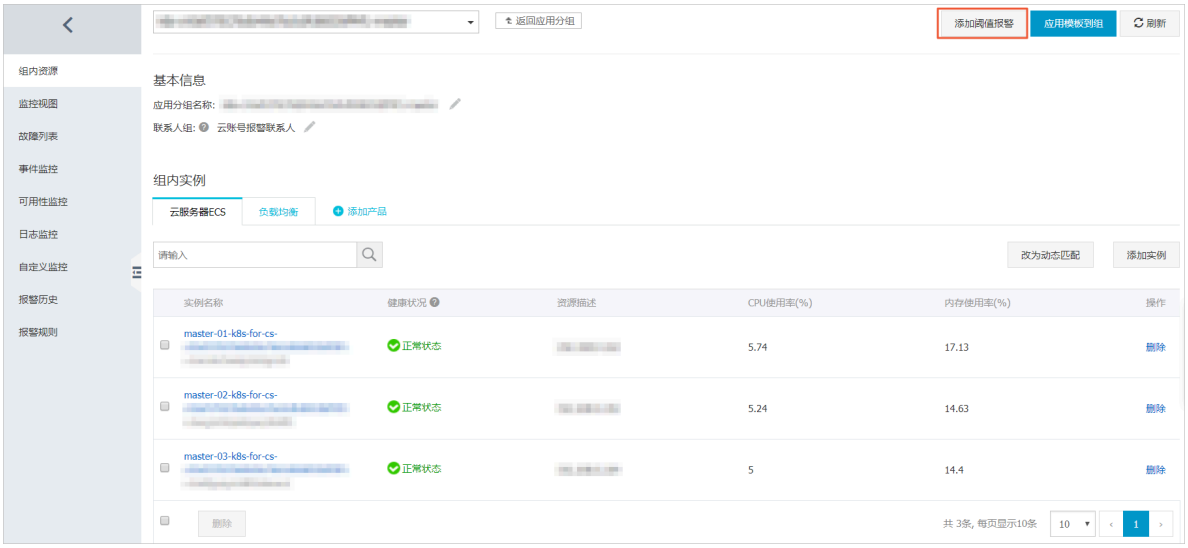
阿里云容器服务与云监控进行集成，通过配置节点监控，提供实时的监控服务。通过添加监控告警规则，节点上的资源使用量很高的时候，可快速定位问题。

通过容器服务创建Kubernetes集群时，会自动在云监控创建两个应用分组：一个对应Master节点，一个对应Worker节点。我们可以在这两个组下面添加一些报警规则，对组里所有的机器生效。后续加入的节点，也会自动出现在组里，不用单独再去配置报警规则。

主要配置ECS资源的报警规则就可以了。

说明:

- 对于ECS的监控，日常运维请设置cpu，memory，磁盘等的报警规则。且尽量将*/var/lib/docker*放在一个独立的盘上。



启动时等待下游服务，不要直接退出

有些应用可能会有一些外部依赖，比如需要从数据库（DB）读取数据或者依赖另外一个服务的接口。应用启动的时候，外部依赖未必都能满足。手工运维的时候，通常采用依赖不满足立即

退出的方式，也就是所谓的failfast，但是在Kubernetes中，这种策略不再适用。原因在于Kubernetes中多数运维操作都是自动的，不需要人工介入，比如部署应用，您不用自己选择节点，再到节点上启动应用，应用fail，也不用手动重启，Kubernetes会自动重启应用。负载增高，还可以通过HPA自动扩容。

针对启动时依赖不满足这个场景，假设有两个应用A和B，A依赖B，刚好运行在同一个节点上。这个节点因为某些原因重启了，重启之后，A首先启动，这个时候B还没启动，对A来说就是依赖不满足。如果A还是按照传统的方式直接退出，当B启动之后，A也不会再启动，必须人工介入处理才行。

Kubernetes的最好的做法是启动时检查依赖，如果不满足，轮询等待，而不是直接退出。可以通过 *Init Container*完成这个功能。

配置restart policy

Pod运行过程中进程退出是个很常见的问题，无论是代码里的一个bug，还是占用内存太多，都会导致应用进程退出，Pod退出。您可在Pod上配置restartPolicy，就能实现Pod挂掉之后自动启动。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    restartPolicy: OnFailure #
```

restartPolicy有三个可选值

- Always：总是自动重启
- OnFailure：异常退出才自动重启（进程退出状态非0）
- Never：永远不重启

配置Liveness Probe和Readiness Probe

Pod处于Running状态和Pod能正常提供服务是完全不同的概念，一个Running状态的Pod，里面的进程可能发生了死锁而无法提供服务。但是因为Pod还是Running的，Kubernetes也不会自动重启这个Pod。所以我们要在所有Pod上配置Liveness Probe，探测Pod是否真的存活，是否还能提供服务。如果Liveness Probe发现了问题，Kubernetes会重启Pod。

Readiness Probe用于探测Pod是不是可以对外提供服务。应用启动过程中需要一些时间完成初始化，在这个过程中是没法对外提供服务的，通过Readiness Probe，可以告诉Ingress或者

Service能不能把流量转发给这个Pod上。当Pod出现问题的时候，Readiness Probe能避免新流量继续转发给这个Pod。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    livenessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
      initialDelaySeconds: 3
      periodSeconds: 3
    readinessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
```

每个进程一个容器

很多刚刚接触容器的人喜欢按照旧习惯把容器当作虚拟机（VM）使用，在一个容器里放多个进程：监控进程、日志进程、sshd进程、甚至整个Systemd。这样操作存在两个问题：

- 判断Pod整体的资源占用会变复杂，不方便实施前面提到resource limit。
- 容器内只有一个进程的情况，进程挂了，外面的容器引擎可以清楚的感知到，然后重启容器。如果容器内有多个进程，某个进程挂了，容器未必受影响，外部的容器引擎感知不到容器内有进程退出，也不会对容器做任何操作，但是实际上容器已经不能正常工作了。

如果有几个进程需要协同工作，在Kubernetes里也可以实现，例如：nginx和php-fpm，通过Unix domain socket通信，我们可以用一个包含两个容器的Pod，unix socket放在两个容器的共享volume中。

确保不存在SPOF（Single Point of Failure）

如果应用只有一个实例，当实例失败的时候，虽然Kubernetes能够重启实例，但是中间不可避免地存在一段时间的不可用。甚至更新应用，发布一个新版本的时候，也会出现这种情况。在Kubernetes里，尽量避免直接使用Pod，尽可能使用Deployment/StatefulSet，并且让应用的Pod在两个以上。

1.2 更新Kubernetes集群已过期的证书

当集群证书已过期时，通过kubectl或api接口调用的方式与集群apiserver的通讯都将被禁止，因此我们无法通过模板部署的方式来自动更新集群各节点上的相应过期证书；此时集群管理员可以登录至集群各节点，通过如下`docker run`启动容器的方式执行目标节点的证书更新任务。

更新Master节点已过期的证书

1. 以root权限登录任意Master节点。
2. 在任意目录下执行以下命令，更新Master节点已经过期的证书。

```
$ docker run -it --privileged=true  -v /:/alicoud-k8s-host --pid
host --net host \
  registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0 /renew/
upgrade-k8s.sh --role master
```

3. 在集群的每个Master节点，重复上述步骤，完成所有Master节点已过期的证书更新。

更新Worker节点已过期的证书

1. 以root权限登录任意Master节点。
2. 执行以下命令，获取集群rootCA私钥。

```
$ cat /etc/kubernetes/pki/ca.key
```

3. 执行以下命令，获取通过base64编码后集群的根私钥。
   - 若获取的集群rootCA私钥，有一行空行，执行以下命令：

     ```
     $ sed '1d' /etc/kubernetes/pki/ca.key| base64 -w 0
     ```

   - 若获取的集群rootCA私钥，无空行，执行以下命令：

     ```
     $ cat /etc/kubernetes/pki/ca.key | base64 -w 0
     ```

4. 以root权限登录任意Worker节点。
5. 在任意目录下执行如下命令，更新Worker节点已经过期的证书。

```
$ docker run -it --privileged=true  -v /:/alicoud-k8s-host --pid
host --net host \
  registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0 /renew/
upgrade-k8s.sh --role node --rootkey ${base64CAKey}
```

说明：

${base64CAKey}为步骤3获取的通过base64编码后的集群根私钥。

6. 在集群的每个Worker节点，重复上述步骤，完成所有Worker节点已过期的证书更新。

1.3 更新Kubernetes集群即将过期的证书

本文介绍如何更新Kubernetes集群即将过期的证书。您可以通过控制台操作，也可以通过命令行自动化一键式更新所有节点证书，也可以手动更新master和worker节点证书。

前提条件

- 您已经成功创建一个Kubernetes集群，参见创建Kubernetes集群。
- 您可以通过kubectl连接集群，参见通过 kubectl 连接 Kubernetes 集群。

控制台更新所有节点证书

在容器服务管理控制台，单击目标集群的更新证书更新即将过期的证书，请参考更新即将过期证书。

命令行自动更新所有节点证书

登录集群任意master节点，执行以下命令完成集群所有节点的证书更新。

```
$ curl http://aliacs-k8s-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/
public/cert-update/renew.sh | bash
```

执行结果

1. 执行以下命令，查看集群master和worker节点状态。

```
$ kubectl get nodes
```

```
[root@                          ~]# kubectl get nodes
NAME                                STATUS    ROLES     AGE       VERSION
cn-hangzhou.                        Ready     <none>    23d       v1.11.2
cn-hangzhou.                        Ready     <none>    23d       v1.11.2
cn-hangzhou.                        Ready     master    47d       v1.11.2
cn-hangzhou.                        Ready     master    47d       v1.11.2
cn-hangzhou.                        Ready     master    47d       v1.11.2
cn-hangzhou.                        Ready     <none>    47d       v1.11.2
cn-hangzhou.                        Ready     <none>    47d       v1.11.2
[root@                          ~]#
```

2. 执行以下命令，当master节点对应的SUCCESSFUL均为1，worker节点对应的SUCCESSFUL为集群worker节点数时，所有证书完成更新。

```
$ kubectl get job -nkube-system
```

```
[root@                    ~]# kubectl get job -nkube-system
NAME                          DESIRED   SUCCESSFUL   AGE
aliyun-cert-renew-master-1    1         1            6m
aliyun-cert-renew-master-2    1         1            5m
aliyun-cert-renew-master-3    1         1            5m
aliyun-cert-renew-worker      4         4            4m
cert-job-2                    1         1            22h
cert-job-3                    1         1            22h
cert-job-4                    1         1            22h
cert-node-2                   4         4            19h
```

手动更新master节点证书

1. 任意路径下，复制以下内容，创建*job-master.yml*文件。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: 1
  parallelism: 1
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                - ${hostname}
      containers:
      - command:
        - /renew/upgrade-k8s.sh
        - --role
        - master
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.
0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
          name: ${jobname}
      hostNetwork: true
      hostPID: true
```

```
      restartPolicy: Never
      schedulerName: default-scheduler
      securityContext: {}
      tolerations:
      - effect: NoSchedule
        key: node-role.kubernetes.io/master
      volumes:
      - hostPath:
          path: /
          type: Directory
```

```
        name: ${jobname}
```

2. 获取集群master节点个数和hostname：

   · 方法一：

   执行以下命令：

   ```
   $ kubectl get nodes
   ```

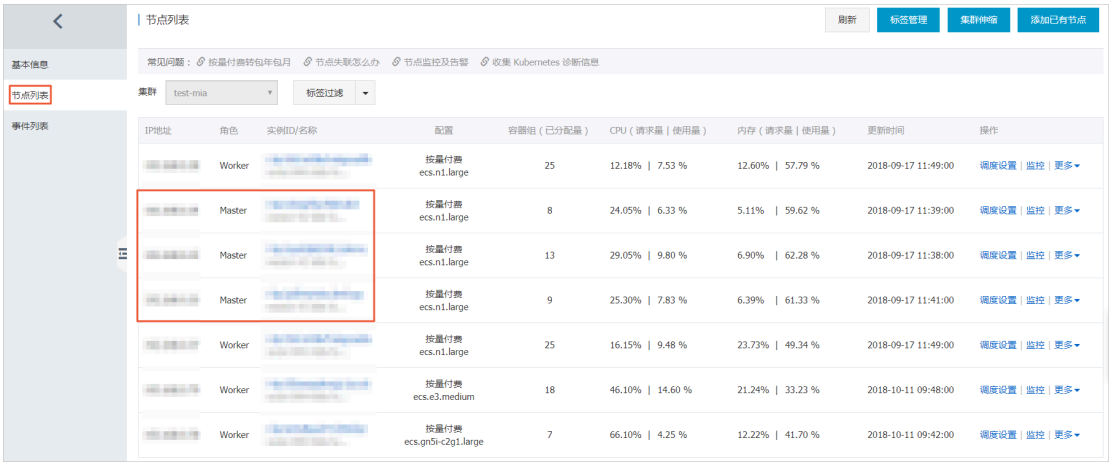
   ```
   [root@                    ~]# kubectl get nodes
   NAME                      STATUS    ROLES     AGE       VERSION
   cn-hangzhou.i             Ready     <none>    22d       v1.11.2
   cn-hangzhou.i             Ready     <none>    22d       v1.11.2
   cn-hangzhou.i             Ready     master    46d       v1.11.2
   cn-hangzhou.i             Ready     master    46d       v1.11.2
   cn-hangzhou.i             Ready     master    46d       v1.11.2
   cn-hangzhou.i             Ready     <none>    46d       v1.11.2
   cn-hangzhou.i             Ready     <none>    46d       v1.11.2
   [root@                    ~]#
   ```

   · 方法二：

   a. 登录容器服务管理控制台。

   b. 在Kubernetes菜单下，单击左侧导航栏的集群，进入集群列表页面。

   c. 选择目标集群，点击集群名称，查看集群详细信息，点击左侧导航栏的节点列表获取master个数和hostname。



3. 执行以下命令替换*job-master.yml*文件中指定的变量${jobname}和${hostname}。

```
$ sed 's/${jobname}/cert-job-2/g; s/${hostname}/hostname/g' job-
master.yml > job-master2.yml
```

其中：

- ${jobname}为Job和Pod的名称，此处设置为cert-job-2。
- ${hostname}为集群master节点的名称，此处请将hostname替换为步骤2中查看到的master名称。

4. 执行以下命令创建Job。

```
$ kubectl create -f job-master2.yml
```

5. 执行以下命令查看Job状态，当SUCCESSFUL均为1时，证书完成更新。

```
$ kubectl get job -nkube-system
```

6. 重复执行步骤3~5，完成所有master节点的证书更新。

```
[root@ ~]# kubectl get job -nkube-system
NAME         DESIRED   SUCCESSFUL   AGE
cert-job-2   1         1            22m
cert-job-3   1         1            2m
cert-job-4   1         1            1m
[root@ ~]#
```

手动更新worker节点证书

1. 任意路径下，复制以下内容，创建*job-node.yml*文件。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: ${nodesize}
  parallelism: ${nodesize}
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: job-name
                operator: In
                values:
                - ${jobname}
            topologyKey: kubernetes.io/hostname
      containers:
      - command:
```

```
        - /renew/upgrade-k8s.sh
        - --role
        - node
        - --rootkey
        - ${key}
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.
0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
          name: ${jobname}
      hostNetwork: true
      hostPID: true
      restartPolicy: Never
      schedulerName: default-scheduler
      securityContext: {}
      volumes:
      - hostPath:
          path: /
          type: Directory
        name: ${jobname}
```

说明:

如果worker节点带有taint，需要在*job-node.yml*文件中增加对该taint的toleration s，即在securityContext: {}与volumes:之间增加以下内容（若有n个带有taint的worker节点，请复制n次）：

```
      tolerations:
      - effect: NoSchedule
        key: ${key}
        operator: Equal
        value: ${value}
```

获取${name}和${value}的方法如下：

a. 任意路径下，复制以下内容，创建*taint.tml*文件。

```
{{printf "%-50s %-12s\n" "Node" "Taint"}}
{{- range .items}}
    {{- if $taint := (index .spec "taints") }}
        {{- .metadata.name }}{{ "\t" }}
        {{- range $taint }}
            {{- .key }}={{ .value }}:{{ .effect }}{{ "\t" }}
        {{- end }}
        {{- "\n" }}
    {{- end}}
```

```
{{- end}}
```

b. 执行以下命令，查询带有taint的worker节点的${name}和${value}。

```
$ kubectl get nodes -o go-template-file="taint.tml"
```

```
[root@ ~]# kubectl get nodes -o go-template-file="taint.tml"
Node                                    Taint
cn-hangzhou.i-                          key1=value1:NoSchedule
cn-hangzhou.i-                          node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-                          node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-                          node-role.kubernetes.io/master=<no value>:NoSchedule
```

2. 执行以下命令，获取集群的CAKey。

```
$ sed '1d' /etc/kubernetes/pki/ca.key | base64 -w 0
```

3. 执行以下命令替换*job-node.yml*文件中指定的变量${jobname}、${nodesize}和${key}。

```
$ sed 's/${jobname}/cert-node-2/g; s/${nodesize}/nodesize/g; s/${key}/key/g' job-node.yml > job-node2.yml
```

其中：

- ${jobname}为Job和Pod的名称，此处设置为cert-node-2。
- ${nodesize}为worker节点个数，获取方法可参考手动更新*master*节点证书的步骤2。此处请将nodesize替换为集群的worker个数。
- ${key}为集群的CAKey，此处请将key替换为手动更新*worker*节点证书步骤3获取到的CAKey。

4. 执行以下命令创建 Job。

```
$ kubectl create -f job-node2.yml
```

5. 执行以下命令查看Job状态，当SUCCESSFUL为集群worker节点数时，证书完成更新。

```
$ kubectl get job -nkube-system
```

```
[root@ ~]# kubectl get job -nkube-system
NAME          DESIRED   SUCCESSFUL   AGE
cert-job-2    1         1            1h
cert-job-3    1         1            47m
cert-job-4    1         1            46m
cert-node-2   4         4            1m
[root@ ~]#
```

1.4 VPC下 Kubernetes 的网络地址段规划

在阿里云上创建 Kubernetes 集群时，通常情况下，可以选择自动创建专有网络，使用默认的网络地址即可。在某些复杂的场景下，需要您自主规划 ECS 地址、Kubernetes Pod 地址和 Service 地址。本文将介绍阿里云 VPC 环境下 Kubernetes 里各种地址的作用，以及地址段该如何规划。

Kubernetes 网段基本概念

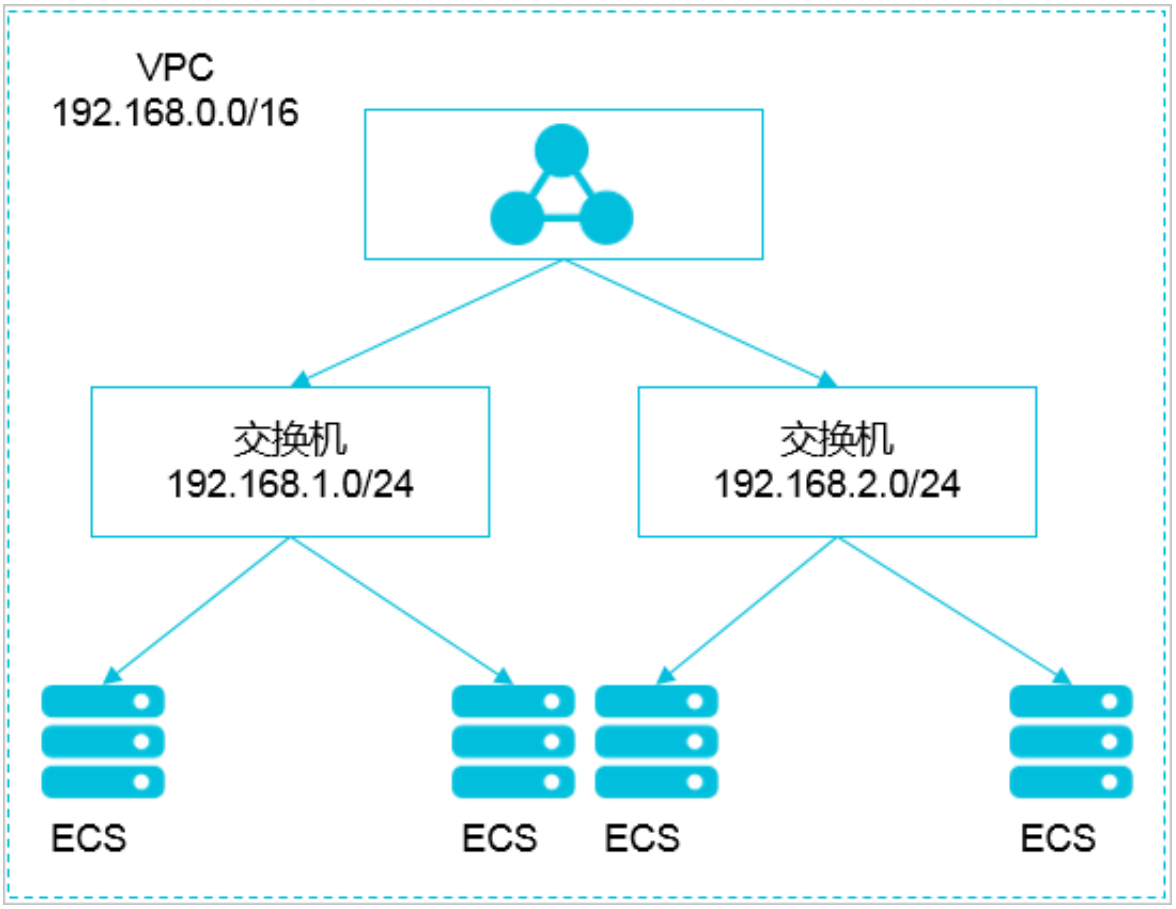
VPC 网段

您在创建 VPC 选择的地址段。只能从 10.0.0.0/8、172.16.0.0/12、192.168.0.0/16 三者当中选择一个。

交换机网段

在 VPC 里创建交换机时指定的网段，必须是当前 VPC 网段的子集（可以跟 VPC 网段地址一样，但不能超过）。交换机下面的 ECS 所分配到的地址，就是从这个交换机地址段内获取的。一个 VPC 下，可以创建多个交换机，但交换机网段不能重叠。

VPC 网段结构如下图。



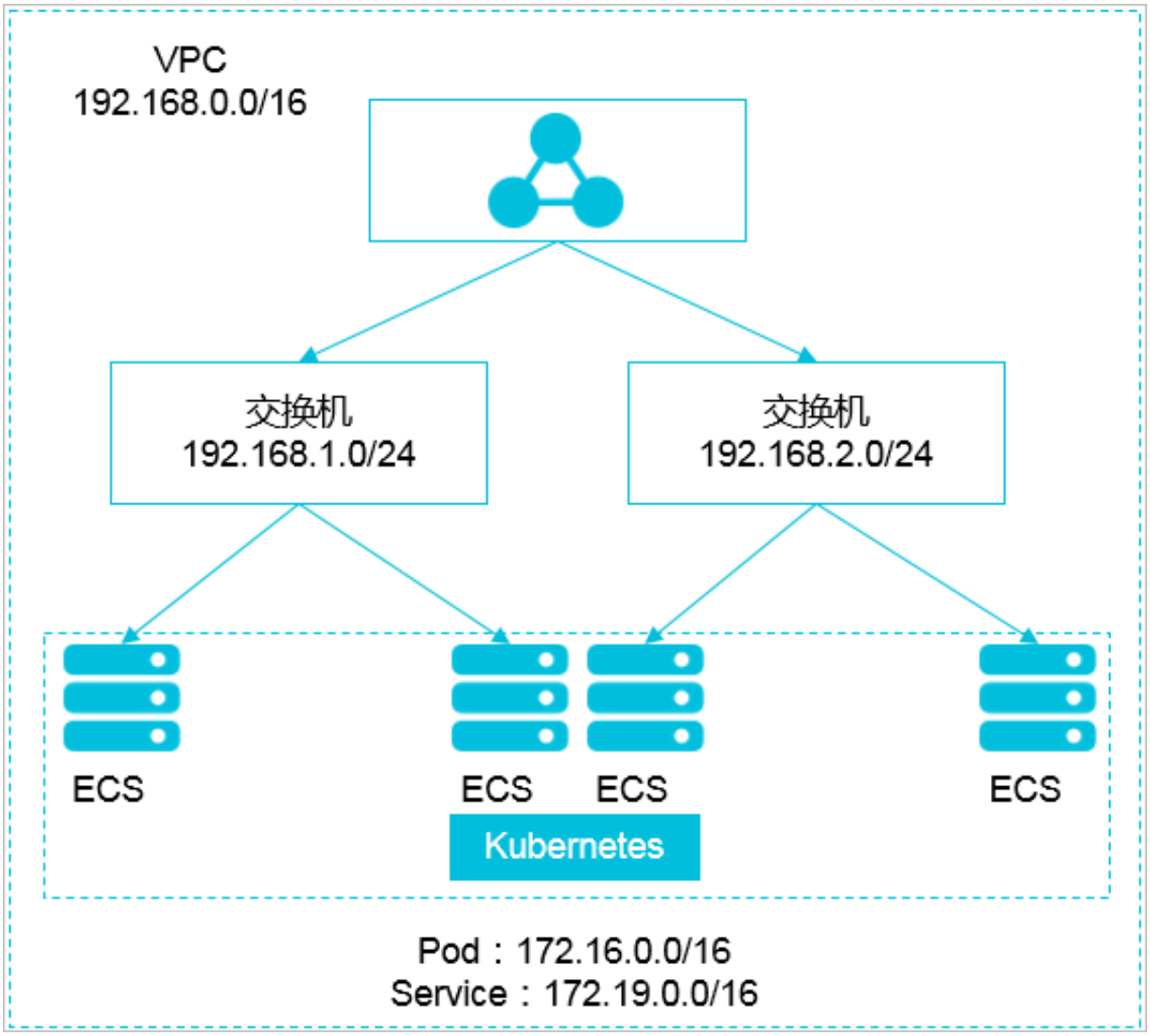
Pod 地址段

Pod 是 Kubernetes 内的概念，每个 Pod 具有一个 IP 地址。在阿里云容器服务上创建 Kubernetes 集群时，可以指定Pod 的地址段，不能和 VPC 网段重叠。比如 VPC 网段用的是 172.16.0.0/12，Kubernetes 的 Pod 网段就不能使用 172.16.0.0/16，172.17.0.0/16等，因为这些地址都涵盖在 172.16.0.0/12 里了。

Service 地址段

Service 也是 Kubernetes 内的概念，每个 Service 有自己的地址。同样，Service 地址段也不能和 VPC 地址段重合，而且 Service 地址段也不能和 Pod 地址段重合。Service 地址只在 Kubernetes 集群内使用，不能在集群外使用。

Kubernetes 网段和 VPC 网段关系如下图。



如何选择地址段

单 VPC+单 Kubernetes 集群场景

这是最简单的情形。VPC 地址在创建 VPC 的时候就已经确定，创建 Kubernetes 集群时，选择和当前 VPC 不一样的地址段就可以了。

单 VPC+多 Kubernetes 集群场景

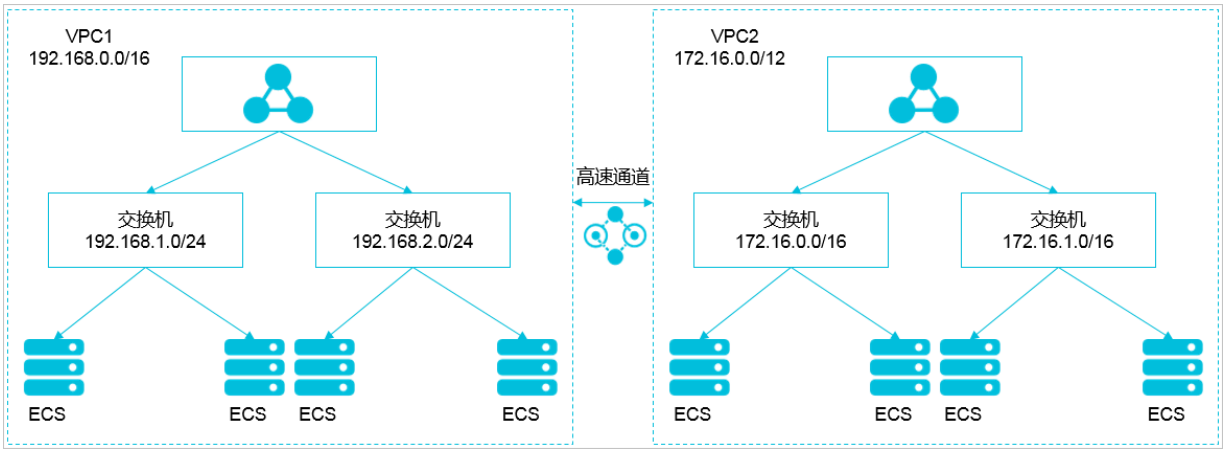
一个 VPC 下创建多个 Kubernetes 集群。在默认的网络模式下（Flannel），Pod 的报文需要通过 VPC 路由转发，容器服务会自动在 VPC 路由上配置到每个 Pod 地址段的路由表。所有 Kubernetes 集群的 Pod 地址段不能重叠，但 Service 地址段可以重叠。

VPC 地址还是创建 VPC 的时候确定的，创建 Kubernetes 的时候，为每个 Kubernetes 集群选择一个不重叠的地址段，不仅不能和 VPC 地址重叠，也不和其他 Kubernetes Pod 段重叠。

需要注意的是，这种情况下 Kubernetes 集群部分互通，一个集群的 Pod 可以直接访问另外一个集群的 Pod 和 ECS，但不能访问另外一个集群的 Service。

VPC 互联场景

两个 VPC 网络互联的情况下，可以通过路由表配置哪些报文要发送到对端 VPC 里。以下面的场景为例，VPC 1 使用地址段 192.168.0.0/16，VPC 2 使用地址段 172.16.0.0/12，我们可以通过路由表，指定在 VPC 1 里把 目的地址为 172.16.0.0/12 的报文都发送到 VPC 2。



在这种情况下，VPC 1 里创建的 Kubernetes 集群，首先不能和 VPC 1 的地址段重叠，同时其地址段也不能和 VPC 2 的地址段重叠。在 VPC 2 上创建 Kubernetes 集群也类似。这个例子中，Kubernetes 集群 Pod 地址段可以选择 10.0.0.0/8 下的某个子段。

说明:

这里要特别关注“路由到 VPC 2” 的地址段，可以把这部分地址理解成已经占用的地址，Kubernetes集群不能和已经占用的地址重叠。

如果 VPC 2 里要访问 VPC 1 的 Kubernetes Pod，则需要在 VPC 2 里配置到 VPC1 Kubernetes集群pod地址的路由。

VPC 网络到 IDC 的场景

和 VPC 互联场景类似，同样存在 VPC 里部分地址段路由到 IDC，Kubernetes 集群的 Pod 地址就不能和这部分地址重叠。IDC 里如果需要访问 Kubernetes 里的 Pod 地址，同样需要在 IDC 端配置到专线 VBR 的路由表。

2 网络

2.1 部署高可靠 Ingress Controller

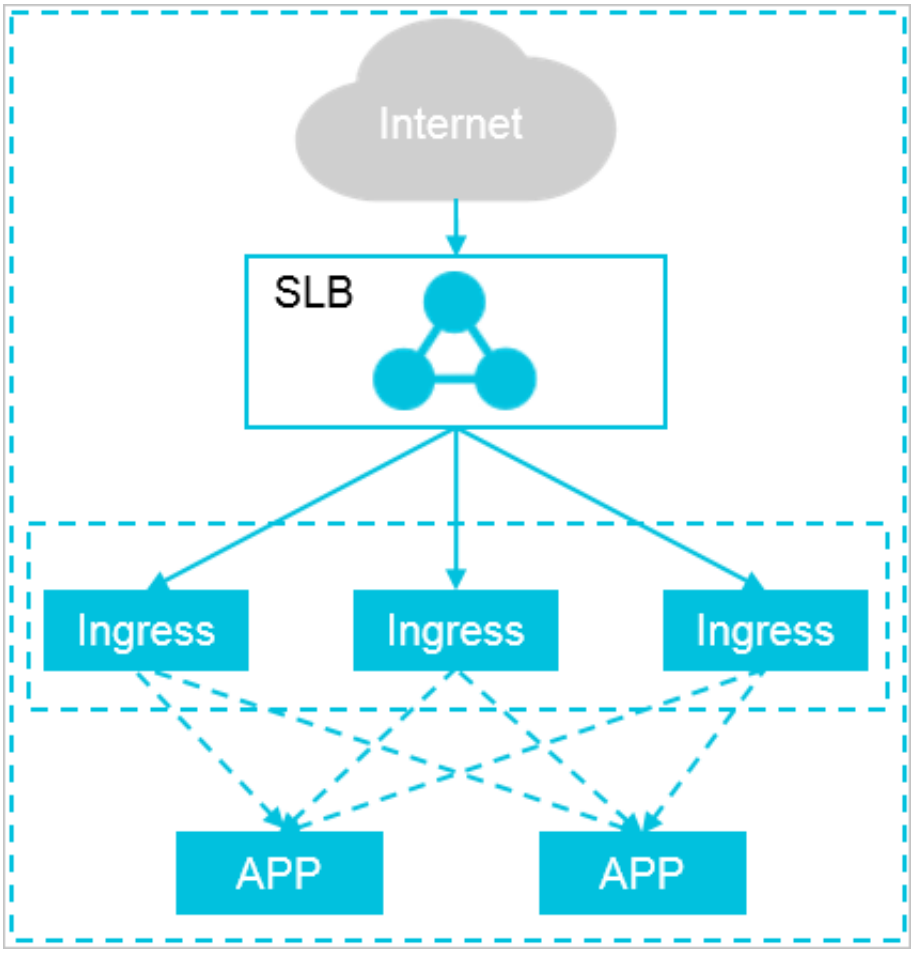
在 Kubernetes 集群中，Ingress 是授权入站连接到达集群服务的规则集合，为您提供七层负载均衡能力，您可以通过 Ingress 配置提供外部可访问的 URL、负载均衡、SSL、基于名称的虚拟主机等。作为集群流量接入层，Ingress 的高可靠性显得尤为重要，本文探讨如何部署一套高性能高可靠的 Ingress 接入层。

前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。
- SSH 连接到 Master 节点，参见SSH访问Kubernetes集群。

高可靠部署架构

高可靠性首先要解决的就是单点故障问题，一般常用的是采用多副本部署的方式，在 Kubernetes 集群中部署高可靠 Ingress 接入层同样采用多节点部署架构，同时由于 Ingress 作为集群流量接入口，建议采用独占 Ingress 节点的方式，以避免业务应用与 Ingress 服务发生资源争抢。



如上述部署架构图，由多个独占 Ingress 实例组成统一接入层承载集群入口流量，同时可依据后端业务流量水平扩缩容 Ingress 节点。当然如果您前期的集群规模并不大，也可以采用将 Ingress 服务与业务应用混部的方式，但建议进行资源限制和隔离。

查看集群默认部署的Pod副本及公网SLB地址

当我们通过成功创建一个集群后，默认情况下，集群内部已经部署了一套拥有2个Pod副本的Nginx Ingress Controller服务，其前端挂载在一个公网SLB实例上。

执行以下命令查看部署Nginx Ingress Controller服务的Pod。

```
$ kubectl -n kube-system get pod | grep nginx-ingress-controller
nginx-ingress-controller-8648ddc696-2bshk                    1/1
Running   0          3h
nginx-ingress-controller-8648ddc696-jvbs9                    1/1
Running   0          3h
```

执行以下命令查看nginx-ingress-lb服务对应的公网SLB地址。

```
$ kubectl -n kube-system get svc nginx-ingress-lb
NAME               TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(
S)                    AGE
```

```
nginx-ingress-lb   LoadBalancer   172.xx.x.xx   118.xxx.xxx.xx   80:
32457/TCP,443:31370/TCP   21d
```

随着集群业务规模的逐渐扩大，我们需要同时扩容Ingress接入层，以保证集群接入层的高性能高可用，可通过以下两种方法完成。

方法一 伸缩Replica数量

我们可以通过简单地调整Nginx Ingress Controller Deployment的Replica数量来快速扩缩容Ingress接入层的规模。

执行以下命令，将Pod副本数量扩容到3个。

```
$ kubectl -n kube-system scale --replicas=3 deployment/nginx-ingress-
controller
deployment.extensions/nginx-ingress-controller scaled
```

执行以下命令，查看部署Nginx Ingress Controller服务的Pod。

```
$ kubectl -n kube-system get pod | grep nginx-ingress-controller
nginx-ingress-controller-8648ddc696-2bshk                    1/1
Running   0          3h
nginx-ingress-controller-8648ddc696-jvbs9                    1/1
Running   0          3h
nginx-ingress-controller-8648ddc696-xqmfn                    1/1
Running   0          33s
```

方法二 在指定节点部署Ingress服务

若希望Nginx Ingress Controller仅运行在指定的一些高配置节点上，可以通过给检点添加标签来完成。

1. 执行以下命令查看当前集群的节点情况。

```
$ kubectl get node
NAME                                 STATUS   ROLES    AGE   VERSION
cn-hangzhou.i-bp11bcmsna8d4bpf17bc   Ready    master   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24lmdc2o   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24lmdc2p   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24lmdc2q   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp181pofzyyksie2ow03   Ready    master   21d   v1.11.5
cn-hangzhou.i-bp1cbsg6rf3580z6uyo7   Ready    master   21d   v1.11.5
```

2. 执行以下命令，给Ingress Node节点`cn-hangzhou.i-bp12h6biv9bg24lmdc2o`和`cn-hangzhou.i-bp12h6biv9bg24lmdc2p`添加标签`node-role.kubernetes.io/ingress="true"`。

说明:

- 添加标签的节点数量要大于等于集群Pod副本数，从而避免多个Pod运行在同一个节点上。

· 不建议将Ingress服务部署到master节点上，尽量选择worker节点添加标签。

```
$ kubectl label nodes cn-hangzhou.i-bp12h6biv9bg24lmdc2o node-role.
kubernetes.io/ingress="true"
node/cn-hangzhou.i-bp12h6biv9bg24lmdc2o labeled
```

```
$ kubectl label nodes cn-hangzhou.i-bp12h6biv9bg24lmdc2p node-role.
kubernetes.io/ingress="true"
node/cn-hangzhou.i-bp12h6biv9bg24lmdc2p labeled
```

3. 执行以下命令，更新deployment，增加nodeSelector配置。

```
$ kubectl -n kube-system patch deployment nginx-ingress-controller
 -p '{"spec": {"template": {"spec": {"nodeSelector": {"node-role.
kubernetes.io/ingress": "true"}}}}}'
deployment.extensions/nginx-ingress-controller patched
```

预期结果

执行以下命令，查看Ingress Pod已部署在添加了标签`node-role.kubernetes.io/ingress="true"`的集群节点上。

```
$ kubectl -n kube-system get pod -o wide | grep nginx-ingress-
controller
nginx-ingress-controller-8648ddc696-2bshk                    1/1
  Running   0          3h    172.16.2.15     cn-hangzhou.i-bp12h6biv9
bg24lmdc2p   <none>
nginx-ingress-controller-8648ddc696-jvbs9                    1/1
  Running   0          3h    172.16.2.145    cn-hangzhou.i-bp12h6biv9
bg24lmdc2o   <none>
```

3 存储

3.1 有状态服务-静态云盘使用最佳实践

本文为您介绍有状态服务-静态云盘的常见使用场景及方法。

背景信息

云盘的使用场景包括：

- 对磁盘I/O要求高的应用，且没有共享数据的需求，如MySQL、Redis等数据存储服务。
- 高速写日志。
- 持久化存储数据，不因Pod生命周期的结束而消失。

静态云盘的使用场景：

已经购买了云盘实例的情况。

静态云盘使用方式：

需手动创建PV及PVC。

前提条件

- 您已成功创建一个Kubernetes集群，参见创建Kubernetes集群。
- 您已成功创建一个云盘，参见创建按量付费云盘。
- 您可以通过kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。

使用限制

- 云盘为阿里云存储团队提供的非共享存储，只能同时被一个Pod挂载。
- 集群中只有与云盘在同一个可用区（Zone）的节点才可以挂载云盘。

创建PV

1. 创建*pv-static.yaml*文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <your-disk-id>
  labels:
    alicloud-pvname: <your-disk-id>
    failure-domain.beta.kubernetes.io/zone: <your-zone>
    failure-domain.beta.kubernetes.io/region: <your-region>
spec:
  capacity:
    storage: 20Gi
```

```
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "alicloud/disk"
    fsType: "ext4"
    options:
      volumeId: "<your-disk-id>"
```

说明:

- `alicloud-pvname: <your-disk-id>`: PV的名称。需要与volumeID云盘ID一致。
- `failure-domain.beta.kubernetes.io/zone: <your-zone>`: 云盘所在的可用区。例如: cn-hangzhou-b。
- `failure-domain.beta.kubernetes.io/region: <your-region>`: 云盘所在的地域。例如: cn-hangzhou。

`failure-domain.beta.kubernetes.io/zone`和`failure-domain.beta.kubernetes.io/region`，在多可用区的情况下，必须配置，可保证您的Pod调度到云盘所在的可用区。

2. 执行以下命令，创建PV。

```
$ kubectl create -f pv-static.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的集群 > 存储卷，进入存储卷列表页面，选择目标集群，可以看到刚刚创建的PV。

创建PVC

创建云盘存储声明PVC，使用`selector`筛选PV，精确配置PVC和PV的绑定关系。

1. 创建*pvc-static.yaml*文件。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
```

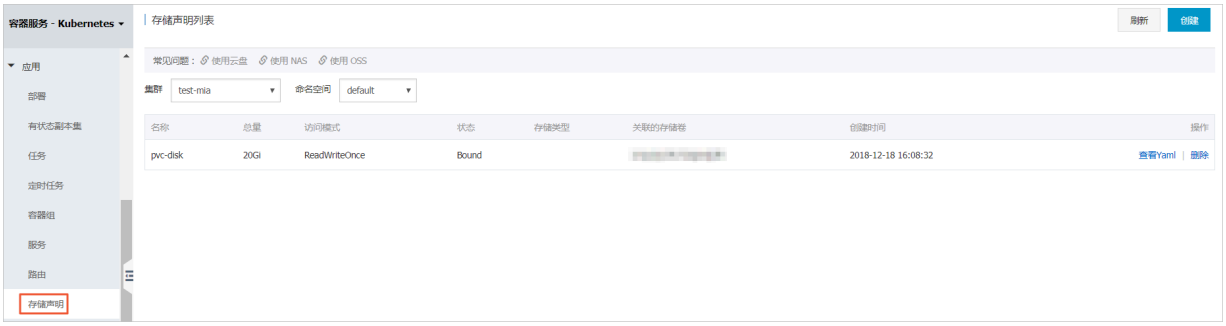
```
      storage: 20Gi
  selector:
    matchLabels:
      alicloud-pvname: <your-disk-id>
```

2. 执行以下命令，创建PVC。

```
$ kubectl create -f pvc-static.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 存储声明，进入存储声明列表页面，选择目标集群和命名空间，可以看到刚刚创建的PVC。



创建应用

1. 创建*static.yaml*文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-static
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - name: disk-pvc
            mountPath: "/data"
      volumes:
        - name: disk-pvc
          persistentVolumeClaim:
```

```
            claimName: pvc-disk
```

2. 执行以下命令，创建Deployment。

```
$ kubectl create -f static.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 部署，进入部署列表页面，选择目标集群和命名空间，可以看到刚刚创建的deployment。

静态云盘的持久化存储

1. 执行以下命令，查看部署的deployment所在Pod的名称。

```
$ kubectl get pod | grep static
nginx-static-78c7dcb9d7-g9lll   2/2     Running     0          32s
```

2. 执行以下命令，查看*/data*路径下是否挂载了新的云盘。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll df | grep data
/dev/vdf        20511312    45080  20449848   1% /data
```

3. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /data
lost+found
```

4. 执行以下命令，在*/data*路径下创建文件*static*。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll touch /data/static
```

5. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /data
static
lost+found
```

6. 执行以下命令，删除名称为`nginx-static-78c7dcb9d7-g9lll`的Pod。

```
$ kubectl delete pod nginx-static-78c7dcb9d7-g9lll
pod "nginx-static-78c7dcb9d7-g9lll" deleted
```

7. 同时在另一个窗口中，执行以下命令，查看Pod删除及Kubernetes重建Pod的过程。

```
$ kubectl get pod -w -l app=nginx
NAME                            READY   STATUS    RESTARTS   AGE
nginx-static-78c7dcb9d7-g9lll   2/2     Running   0          50s
nginx-static-78c7dcb9d7-g9lll   2/2   Terminating   0     72s
nginx-static-78c7dcb9d7-h6brd   0/2   Pending   0     0s
```

```
nginx-static-78c7dcb9d7-h6brd   0/2   Pending   0     0s
nginx-static-78c7dcb9d7-h6brd   0/2   Init:0/1   0     0s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     73s
nginx-static-78c7dcb9d7-h6brd   0/2   Init:0/1   0     5s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     78s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     78s
nginx-static-78c7dcb9d7-h6brd   0/2   PodInitializing   0     6s
nginx-static-78c7dcb9d7-h6brd   2/2   Running   0     8sg   0     8s
```

8. 执行以下命令，查看Kubernetes重建的Pod名称。

```
$ kubectl get pod
NAME                            READY   STATUS      RESTARTS   AGE
nginx-static-78c7dcb9d7-h6brd   2/2     Running     0          14s
```

9. 执行以下命令，查看*/data*路径下的文件，刚刚创建的文件*static*并没有被删除，说明静态云盘的数据可持久保存。

```
$ kubectl exec nginx-static-78c7dcb9d7-h6brd ls /data
static
lost+found
```

3.2 有状态服务-动态云盘使用最佳实践

本文为您介绍有状态服务-动态云盘的常见使用场景及方法。

背景信息

动态云盘的使用场景：

没有购买云盘，在应用部署时自动购买云盘的情况。

动态云盘的使用方式：

1. 手动创建PVC，在PVC中声明StorageClass。
2. 部署应用时通过StorageClass自动创建PV。

前提条件

- 您已成功创建一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您可以通过kubectl连接到Kubernetes集群，参见通过 *kubectl* 连接 *Kubernetes* 集群。
- 集群中需要安装Provisioner插件，该插件可以根据StorageClass自动创建云盘。

Provisioner插件

容器服务Kubernetes版在创建集群时，默认安装Provisioner插件。

创建StorageClass

阿里云容器服务Kubernetes在系统初始化的时候会默认创建4个StorageClass，且使用参数的默认情况。同时这4个StorageClass仅适用于单可用区集群，若是多可用区集群，则需要自己另行创建StorageClass。这4个StorageClass分别为：

- *alicloud-disk-common*：自动创建普通云盘。
- *alicloud-disk-efficiency*：自动创建高效云盘。
- *alicloud-disk-ssd*：自动创建SSD云盘。
- *alicloud-disk-available*：提供高可用选项，先尝试自动创建高效云盘；如果相应可用区的高效云盘资源售尽，再尝试自动创建SSD云盘，如果该可用区的SSD云盘也售尽，则尝试自动创建普通云盘。

1. 创建*storageclass.yaml*文件。

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-ssd-hangzhou-b
provisioner: alicloud/disk
reclaimPolicy: Retain
parameters:
  type: cloud_ssd
  regionid: cn-hangzhou
  zoneid: cn-hangzhou-b
  fstype: "ext4"
  readonly: "false"
```

参数解释：

- `provisioner`：动态云盘配置为 alicloud/disk，标识使用provisioner 插件自动创建阿里云云盘。
- `reclaimPolicy`：云盘的回收策略。支持Delete和Retain，默认情况为Delete。

说明：

如果配置为Delete，删除PVC时，云盘会一起删除，云盘上的数据不可恢复。

- type：自动创建云盘的类型，支持cloud、cloud_efficiency、cloud_ssd、available。
- regionid：（可选）自动创建云盘所在的region。与集群的region相同。
- zoneid：（可选）自动创建云盘所在的zone。
  - 单可用区集群，与集群所在zone相同。
  - 多可用区集群，zoneid可同时配置多个，例如：

```
zoneid: cn-hangzhou-a,cn-hangzhou-b,cn-hangzhou-c
```

- fstype：（可选）自动创建云盘所使用的文件系统，默认情况为ext4。
- readonly：（可选）挂载自动创建云盘的权限是否为可读，支持true：云盘具有只读权限，false：云盘具有可读可写权限，默认情况为false。
- encrypted：（可选）自动创建的云盘是否加密，支持true：加密，false：不加密，默认情况为false。

2. 执行以下命令，创建StorageClass。

```
$ kubectl create -f storageclass.yaml
```

创建PVC

1. 创建*pvc-ssd.yaml*文件。

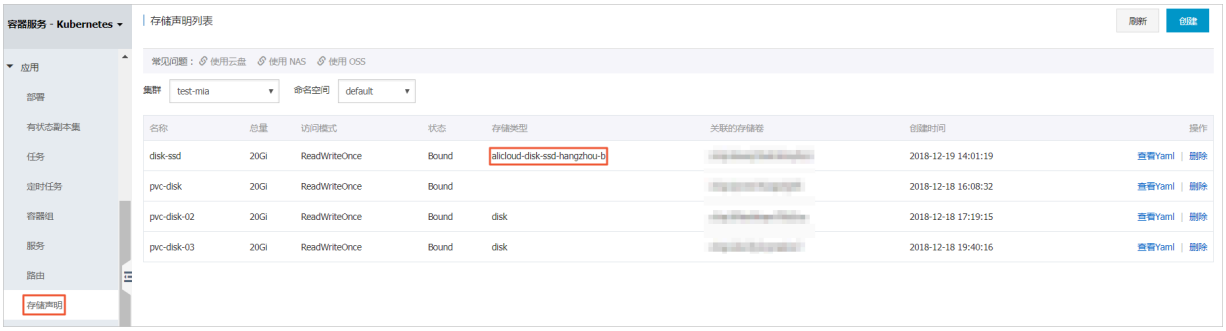
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: disk-ssd
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: alicloud-disk-ssd-hangzhou-b
  resources:
    requests:
      storage: 20Gi
```

2. 执行以下命令，创建PVC。

```
$ kubectl create -f pvc-ssd.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 存储声明，进入存储声明列表页面，选择目标集群和命名空间，可以看到该PVC绑定的存储类型为StorageClass中声明的alicloud-disk-ssd-hangzhou-b，并且关联存储卷。



创建应用

1. 创建*pvc-dynamic.yaml*文件。

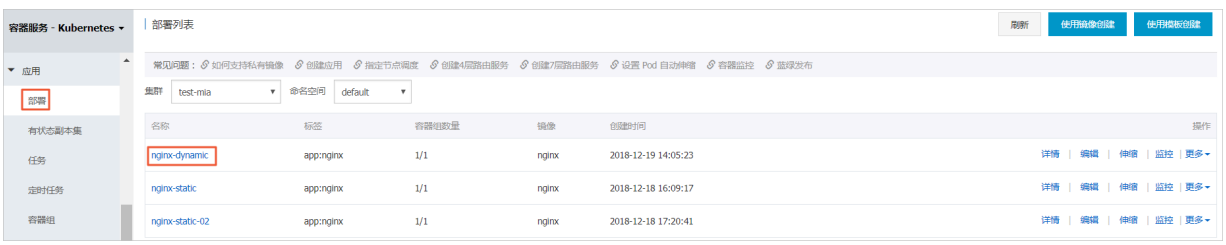
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-dynamic
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - name: disk-pvc
            mountPath: "/data"
      volumes:
        - name: disk-pvc
          persistentVolumeClaim:
            claimName: disk-ssd
```

2. 执行以下命令，创建Deployment。

```
$ kubectl create -f nginx-dynamic.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 部署，进入部署列表页面，选择目标集群和命名空间，可以看到刚刚创建的deployment。



动态云盘的持久化存储

1. 执行以下命令，查看部署的deployment所在Pod的名称。

```
$ kubectl get pod | grep dynamic
nginx-dynamic-5c74594ccb-zl9pf     2/2     Running     0          3m
```

2. 执行以下命令，查看*/data*路径下是否挂载了新的云盘。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf df | grep data
/dev/vdh        20511312    45080  20449848   1% /data
```

3. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /data
lost+found
```

4. 执行以下命令，在*/data*路径下创建文件*dynamic*。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf touch /data/dynamic
```

5. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /data
dynamic
lost+found
```

6. 执行以下命令，删除名称为`nginx-dynamic-5c74594ccb-zl9pf`的Pod。

```
$ kubectl delete pod nginx-dynamic-5c74594ccb-zl9pf
pod "nginx-dynamic-5c74594ccb-zl9pf" deleted
```

7. 同时在另一个窗口中，执行以下命令，查看Pod删除及Kubernetes重建Pod的过程。

```
$ kubectl get pod -w -l app=nginx
NAME                                READY   STATUS    RESTARTS   AGE
nginx-dynamic-5c74594ccb-zl9pf     2/2     Running   0
6m48s
nginx-dynamic-5c74594ccb-zl9pf   2/2   Terminating   0     7m32s
nginx-dynamic-5c74594ccb-45sd4   0/2   Pending   0     0s
nginx-dynamic-5c74594ccb-45sd4   0/2   Pending   0     0s
nginx-dynamic-5c74594ccb-45sd4   0/2   Init:0/1   0     0s
nginx-dynamic-5c74594ccb-zl9pf   0/2   Terminating   0     7m32s
nginx-dynamic-5c74594ccb-zl9pf   0/2   Terminating   0     7m33s
nginx-dynamic-5c74594ccb-zl9pf   0/2   Terminating   0     7m33s
nginx-dynamic-5c74594ccb-45sd4   0/2   PodInitializing   0     5s
nginx-dynamic-5c74594ccb-45sd4   2/2   Running   0     22s
```

8. 执行以下命令，查看Kubernetes重建的Pod名称。

```
$ kubectl get pod
NAME                                READY   STATUS      RESTARTS
AGE
```

```
nginx-dynamic-5c74594ccb-45sd4      2/2       Running      0          2m
```

9. 执行以下命令，查看/data路径下的文件，刚刚创建的文件dynamic并没有被删除，说明动态云盘的数据可持久保存。

```
$ kubectl exec nginx-dynamic-5c74594ccb-45sd4 ls /data
dynamic
lost+found
```

3.3 有状态服务-StatefulSet使用最佳实践

本文为您介绍有状态服务-StatefulSet的常见使用场景及方法。

背景信息

有状态服务-StatefulSet的应用场景：

- 稳定的部署次序：有序部署或扩展，需要根据定义的顺序依次进行（即从0到N-1，在下一个Pod运行之前，所有之前的Pod必须都是Running和Ready状态）。
- 稳定的扩展次序：有序收缩或删除，需要根据定义的顺序依次进行（即从N-1到0，在下一个Pod运行之前，所有之前的Pod必须都是Running和Ready状态）。
- 稳定的网络标志：Pod重新调度后其PodName和HostName不变。
- 稳定的持久化存储：基于PVC，Pod重新调度后仍能访问到相同的持久化数据。

有状态服务-StatefulSet的使用方式：

通过`volumeClaimTemplates`自动创建PVC及PV。

文本将介绍：

- 创建StatefulSet服务
- 伸缩StatefulSet服务
- 删除StatefulSet服务
- StatefulSet服务的持久化存储

前提条件

- 您已成功创建一个Kubernetes集群，参见创建Kubernetes集群。
- 您已连接到Kubernetes集群的Master节点，参见通过 kubectl 连接 Kubernetes 集群。

部署StatefulSet服务

说明：

`volumeClaimTemplates`：表示一类PVC的模板，系统会根据有状态服务-StatefulSet配置的replicas数量，创建相应数量的PVC。这些PVC除了名字不一样之外其他配置都是一样的。

1. 创建*statefulset.yaml*文件。

说明：

`storageClassName`配置为`alicloud-disk-ssd`，表示使用的是阿里SSD类型的云盘。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: disk-ssd
          mountPath: /data
  volumeClaimTemplates:
  - metadata:
      name: disk-ssd
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "alicloud-disk-ssd"
      resources:
        requests:
```

```
            storage: 20Gi
```

2. 执行以下命令，部署StatefulSet服务。

```
$ kubectl create -f statefulset.yaml
```

3. 同时在另一个窗口中，执行以下命令，查看Pod按照次序部署。

```
$ kubectl get pod -w -l app=nginx
NAME          READY   STATUS    RESTARTS   AGE
web-0         0/1     Pending   0          0s
web-0         0/1     Pending   0          0s
web-0         0/1     ContainerCreating    0          0s
web-0         1/1     Running   0          20s
web-1         0/1     Pending   0          0s
web-1         0/1     Pending   0          0s
web-1         0/1     ContainerCreating    0          0s
web-1         1/1     Running   0          7s
```

4. 执行以下命令，查看已部署的Pod。

```
$ kubectl get pod
NAME                           READY   STATUS    RESTARTS   AGE
web-0                          1/1     Running   0          6m
web-1                          1/1     Running   0          6m
```

5. 执行以下命令，查看PVC。

```
$ kubectl get pvc
NAME              STATUS   VOLUME                     CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   7m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   6m
```

伸缩StatefulSet服务

扩容StatefulSet服务

1. 执行以下命令，扩容StatefulSet服务到3个Pod。

```
$ kubectl scale sts web --replicas=3
statefulset.apps/web scaled
```

2. 执行以下命令，查看扩容后的Pod。

```
$ kubectl get pod
NAME                           READY   STATUS    RESTARTS   AGE
web-0                          1/1     Running   0          34m
web-1                          1/1     Running   0          33m
web-2                          1/1     Running   0          26m
```

3. 执行以下命令，查看扩容后的PVC。

```
$ kubectl get pvc
NAME              STATUS   VOLUME                     CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
```

```
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   35m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   34m
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   27m
```

缩容StatefulSet服务

1. **执行以下命令，缩减StatefulSet服务到2个Pod。**

```
$ kubectl scale sts web --replicas=2
statefulset.apps/web scaled
```

2. **执行以下命令，查看缩容后的Pod。Pod的数量已缩减为2。**

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          38m
web-1                         1/1     Running   0          38m
```

3. **执行以下命令，查看缩容后的PVC。PVC/PV并没有随Pod一起缩减。**

```
$ kubectl get pvc
NAME             STATUS   VOLUME                   CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   39m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   39m
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   31m
```

再次扩容StatefulSet服务

1. **执行以下命令，扩容StatefulSet服务到3个Pod。**

```
$ kubectl scale sts web --replicas=3
statefulset.apps/web scaled
```

2. **执行以下命令，查看扩容后的Pod。**

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          1h
web-1                         1/1     Running   0          1h
web-2                         1/1     Running   0          8s
```

3. **执行以下命令，查看扩容后的PVC。扩容后新创建的Pod仍会使用原来的PVC/PV。**

```
$ kubectl get pvc
NAME             STATUS   VOLUME                   CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   1h
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   1h
```

```
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   1h
```

删除StatefulSet服务

1. 执行以下命令，查看名称为web-1的Pod，引用的PVC。

```
$  kubectl describe pod web-1 | grep ClaimName
    ClaimName:  disk-ssd-web-1
```

2. 执行以下命令，删除名称为web-1的Pod。

```
$ kubectl delete pod web-1
pod "web-1" deleted
```

3. 执行以下命令，查看Pod。重新创建的Pod与删除前的Pod名称一致。

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          1h
web-1                         1/1     Running   0          25s
web-2                         1/1     Running   0          9m
```

4. 执行以下命令，查看PVC。重新创建的Pod所使用的PVC与删除前的Pod一致。

```
$ kubectl get pvc
NAME              STATUS   VOLUME                   CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   1h
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   1h
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   1h
```

5. 同时在另一个窗口中，执行以下命令，查看Pod删除和重新创建的过程。

```
$ kubectl get pod -w -l app=nginx
NAME    READY   STATUS    RESTARTS   AGE
web-0   1/1     Running   0          102m
web-1   1/1     Running   0          69s
web-2   1/1     Running   0          10m
web-1   1/1   Terminating   0     89s
web-1   0/1   Terminating   0     89s
web-1   0/1   Terminating   0     90s
web-1   0/1   Terminating   0     90s
web-1   0/1   Pending   0     0s
web-1   0/1   Pending   0     0s
web-1   0/1   ContainerCreating   0     0s
web-1   1/1   Running   0     20s
```

StatefulSet服务的持久化存储

1. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec web-1 ls /data
```

```
lost+found
```

2. 执行以下命令，在*/data*路径下创建文件*statefulset*。

```
$ kubectl exec web-1 touch /data/statefulset
```

3. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec web-1 ls /data
lost+found
statefulset
```

4. 执行以下命令，删除名称为*web-1*的Pod。

```
$ kubectl delete pod web-1
pod "web-1" deleted
```

5. 执行以下命令，查看*/data*路径下的文件，刚刚创建的文件*statefulset*并没有被删除，说明云盘的数据可持久保存。

```
$ kubectl exec web-1 ls /data
lost+found
statefulset
```

3.4 有状态服务-NAS使用最佳实践

本文为您介绍有状态服务-NAS的常见使用场景及方法。

背景信息

NAS支持同时被多个Pod挂载，此时多个Pod可能同时修改相同数据，需要应用自行实现数据的同步。

NAS的使用场景：

- 对磁盘I/O要求较高的应用。
- 读写性能相对于对象存储OSS高。
- 可实现跨主机文件共享，例如：可作为文件服务器。

NAS的使用方式：

1. 手动创建文件系统，并添加挂载点。
2. 手动创建PV及PVC。

本文为您介绍利用阿里云提供的*flexvolume*插件，通过PV/PVC的方式使用阿里云NAS。

前提条件

- 您已成功创建一个Kubernetes集群，参见创建*Kubernetes*集群。

- 您可以通过kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。
- 您已在文件存储控制台创建一个文件系统，参见创建文件系统。创建的文件系统需要与您的Kubernetes集群在同一可用区。
- 您已在创建好的文件系统中添加容器服务Kubernetes集群的挂载点，参见添加挂载点。文件系统挂载时的VPC网络要与您Kubernetes集群所在的VPC网络保持一致。

创建PV

1. 创建*pv-nas.yaml*文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nas
  labels:
    alicloud-pvname: pv-nas
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  flexVolume:
    driver: "alicloud/nas"
    options:
      server: "***-**.cn-hangzhou.nas.aliyuncs.com"    ////请替换为您实际的挂载点
      path: "/k8s1"
```

```
      vers: "4.0"
```

参数解释

- `alicloud-pvname`：PV的名称。
- `server`：NAS的挂载点。可在文件存储控制台，单击左侧导航栏的文件系统列表，选择目标文件系统，单击操作列管理，在挂载点区域，挂载地址即为NAS数据盘的挂载点。

- `path`：NAS的挂载目录，支持挂载NAS的子目录，当子目录不存在时，会在NAS上自动创建子目录并挂载。
- `vers`：（可选）NFS挂载协议的版本号，支持3和 4.0，默认情况是4.0。
- `mode`：（可选）挂载目录的访问权限，默认情况是不配置。

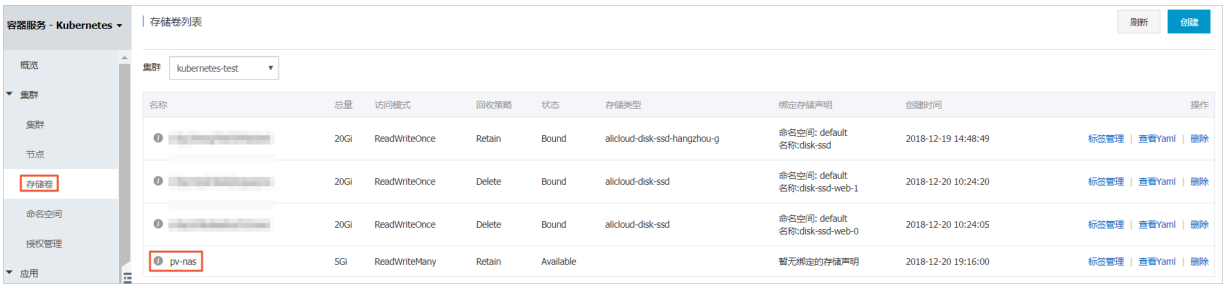
> **说明:**
> - 挂载 NAS的根目录不能配置访问权限。
> - 当 NAS中数据量很大时，配置`mode`会导致挂载非常慢，甚至挂载失败，不建议配置。

2. 执行以下命令，创建PV。

```
$ kubectl create -f pv-nas.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的集群 > 存储卷，进入存储卷列表页面，选择目标集群，可以看到刚刚创建的PV。



创建PVC

创建NAS存储声明PVC，使用`selector`筛选PV，精确配置PVC和PV的绑定关系。

1. 创建*pvc-nas.yaml*文件。

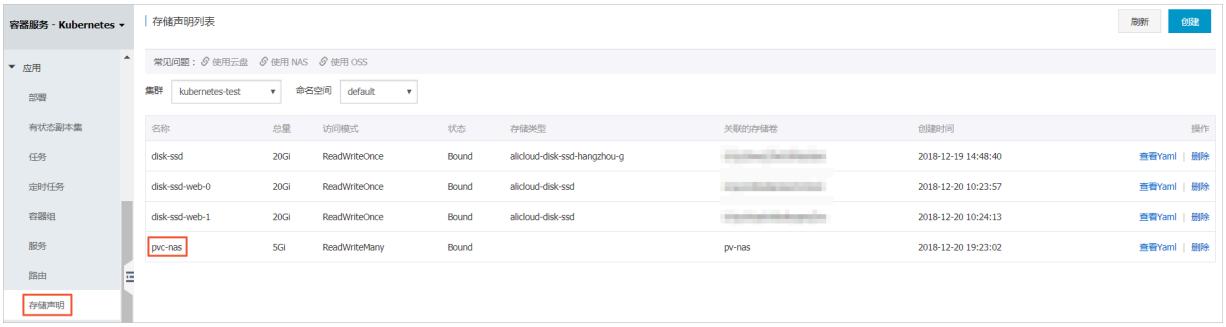
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-nas
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-nas
```

2. 执行以下命令，创建PVC。

```
$ kubectl create -f pvc-nas.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 存储声明，进入存储声明列表页面，选择目标集群和命名空间，可以看到刚刚创建的PVC。



创建应用

1. 创建*nas.yaml*文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nas-static
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
```

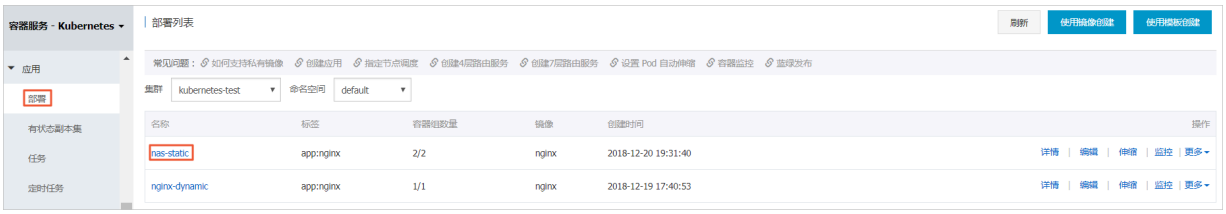
```
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-nas
            mountPath: "/data"
      volumes:
        - name: pvc-nas
          persistentVolumeClaim:
            claimName: pvc-nas
```

2. 执行以下命令，创建Deployment。

```
$ kubectl create -f nas.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 部署，进入部署列表页面，选择目标集群和命名空间，可以看到刚刚创建的deployment。



NAS的共享存储

1. 执行以下命令，查看部署的deployment所在Pod的名称。

```
$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
nas-static-f96b6b5d7-rcb2f          1/1     Running   0          9m
nas-static-f96b6b5d7-wthmb          1/1     Running   0          9m
```

2. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f ls /data
```

```
$ kubectl exec nas-static-f96b6b5d7-wthmb ls /data
```

说明:

*/data*路径下为空，没有文件。

3. 执行以下命令，在任意一个Pod的*/data*路径下创建文件*nas*。

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f touch /data/nas
```

4. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f ls /data
nas
```

```
$ kubectl exec nas-static-f96b6b5d7-wthmb ls /data
nas
```

说明:

在任意一个Pod的*/data*下创建的文件，两个Pod下的*/data*路径下均存在此文件，说明两个Pod共享一个NAS。

NAS的持久化存储

1. 执行以下命令，删除该应用的所有Pod。

```
$ kubectl delete pod nas-static-f96b6b5d7-rcb2f nas-static-f96b6b5d7
-wthmb
pod "nas-static-f96b6b5d7-rcb2f" deleted
pod "nas-static-f96b6b5d7-wthmb" deleted
```

2. 同时在另一个窗口中，执行以下命令，查看Pod删除及Kubernetes重建Pod的过程。

```
$ kubectl get pod -w -l app=nginx
NAME                              READY   STATUS    RESTARTS   AGE
nas-static-f96b6b5d7-rcb2f        1/1     Running   0          27m
nas-static-f96b6b5d7-wthmb        1/1     Running   0          27m
nas-static-f96b6b5d7-rcb2f   1/1   Terminating   0     28m
nas-static-f96b6b5d7-wnqdj   0/1   Pending   0     0s
nas-static-f96b6b5d7-wnqdj   0/1   Pending   0     0s
nas-static-f96b6b5d7-wnqdj   0/1   ContainerCreating   0     0s
nas-static-f96b6b5d7-wthmb   1/1   Terminating   0     28m
nas-static-f96b6b5d7-nwkds   0/1   Pending   0     0s
nas-static-f96b6b5d7-nwkds   0/1   Pending   0     0s
nas-static-f96b6b5d7-nwkds   0/1   ContainerCreating   0     0s
nas-static-f96b6b5d7-rcb2f   0/1   Terminating   0     28m
nas-static-f96b6b5d7-wthmb   0/1   Terminating   0     28m
nas-static-f96b6b5d7-rcb2f   0/1   Terminating   0     28m
nas-static-f96b6b5d7-rcb2f   0/1   Terminating   0     28m
nas-static-f96b6b5d7-wnqdj   1/1   Running   0     10s
nas-static-f96b6b5d7-wthmb   0/1   Terminating   0     28m
nas-static-f96b6b5d7-wthmb   0/1   Terminating   0     28m
nas-static-f96b6b5d7-nwkds   1/1   Running   0     17s
```

3. 执行以下命令，查看Kubernetes重建的Pod名称。

```
$ kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
nas-static-f96b6b5d7-nwkds        1/1     Running   0          21s
```

```
nas-static-f96b6b5d7-wnqdj        1/1     Running   0          21s
```

4. 执行以下命令，查看/data路径下的文件。

```
$ kubectl exec nas-static-f96b6b5d7-nwkds ls /data
nas
```

```
$ kubectl exec nas-static-f96b6b5d7-wnqdj ls /data
nas
```

说明:

刚刚创建的文件nas并没有被删除，说明NAS的数据可持久保存。

3.5 有状态服务-OSS存储使用最佳实践

本文为您介绍有状态服务-OSS的常见使用场景及方法。

背景信息

阿里云对象存储服务（OSS）提供海量、安全、低成本、高可靠的云存储服务。OSS支持同时被多个Pod挂载。

OSS使用场景：

- 对磁盘I/O要求低。
- 配置文件、图片、小视频等共享业务。

OSS使用方式：

1. 手动创建Bucket。
2. 获取AccessKey ID和AccessKey Secret。
3. 手动创建PV及PVC。

前提条件

- 您已成功创建一个Kubernetes集群，参见创建Kubernetes集群。
- 您可以通过kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。
- 您已在OSS管理控制台创建Bucket，参见#unique_28。

注意事项

- 容器服务Kubernetes集群升级会重启kubelet，OSSFS驱动跟随一起重启，导致OSS目录不可用。这时使用OSS的Pod需要重建，可在yaml文件中增加健康检查的配置，在容器内OSS目录不可用时自动重启Pod，重新挂载OSS。
- 通过最新版本挂载的OSS已经解决了此问题。

创建PV

1. 创建*pv-oss.yaml*文件。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-oss
  labels:
    alicloud-pvname: pv-oss
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  storageClassName: oss
  flexVolume:
    driver: "alicloud/oss"
    options:
      bucket: "docker"                          ////请替换为您的Bucket名
称
      url: "oss-cn-hangzhou.aliyuncs.com"       ////请替换为您的URL
      akId: "***"                               ////请替换为您的akId
      akSecret: "***"                           ////请替换为您的akSecret
      otherOpts: "-o max_stat_cache_size=0 -o allow_other"   ////请替
换为您的otherOpts
```

参数解释

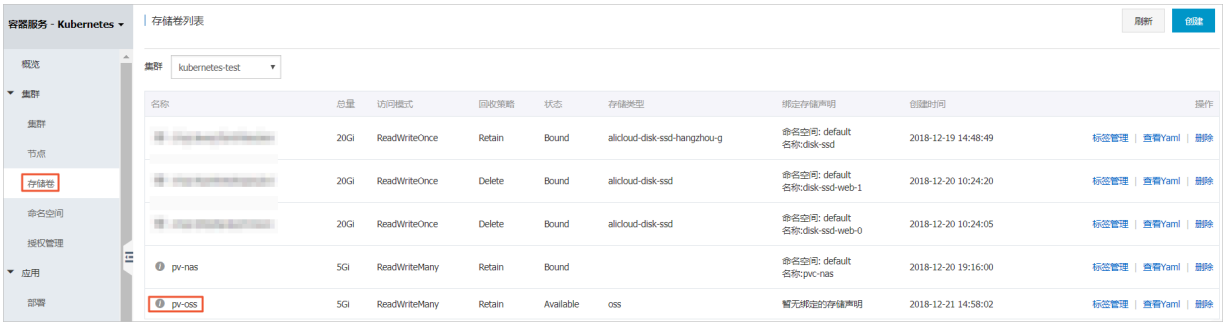
- `alicloud-pvname`：PV的名称，与PVC中的`selector`配合使用。
- `bucket`：Bucket名称，目前只支持挂载 Bucket，不支持挂载 Bucket 下面的子目录或文件。
- `url`：OSS Bucket的访问域名（Endpoint），可参见访问域名和数据中心。也可以在OSS管理控制台查询：登录OSS 管理控制台，在左侧导航栏选择目标Bucket，在访问域名区域，查看Endpoint（地域节点）。
- `akId`：用户的AccessKey ID。在容器服务管理控制台，单击右上角  ，主账号选择accesskeys，子账号选择AccessKey管理，设置AccessKey ID和AccessKey Secret。
- `akSecret`：用户的AccessKey Secret。获取方法同`akId`。
- `otherOpts`：挂载OSS时支持定制化参数，格式为：`-o *** -o ***`，可参见*FAQ*。

2. 执行以下命令，创建PV。

```
$ kubectl create -f pv-oss.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的集群 > 存储卷，进入存储卷列表页面，选择目标集群，可以看到刚刚创建的PV。



创建PVC

创建OSS存储声明PVC，使用`selector`筛选PV，精确配置PVC和PV的绑定关系。使用`storageClassName`，表示PVC只与OSS类型的PV绑定。

1. 创建*pvc-oss.yaml*文件。

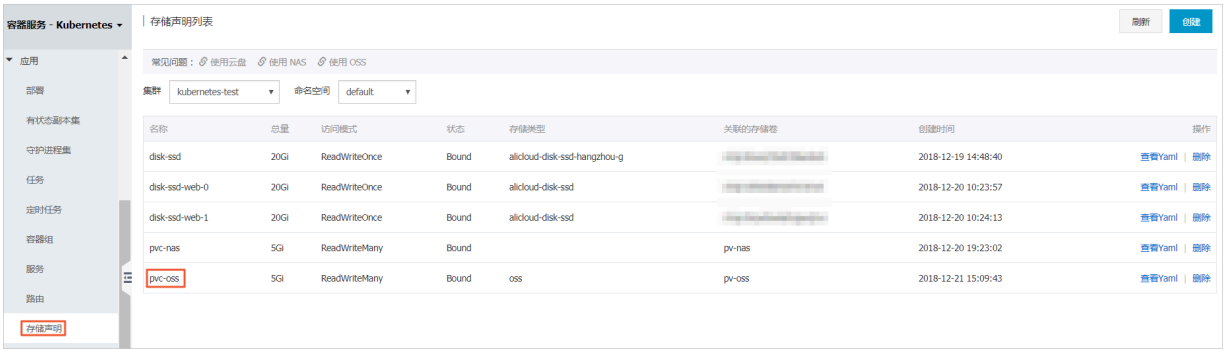
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-oss
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: oss
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-oss
```

2. 执行以下命令，创建PVC。

```
$ kubectl create -f pvc-oss.yaml
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 存储声明，进入存储声明列表页面，选择目标集群和命名空间，可以看到刚刚创建的PVC。



创建应用

1. 创建*oss-static.yaml*文件。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oss-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-oss
            mountPath: "/data"
          - name: pvc-oss
            mountPath: "/data1"
        livenessProbe:
          exec:
            command:
            - sh
            - -c
            - cd /data
          initialDelaySeconds: 30
          periodSeconds: 30
      volumes:
        - name: pvc-oss
          persistentVolumeClaim:
            claimName: pvc-oss
```

说明:

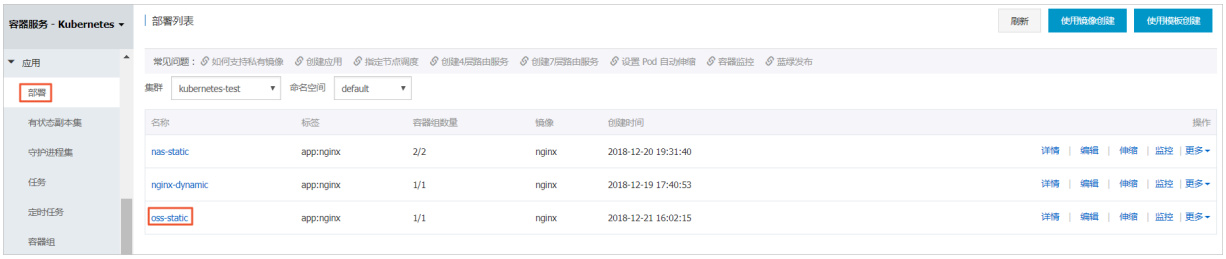
健康检查`livenessProbe`的详细内容及解释，请参见使用阿里云 *OSS*。

2. 执行以下命令，创建Deployment。

```
$ kubectl create -f oss-static.yaml d
```

预期结果

在 Kubernetes 菜单下，单击左侧导航栏的应用 > 部署，进入部署列表页面，选择目标集群和命名空间，可以看到刚刚创建的deployment。



OSS的持久化存储

1. 执行以下命令，查看部署的deployment所在Pod的名称。

```
$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
oss-static-66fbb85b67-dqbl2         1/1     Running   0          1h
```

2. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 ls /data | grep tmpfile
```

> **说明：**
>
> */data*路径下为空，没有文件。

3. 执行以下命令，在*/data*路径下创建文件*tmpfile*。

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 touch /data/tmpfile
```

4. 执行以下命令，查看*/data*路径下的文件。

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 ls /data | grep tmpfile
tmpfile
```

5. 执行以下命令，删除名称为*oss-static-66fbb85b67-dqbl2*的Pod。

```
$ kubectl delete pod oss-static-66fbb85b67-dqbl2
pod "oss-static-66fbb85b67-dqbl2" deleted
```

6. 同时在另一个窗口中，执行以下命令，查看Pod删除及Kubernetes重建Pod的过程。

```
$ kubectl get pod -w -l app=nginx
NAME                                READY   STATUS    RESTARTS   AGE
oss-static-66fbb85b67-dqbl2         1/1     Running   0          78m
oss-static-66fbb85b67-dqbl2   1/1   Terminating   0     78m
oss-static-66fbb85b67-zlvmw   0/1   Pending   0     <invalid>
oss-static-66fbb85b67-zlvmw   0/1   Pending   0     <invalid>
oss-static-66fbb85b67-zlvmw   0/1   ContainerCreating   0     <
invalid>
oss-static-66fbb85b67-dqbl2   0/1   Terminating   0     78m
oss-static-66fbb85b67-dqbl2   0/1   Terminating   0     78m
oss-static-66fbb85b67-dqbl2   0/1   Terminating   0     78m
```

```
oss-static-66fbb85b67-zlvmw   1/1   Running   0     <invalid>
```

7. **执行以下命令，查看Kubernetes重建的Pod名称。**

```
$ kubectl get pod
NAME                             READY   STATUS    RESTARTS   AGE
oss-static-66fbb85b67-zlvmw      1/1     Running   0          40s
```

8. **执行以下命令，查看*/data*路径下的文件，刚刚创建的文件*tmpfile*并没有被删除，说明OSS上的数据可持久保存。**

```
$ kubectl exec oss-static-66fbb85b67-zlvmw ls /data | grep tmpfile
tmpfile
```

4 发布

4.1 Kubernetes集群中使用阿里云 SLB 实现四层金丝雀发布

在 Kubernetes 集群中，对于通过 tcp/udp 访问的服务来说，七层的 ingress 不能很好地实现灰度发布的需求。这里我们就来介绍一下如何使用 SLB 来进行四层的金丝雀发布。

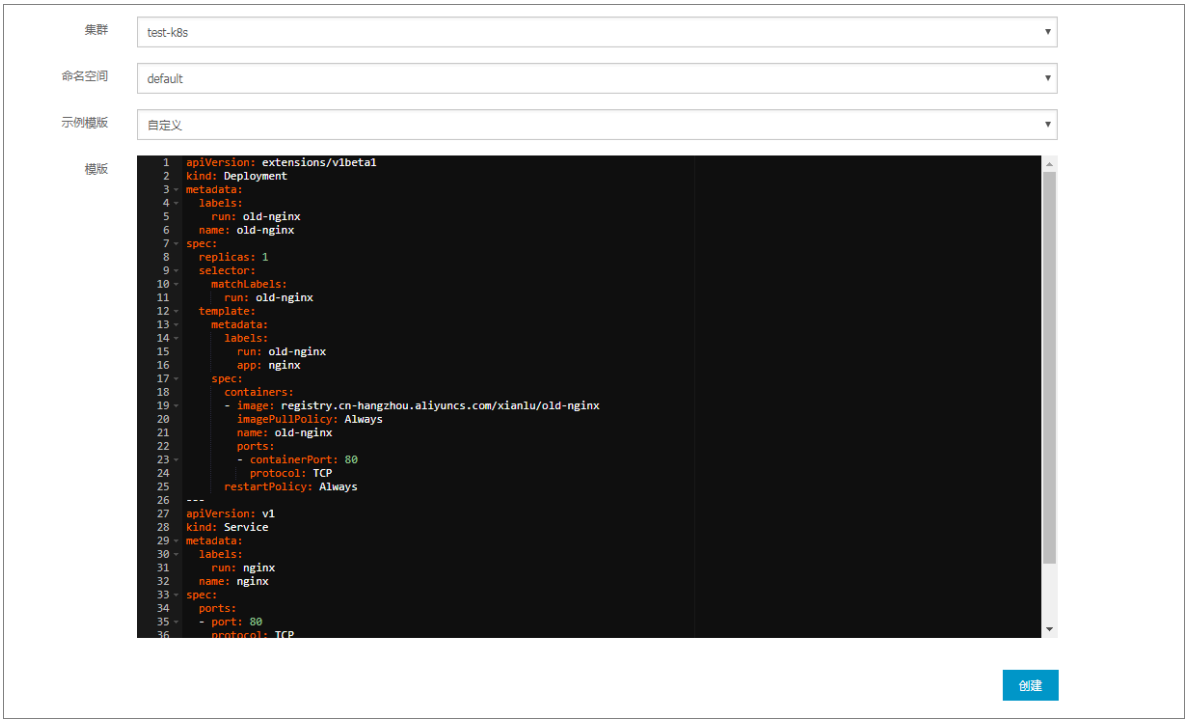
前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。
- SSH 连接到 Master 节点，参见SSH 访问 Kubernetes 集群。

步骤1 部署旧版本服务

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。

4. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击创建。

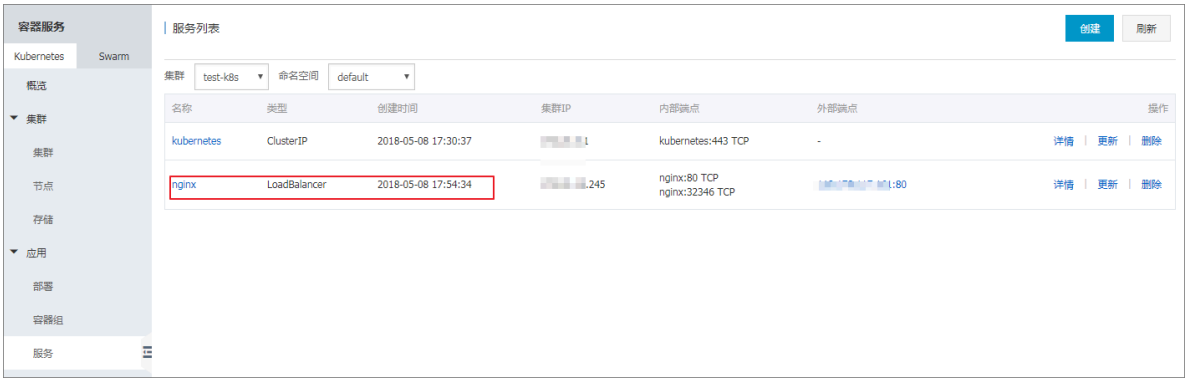


本例是一个 nginx 的编排，通过 SLB 对外暴露的服务。

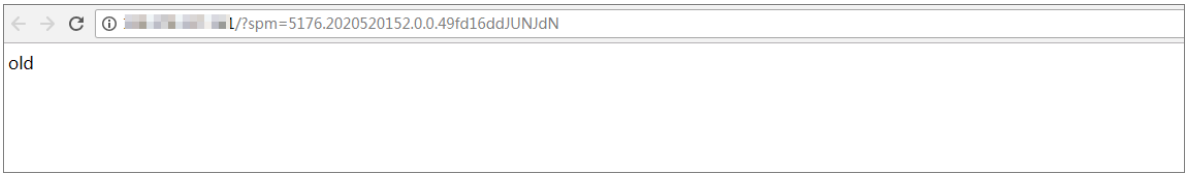
```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   labels:
     run: old-nginx
   name: old-nginx
 spec:
   replicas: 1
   selector:
     matchLabels:
       run: old-nginx
   template:
     metadata:
       labels:
         run: old-nginx
         app: nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
         imagePullPolicy: Always
         name: old-nginx
         ports:
         - containerPort: 80
           protocol: TCP
       restartPolicy: Always
 ---
 apiVersion: v1
 kind: Service
 metadata:
   labels:
     run: nginx
   name: nginx
 spec:
   ports:
```

```
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: LoadBalancer                                    ##通过阿里云SLB对外暴露服务
```

5. 单击左侧导航栏中的应用 > 部署，以及应用 > 服务，进行查看。



6. 您可单击服务右侧的外部端点，进入 nginx 默认欢迎页面，本示例中的 nginx 欢迎页面会显示 old，表示当前访问的服务对应后端 old-nginx 容器。



为了方便展示多次发布的情况，建议登录 Master 节点，后面都将通过 `curl`命令查看部署的效果。

```
# bash
# for x in {1..10} ; do curl EXTERNAL-IP; done                    ##
EXTERNAL-IP即是服务的外部端点
old
old
old
old
old
old
old
old
old
```

```
old
```

步骤2 上线新版本 deployment

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。

4. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击创建。

   本示例是一个 nginx 的新版本的 deployment，其中的 label，也是含有 `app:nginx` 标签。这个标签就是为了使用与旧版本 deployment 相同的 nginx 服务，这样就可以将对应的流量导入进来。

   本示例的编排模板如下。

```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   labels:
     run: new-nginx
   name: new-nginx
 spec:
   replicas: 1
   selector:
     matchLabels:
       run: new-nginx
   template:
     metadata:
       labels:
         run: new-nginx
         app: nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
         imagePullPolicy: Always
         name: new-nginx
         ports:
         - containerPort: 80
           protocol: TCP
```

```
        restartPolicy: Always
```

5. 单击左侧导航栏中的部署，进入部署列表，您可看到名为 new-nginx 的 deployment。

6. 登录 Master 节点，执行 curl 命名，查看服务访问的情况。

```
# bash
# for x in {1..10} ; do curl EXTERNAL-IP; done                    ##
EXTERNAL-IP即是服务的外部端点
new
new
new
old
new
old
new
new
old
old
```

可看到五次访问到旧的服务，五次访问到新的服务。主要原因是 service 对于流量请求是平均的负载均衡策略，而且新老 deployment 均为一个 pod，因此他们的流量百分比为 1：1 。

步骤3 调整流量权重

基于 SLB 的金丝雀发布需要通过调整后端的 pod 数量来调整对应的权重。比如我们这里希望新的服务权重更大一些，假如将新的 pod 数量调整到 4 个。

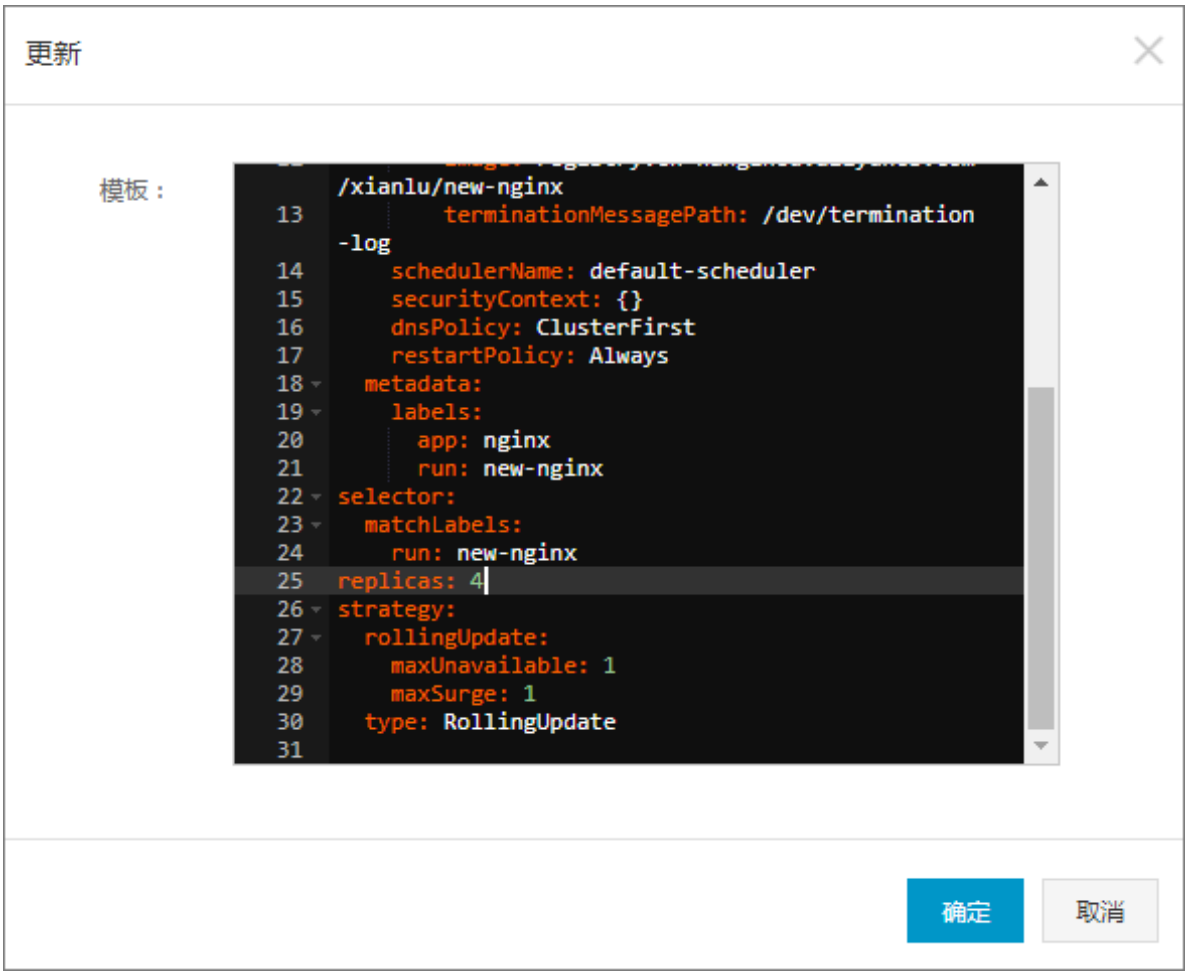
> **说明:**
> 新旧版本应用同时存在时，某个样本的 curl 命令返回的结果并非严格按照设置的权重，本例中执行 10 次 curl 命令，您可通过观察更大的样本获取接近的效果。

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。

3. 选择所需的集群和命名空间，选择所需的 deployment，单击右侧的更新。

4. 在弹出的对抗框中，设置 pod 的数量，将其调整为 4。

> **说明:**
> Kubernetes 的 Deployment 资源默认的更新方式就是 rollingUpdate，所以在更新过程中，会保证最小可服务的容器个数，该个数也可以在模板里面调整。



5. 待部署完成后，登录 Master 节点，执行 curl 命名，查看效果。

```
# bash
 # for x in {1..10} ; do curl EXTERNAL-IP; done
 ##EXTERNAL-IP即是服务的外部端点
 new
 new
```

```
new
new
new
old
new
new
new
old
```

可以看到，10 个请求里面，有 8 个请求到新的服务，2 个到老的服务。

您可通过动态地调整 pod 的数量来调整新老服务的权重，实现金丝雀发布。

4.2 应用

4.2.1 利用 Kubernetes+GPU 方案构建 Tensorflow 应用

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

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

前提准备

- 登录 容器服务管理控制台、*ROS* 管理控制台 和 *RAM* 管理控制台 开通相应的服务。
- 所创建的资源均为按量付费，根据阿里云的计费要求，请确保您的现金账户余额不少于 100 元。
- 目前，按量付费的异构计算 GN5 需要申请工单开通，请登录阿里云账号后，按照如下内容提交 *ECS* 工单。

  我需要申请按量付费的GPU计算型gn5，请帮忙开通，谢谢。

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

使用限制

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

集群部署

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

1. 选择 ROS 创建入口。

   - 创建一个位于华北2的GPU Kubernetes集群
   - 创建一个位于华东2的GPU Kubernetes集群
   - 创建一个位于华南1的GPU Kubernetes集群

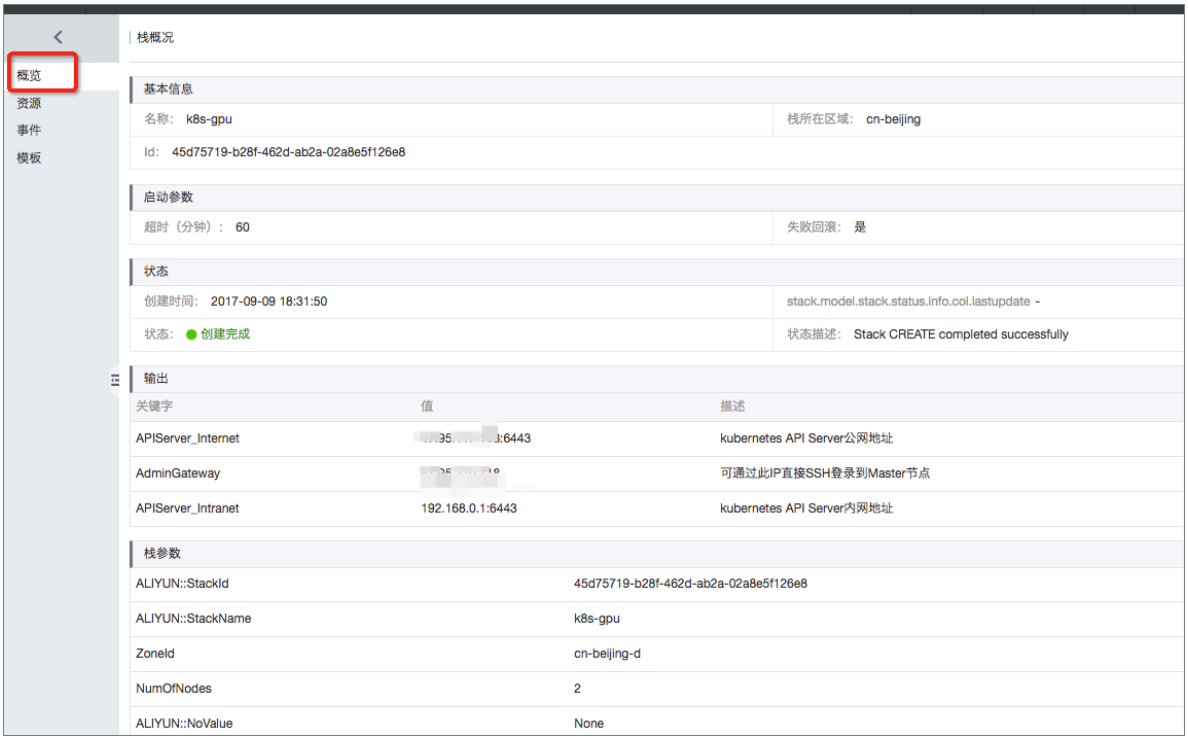
2. 填写参数并单击创建。

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

3. 单击创建， 启动部署。

这样，部署请求已经成功提交。 可以单击进入事件列表实时监控部署过程。

4. 单击概览查看，部署完成后的输出结果。



通过输出结果中返回的信息，可以对 Kubernetes 集群进行管理：

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

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

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

```
  ...
```

部署GPU应用

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

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: jupyter
spec:
  replicas: 1
  template:
    metadata:
      labels:
        k8s-app: jupyter
    spec:
      containers:
      - name: jupyter
        image: registry-vpc.cn-beijing.aliyuncs.com/tensorflow-samples
/jupyter:1.1.0-devel-gpu
        imagePullPolicy: IfNotPresent
        env:
          - name: PASSWORD
            value: mypassw0rd
        resources:
          limits:
            alpha.kubernetes.io/nvidia-gpu: 1
        volumeMounts:
        - mountPath: /usr/local/nvidia
          name: nvidia
      volumes:
        - hostPath:
            path: /var/lib/nvidia-docker/volumes/nvidia_driver/375.39
          name: nvidia
---
apiVersion: v1
kind: Service
metadata:
  name: jupyter-svc
spec:
  ports:
  - port: 80
    targetPort: 8888
    name: jupyter
  selector:
    k8s-app: jupyter
  type: LoadBalancer
```

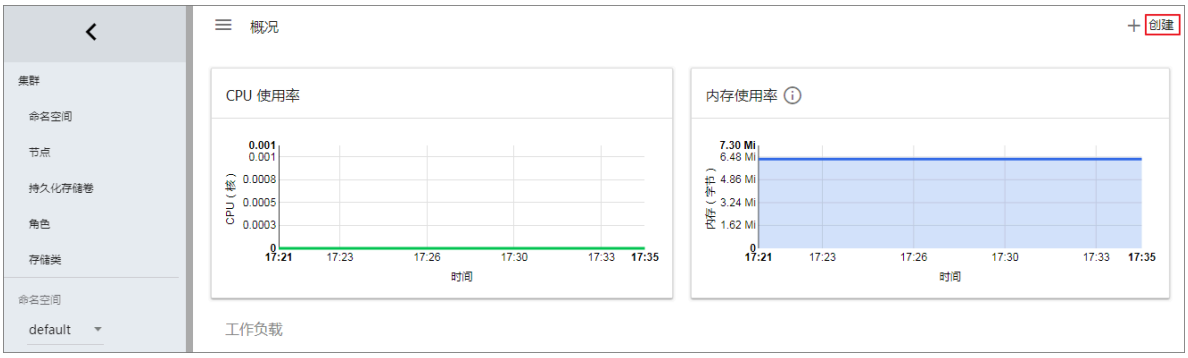
Deployment配置：

- `alpha.kubernetes.io/nvidia-gpu` 指定调用 Nvidia GPU 的数量。
- `type=LoadBalancer` 指定使用阿里云的负载均衡访问内部服务和负载均衡。

- 为了能让 GPU 容器运行起来，需要将 Nvidia 驱动和 CUDA 库文件指定到容器中。这里需要使用 hostPath，在阿里云上您只需要将 hostPath 指定到 `/var/lib/nvidia-docker/volumes/nvidia_driver/375.39` 即可，并不需要指定多个 bin 和 lib 目录。
- 环境变量 `PASSWORD` 指定了访问 Jupyter 服务的密码，您可以按照您的需要修改。

创建应用

1. 按照 通过 Web UI 管理应用 介绍的方式连接 Kubernetes Web UI， 单击创建创建应用。

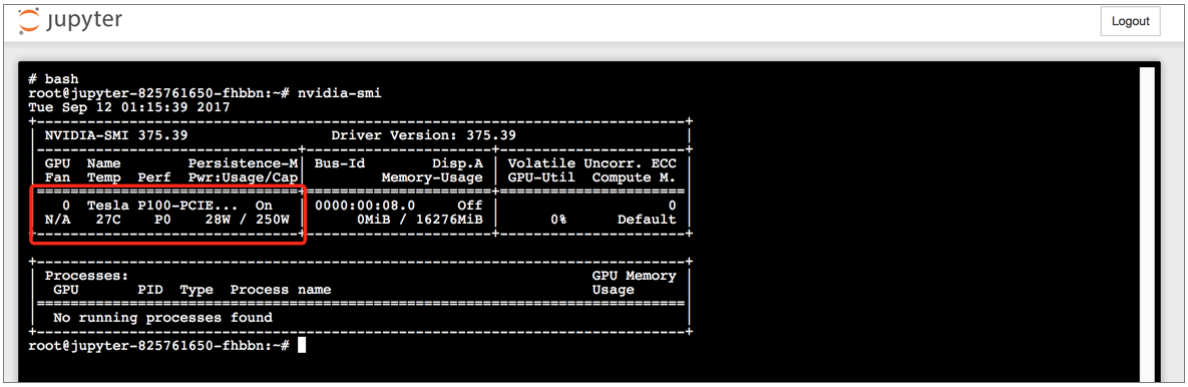


2. 单击使用文件创建。选择刚才创建的 jupyter.yml 文件

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

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

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



4.3 Kubernetes集群中通过 Ingress 实现灰度发布和蓝绿发布

4.3.1 Kubernetes集群中通过 Ingress 实现蓝绿发布

在发布应用时，经常需要先上线一个新版本，用较小的流量去测试一下该新版本的可用性。Kubernetes 的 ingress resource 并没有实现流量控制与切分的功能，导致针对同一个域名下的路径，只能有一个 service 来进行服务，这样对于灰度发布十分不利。本文介绍如何利用阿里云容器服务的路由功能，来实现蓝绿发布，轻松实现流量切分。

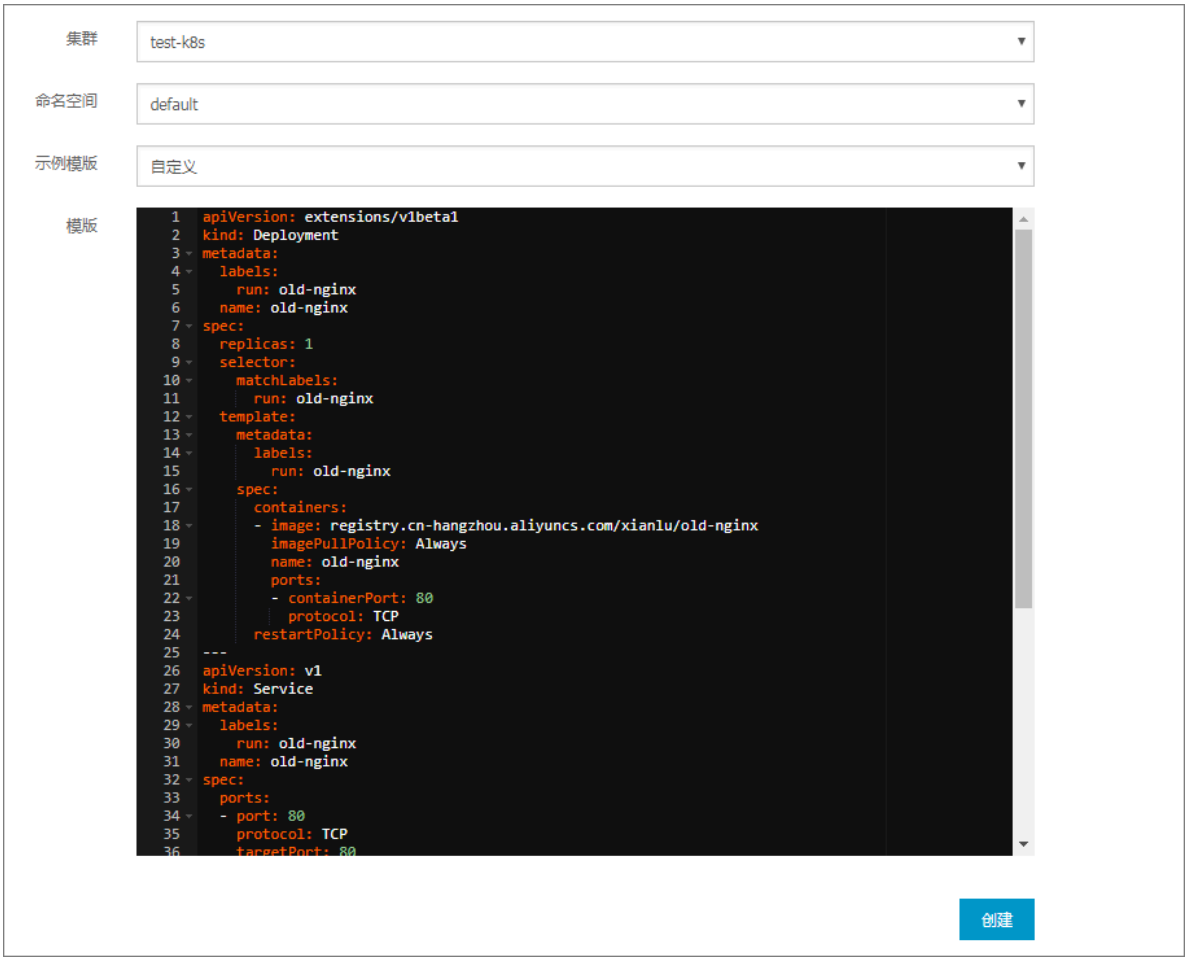
前提条件

- 您已成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。
- SSH 连接到 Master 节点，参见SSH 访问 Kubernetes 集群。

步骤1 创建应用

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。

4. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击创建。




```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: old-nginx
  name: old-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: old-nginx
  template:
    metadata:
      labels:
        run: old-nginx
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
        imagePullPolicy: Always
        name: old-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  labels:
    run: old-nginx
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
```

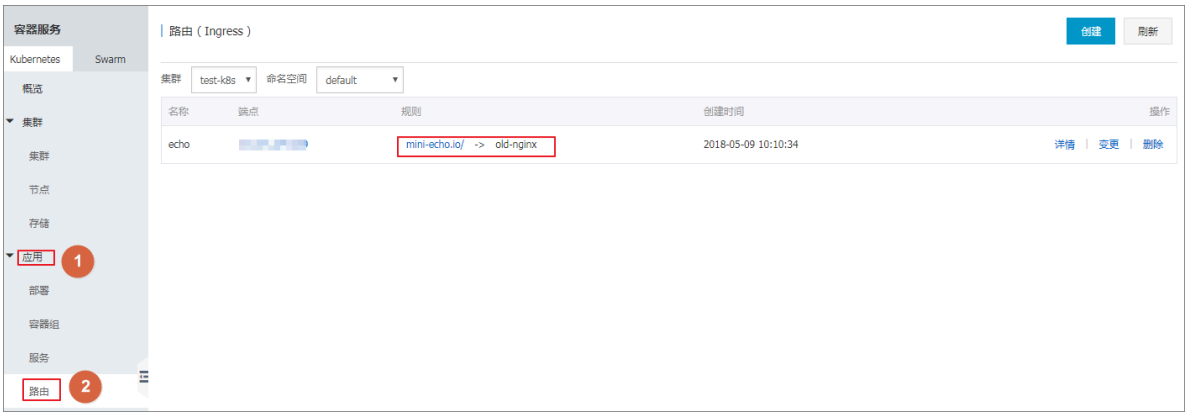
本例是一个 nginx 应用，包含一个 deployment、 service 以及 ingress。deployment 通过 NodePort 对外暴露端口，并且有一个 ingress 正在对外提供服务。编排模板如下。

```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   labels:
     run: old-nginx
   name: old-nginx
 spec:
   replicas: 1
   selector:
     matchLabels:
       run: old-nginx
   template:
     metadata:
       labels:
         run: old-nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
         imagePullPolicy: Always
         name: old-nginx
         ports:
         - containerPort: 80
           protocol: TCP
       restartPolicy: Always
```

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    run: old-nginx
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: old-nginx
  sessionAffinity: None
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: echo
spec:
  backend:
    serviceName: default-http-backend
    servicePort: 80
  rules:
  - host: mini-echo.io                                    ##虚拟主机名称
    http:
      paths:
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
```

5. 创建成功后，单击左侧导航栏中的应用 > 路由。

   您可看到虚拟主机名称指向 old-nginx。



6. 登录 Master 节点，执行 curl 命令，查看路由的访问情况。

```
# curl -H "Host: mini-echo.io" http://101.37.224.229
 ##即 Ingress 的访问地址
old
```

步骤2 创建新版本的 deployment 和 service

1. 登录容器服务管理控制台。

2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。

4. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击创建。

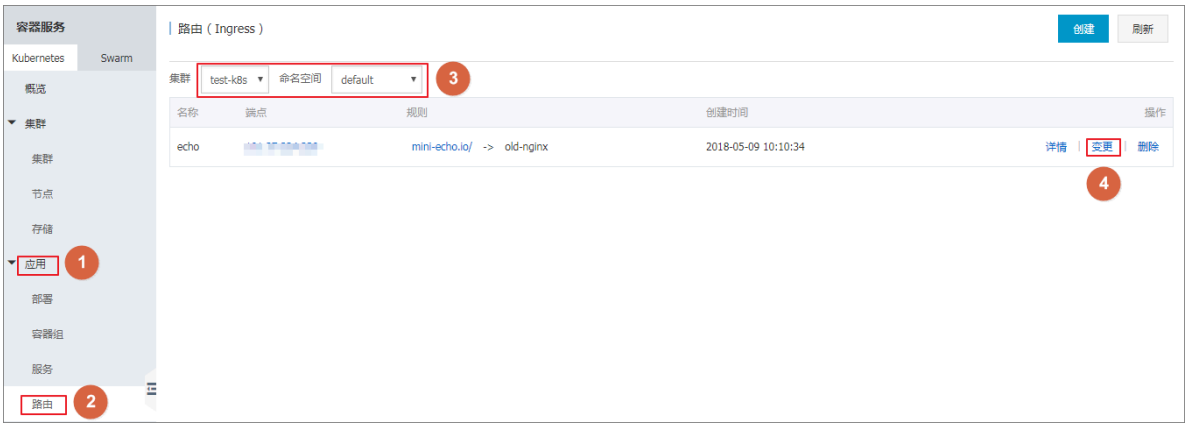
   新版本的 deployment 和service 的编排如下。

```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   labels:
     run: new-nginx
   name: new-nginx
 spec:
   replicas: 1
   selector:
     matchLabels:
       run: new-nginx
   template:
     metadata:
       labels:
         run: new-nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
         imagePullPolicy: Always
         name: new-nginx
         ports:
         - containerPort: 80
           protocol: TCP
       restartPolicy: Always
 ---
 apiVersion: v1
 kind: Service
 metadata:
   labels:
     run: new-nginx
   name: new-nginx
 spec:
   ports:
   - port: 80
     protocol: TCP
     targetPort: 80
   selector:
     run: new-nginx
   sessionAffinity: None
```

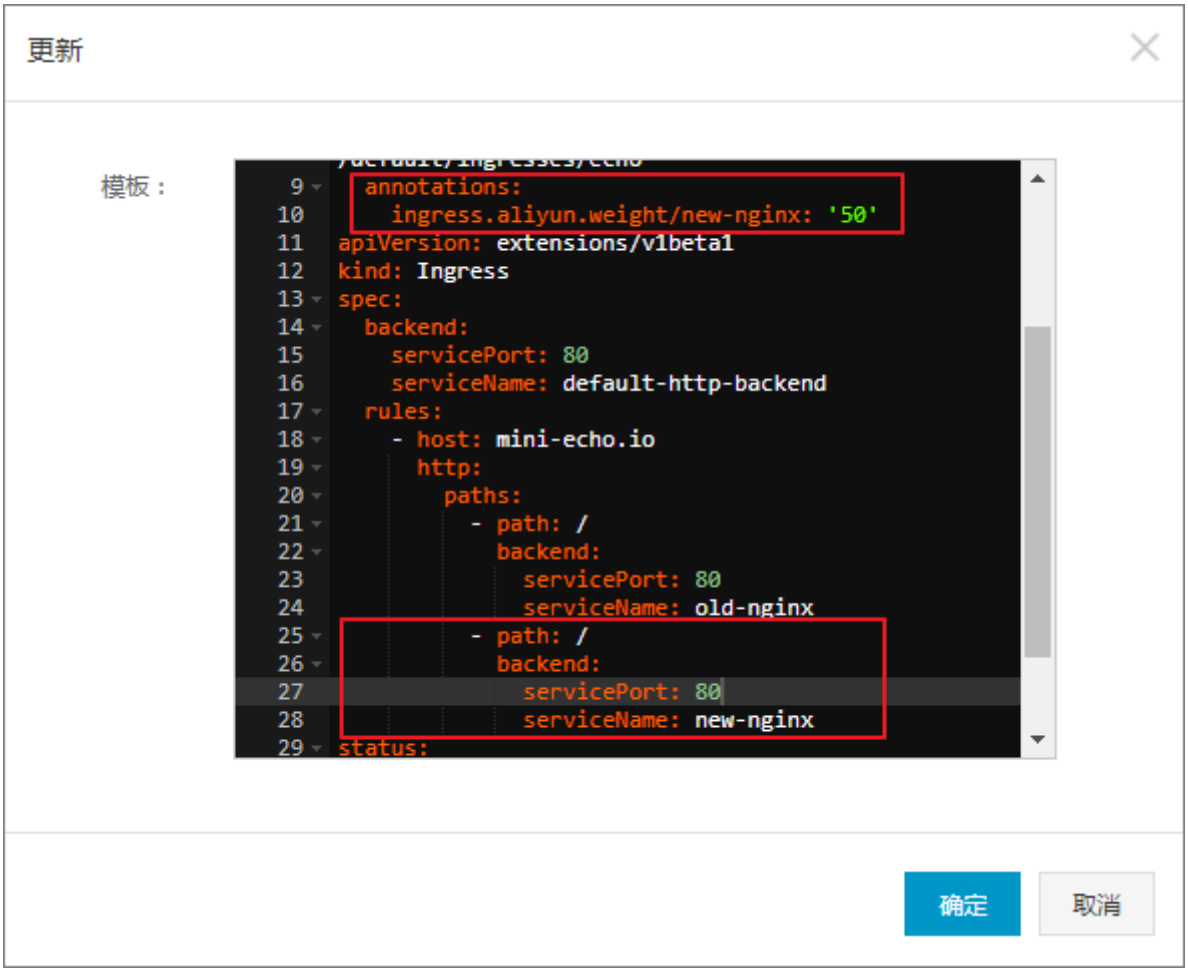
```
    type: NodePort
```

步骤3 修改 Ingress 实现蓝绿发布

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 路由，进入路由列表页面。
3. 选择所需的集群和命名空间，选择前面创建的路由，并单击右侧的更新。



4. 在弹出的对话框中，对 Ingress 进行修改，最后单击确定。



- 添加注解（annotations）：`/`后面为新服务的服务名 new-nginx；50 是一个流量值，代表 50%。标签`ingress.aliyun.weight/new-nginx: "50"`完整含义是将流量的百分之 50 引入到新服务的 pod 里面。
- 配置新的 serviceName：这里是和上面旧版本服务并列，即在相同的 path 下，挂两个不同的 service，分别对应于两个新老应用。

返回路由列表页面，您可看到该路由新增了一条路由规则，指向新版本的 new-nginx 服务。

5. 登录 Master 节点，执行 curl 命令，查看路由的访问情况。

```
# curl -H "Host: mini-echo.io" http://101.37.224.229
 ##即 Ingress 的访问地址
 old
 # curl -H "Host: mini-echo.io" http://101.37.224.229
 new
```

```
# curl -H "Host: mini-echo.io" http://101.37.224.229
old
# curl -H "Host: mini-echo.io" http://101.37.224.229
new
# curl -H "Host: mini-echo.io" http://101.37.224.229
old
# curl -H "Host: mini-echo.io" http://101.37.224.229
new
```

本例中，执行 6 次 curl 命令，分别得到三次新服务，三次老服务的返回结果，这表明权重设置生效了。

您可通过灵活调整 Ingress 中的注解 `ingress.aliyun.weight/new-nginx: "50"` 的值，来设置蓝绿发布的流量占比。

完成新版本应用测试后，您可将 Ingress 的路由权重设置为 100 将流量完全导向新服务；或者删除 Ingress 中的注解和旧版本服务，实现蓝绿发布。

4.3.2 使用限制

本文介绍如何利用阿里云容器服务的Ingress功能，实现灰度发布的使用限制。

阿里云容器服务Kubernetes Ingress Controller的版本需要在`0.12.0-5`及以上，才支持灰度发布。

具体可参考*K8S Ingress Controller* 发布公告

可执行以下命令查看Ingress Controller的当前版本号：

- 采用Deployment部署的情况：

```
kubectl -n kube-system get deploy nginx-ingress-controller -o yaml
 | grep -v 'apiVersion' | grep 'aliyun-ingress-controller'
```

- 采用DaemonSet部署的情况：

```
kubectl -n kube-system get ds nginx-ingress-controller -o yaml |
grep -v 'apiVersion' | grep 'aliyun-ingress-controller'
```

若您的Ingress Controller的版本在`0.12.0-5`以下，可通过以下命令进行升级：

- 采用Deployment部署的情况：

```
kubectl -n kube-system set image deploy/nginx-ingress-controller
 nginx-ingress-controller=registry.cn-hangzhou.aliyuncs.com/acs/
aliyun-ingress-controller:0.12.0-5
```

- 采用DaemonSet部署的情况：

```
kubectl -n kube-system set image ds/nginx-ingress-controller nginx
-ingress-controller=registry.cn-hangzhou.aliyuncs.com/acs/aliyun-
ingress-controller:0.12.0-5
```

4.3.3 Annotation

本文介绍阿里云容器服务的Ingress功能实现灰度发布使用的Annotation。

阿里云容器服务Kubernetes的Ingress功能通过Annotation：路由规则`nginx.ingress.kubernetes.io/service-match`及服务权重`nginx.ingress.kubernetes.io/service-weight`来支持应用服务的灰度发布。

说明：

如果用户同时配置了：路由规则`nginx.ingress.kubernetes.io/service-match`及服务权重`nginx.ingress.kubernetes.io/service-weight`，当收到一个请求时，先判断是否满足已配置的路由规则`nginx.ingress.kubernetes.io/service-match`：

- 若不满足，则会将请求转发到老版本中。
- 若满足，则会按照配置的服务权重`nginx.ingress.kubernetes.io/service-weight`转发请求。

路由规则`nginx.ingress.kubernetes.io/service-match`

该Annotation用来配置新版本服务的路由匹配规则，格式如下：

```
nginx.ingress.kubernetes.io/service-match: |
    <service-name>: <match-rule>
```

参数解释

`service-name`：服务名称，满足match-rule的请求会被路由到该服务中。

`match-rule`：路由匹配规则。

- 匹配类型：
  - header：基于请求头，支持正则匹配和完整匹配。
  - cookie：基于cookie，支持正则匹配和完整匹配。
  - query：基于请求参数，支持正则匹配和完整匹配。
- 匹配方式：
  - 正则匹配格式：/{regular expression}/
  - 完整匹配格式："{exact expression}"

配置示例

```
# 请求头中满足foo正则匹配^bar$的请求被转发到新版本服务new-nginx中
new-nginx: header("foo", /^bar$/)

# 请求头中满足foo完整匹配bar的请求被转发到新版本服务new-nginx中
new-nginx: header("foo", "bar")

# cookie中满足foo正则匹配^sticky-.+$的请求被转发到新版本服务new-nginx中
new-nginx: cookie("foo", /^sticky-.+$/)

# query param中满足foo完整匹配bar的请求被转发到新版本服务new-nginx中
new-nginx: query("foo", "bar")
```

服务权重nginx.ingress.kubernetes.io/service-weight

该Annotation用来配置新老版本服务的流量权重，格式如下：

```
nginx.ingress.kubernetes.io/service-weight: |
    <new-svc-name>:<new-svc-weight>, <old-svc-name>:<old-svc-weight>
```

参数解释

new-svc-name：新版本服务名称。

new-svc-weight：新版本服务权重。

old-svc-name：老版本服务名称。

old-svc-weight：老版本服务权重。

配置示例

```
nginx.ingress.kubernetes.io/service-weight: |
    new-nginx: 20, old-nginx: 60
```

说明：

- 服务权重采用相对值计算方式。如配置示例中的新版服务权重设置为20，旧版服务权重设置为60，则：新版服务的权重百分比为25%，旧版服务的权重百分比为75%。
- 一个服务组（同一个Ingress yaml中具有相同Host和Path的服务）中未明确设置权重的服务默认权重值为100。

4.3.4 步骤1：部署服务

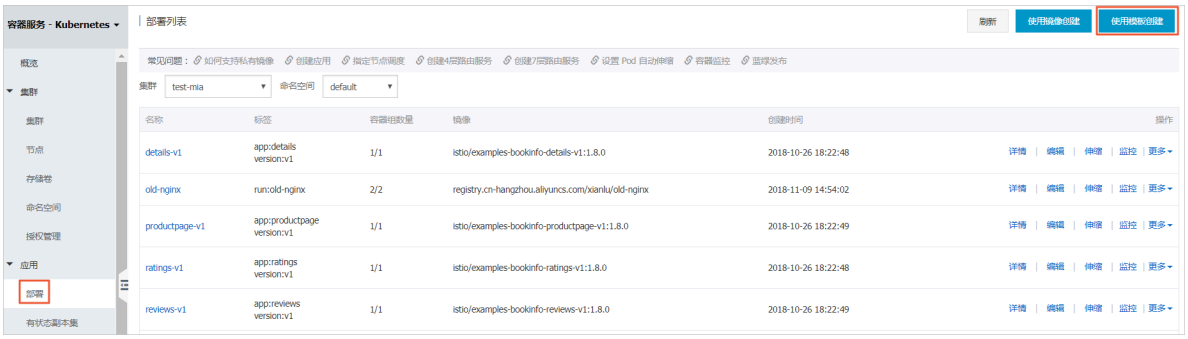
本文介绍如何部署服务。

前提条件

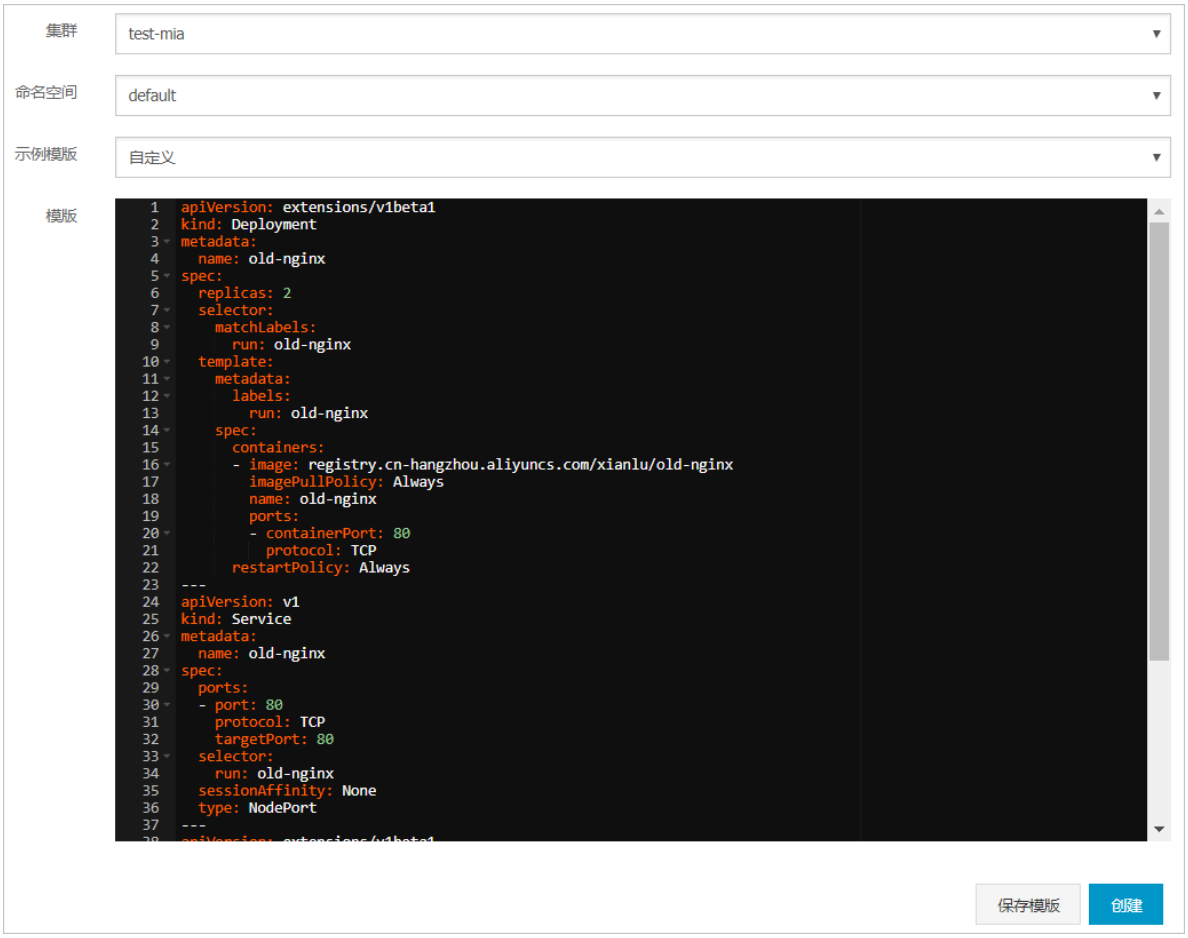
- 您已成功部署一个Kubernetes集群，参见创建Kubernetes集群。
- 您可以通过Kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。



4. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击创建。



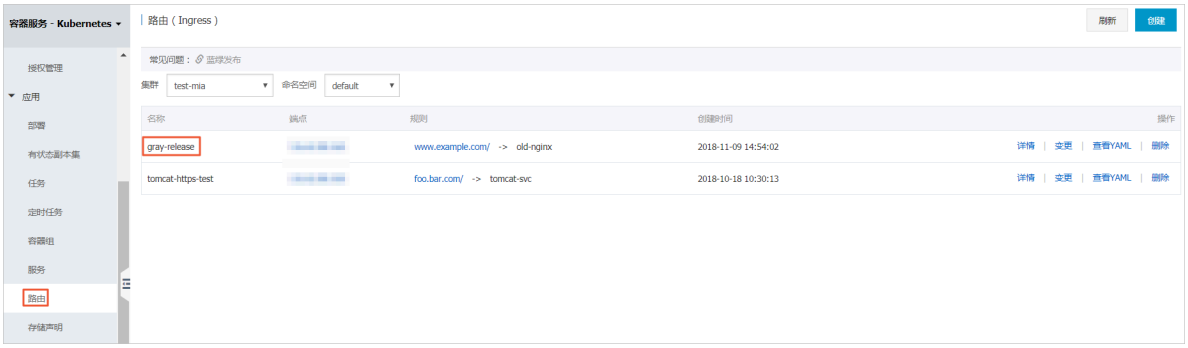
本例是一个 nginx 应用，包含一个 deployment、 service 以及 ingress。deployment 通过 NodePort 对外暴露端口，并且有一个 ingress 正在对外提供服务。编排模板如下。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: old-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: old-nginx
  template:
    metadata:
      labels:
        run: old-nginx
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
        imagePullPolicy: Always
        name: old-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: old-nginx
  sessionAffity: None
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
    name: gray-release
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # 老版本服务
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
```

5. 创建成功后，单击左侧导航栏中的应用 > 路由。

   您可看到虚拟主机名称指向 old-nginx。



6. 登录 Master 节点，执行 curl 命令，查看路由的访问情况。

```
# curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

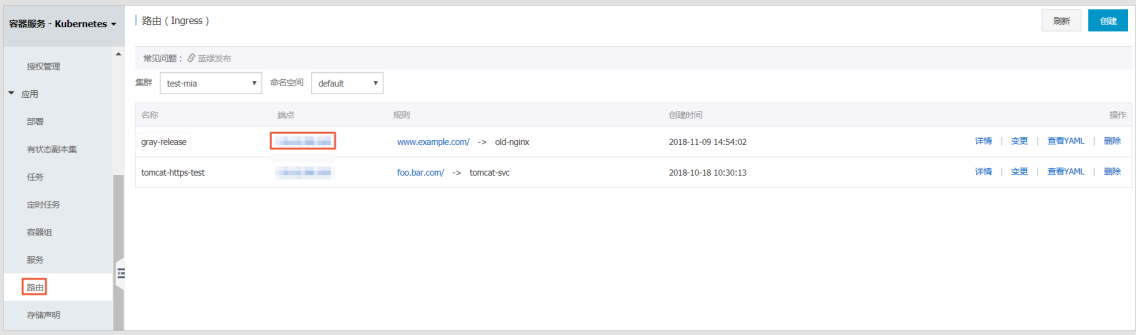
说明:

<EXTERNAL_IP>可通过以下两种方式获取:

- 执行以下命令获取：

```
kubectl get ingress
```

- 在Kubernetes菜单下，单击应用 > 路由进入路由（Ingress）页面，选择目标路由，查看对应的端点信息。




```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
```

4.3.5 步骤2：发布新版本服务

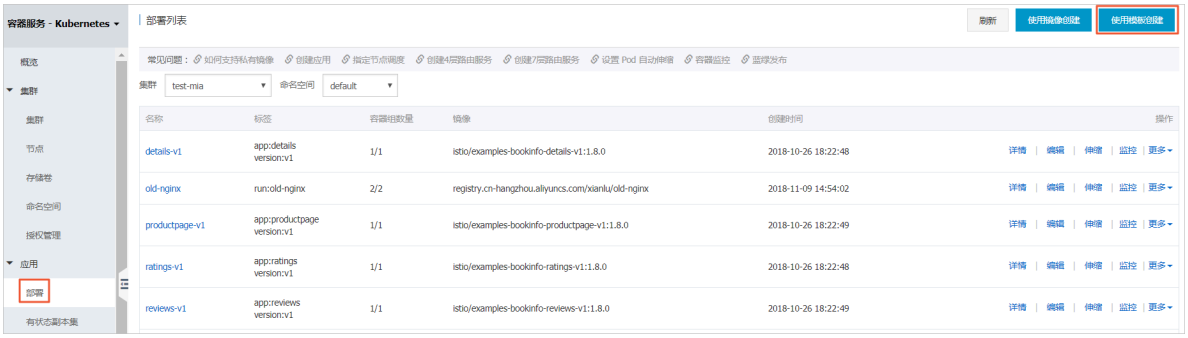
本文介绍如何利用阿里云容器服务的Ingress功能，发布新版本服务。

前提条件

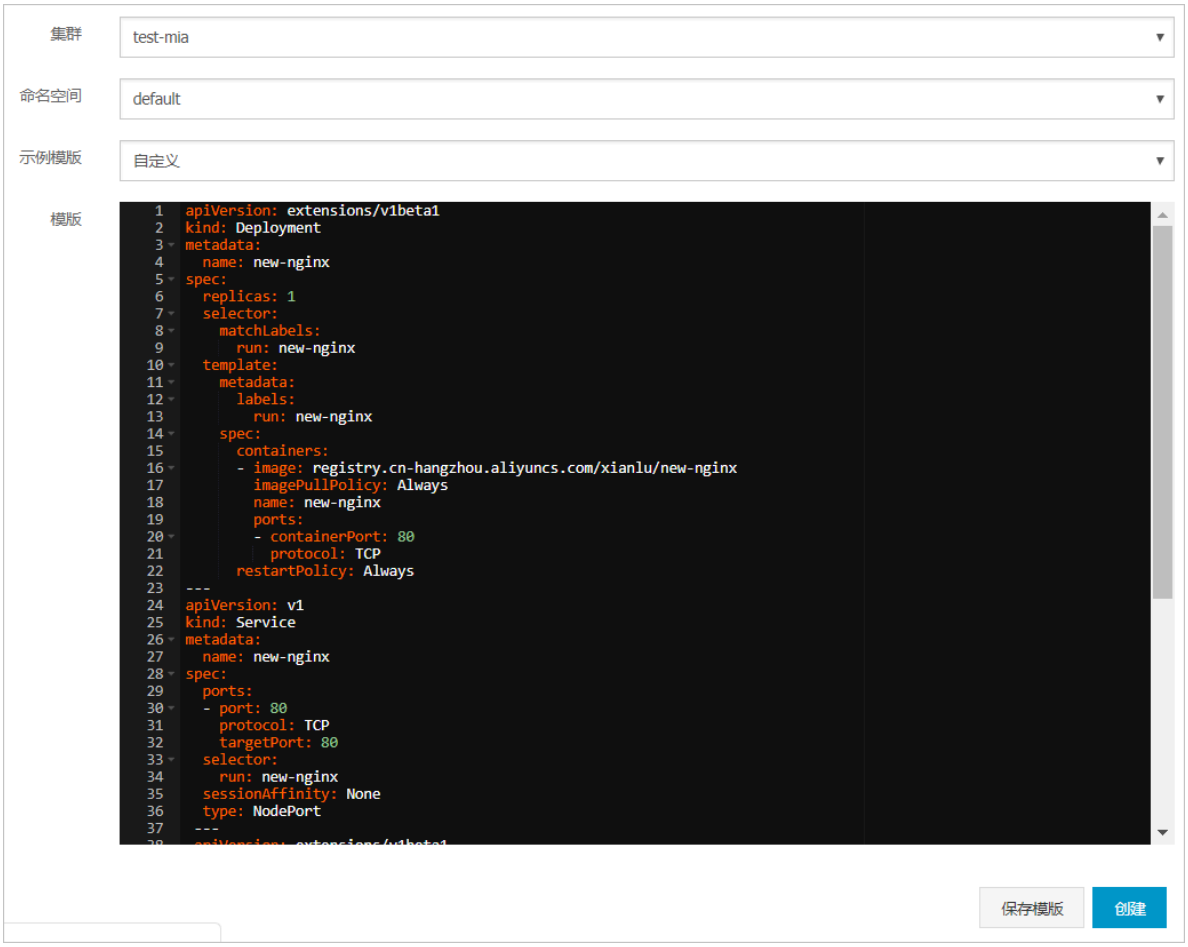
- 您已成功部署一个Kubernetes集群，参见创建Kubernetes集群。
- 您可以通过Kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。

操作步骤

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 部署，进入部署列表页面。
3. 单击页面右上角的使用模板创建。



4. 选择所需的集群，命名空间，选择样例模板或自定义，然后单击创建。



部署新版本nginx应用，包含一个 deployment、 service 以及 ingress。deployment及 service 编排模板如下：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: new-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: new-nginx
  template:
    metadata:
      labels:
        run: new-nginx
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
        imagePullPolicy: Always
        name: new-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: new-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: new-nginx
  sessionAffinity: None
  type: NodePort
```

ingress模板：

说明：

若 ingress的Anotation既没有设置service-match，也没有设置service-weight，Ingress Controller在转发客户端请求时默认会将请求均衡地随机转发到新老版本服务中。

- **满足正则匹配foo=bar的客户端请求才能访问新版本服务**

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gray-release
  annotations:
      nginx.ingress.kubernetes.io/service-match: |      # 请求头中满足正则匹配foo=bar的请求才会被路由到新版本服务new-nginx中
          new-nginx: header("foo", /^bar$/)
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # 老版本服务
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
      # 新版本服务
      - path: /
        backend:
          serviceName: new-nginx
          servicePort: 80
```

- **满足一定比例的请求被路由到新版本服务**

说明：

此处新旧版本服务权重分别为50%。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gray-release
  annotations:
```

```
      nginx.ingress.kubernetes.io/service-weight: |    # 允许50%的
流量被路由到新版本服务new-nginx中
          new-nginx: 50, old-nginx: 50
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # 老版本服务
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
      # 新版本服务
      - path: /
        backend:
          serviceName: new-nginx
          servicePort: 80
```

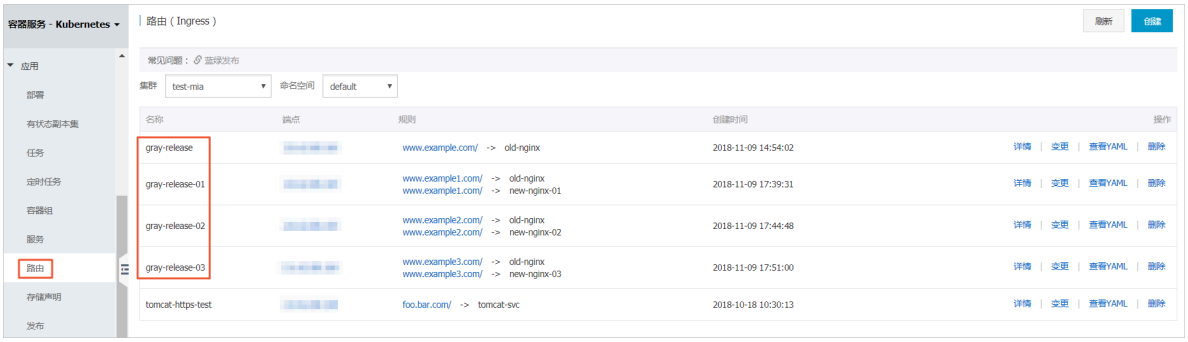
- 满足foo=bar的客户端请求仅允许50%的流量被路由到新版本服务

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gray-release
  annotations:
    nginx.ingress.kubernetes.io/service-match: |    # 请求头中满足正
则匹配foo=bar的请求才会被路由到新版本服务new-nginx中
        new-nginx: header("foo", /^bar$/)
    nginx.ingress.kubernetes.io/service-weight: |   # 在满足上述匹配
规则的基础上仅允许50%的流量会被路由到新版本服务new-nginx中
        new-nginx: 50, old-nginx: 50
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # 老版本服务
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
      # 新版本服务
      - path: /
        backend:
          serviceName: new-nginx
```

```
            servicePort: 80
```

5. 创建成功后，单击左侧导航栏中的应用 > 路由。

   您可看到虚拟主机名称指向 old-nginx。



6. 登录 Master 节点，执行 curl 命令，查看路由的访问情况。

   - 满足正则匹配foo=bar的客户端请求才能访问新版本服务

```
# curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_I
P>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://
new
```

   - 满足一定比例的请求被路由到新版本服务

```
# curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
old
```

   - 满足foo=bar的客户端请求仅允许50%的流量被路由到新版本服务

```
# curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_I
P>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example3.com" -H "foo: bar" http://
new
```

4.3.6 步骤3：删除老版本服务

本文介绍灰度发布新版本服务，系统运行一段时间，新版本服务稳定后，如何删除老版本服务。

前提条件

- 您已成功部署一个Kubernetes集群，参见创建Kubernetes集群。
- 您可以通过Kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。
- 您已部署老版本服务，参见步骤1#部署服务，同时已灰度发布新版本服务，参见步骤2#发布新版本服务。

通过命令行删除

1. 执行以下命令，编辑步骤2#发布新版本服务已经部署的yaml文件，删除老版本的服务。

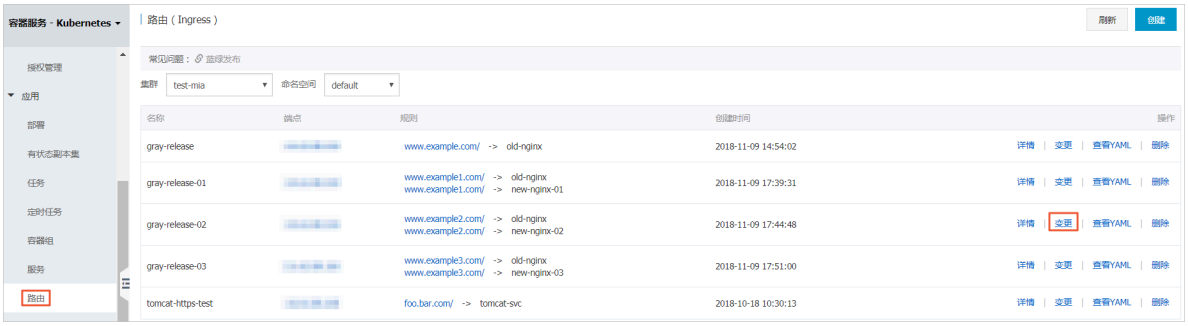
   说明:

   请将`annotations`一并删除。

   ```
   $ kubectl get ingress gray-release-02
   ```

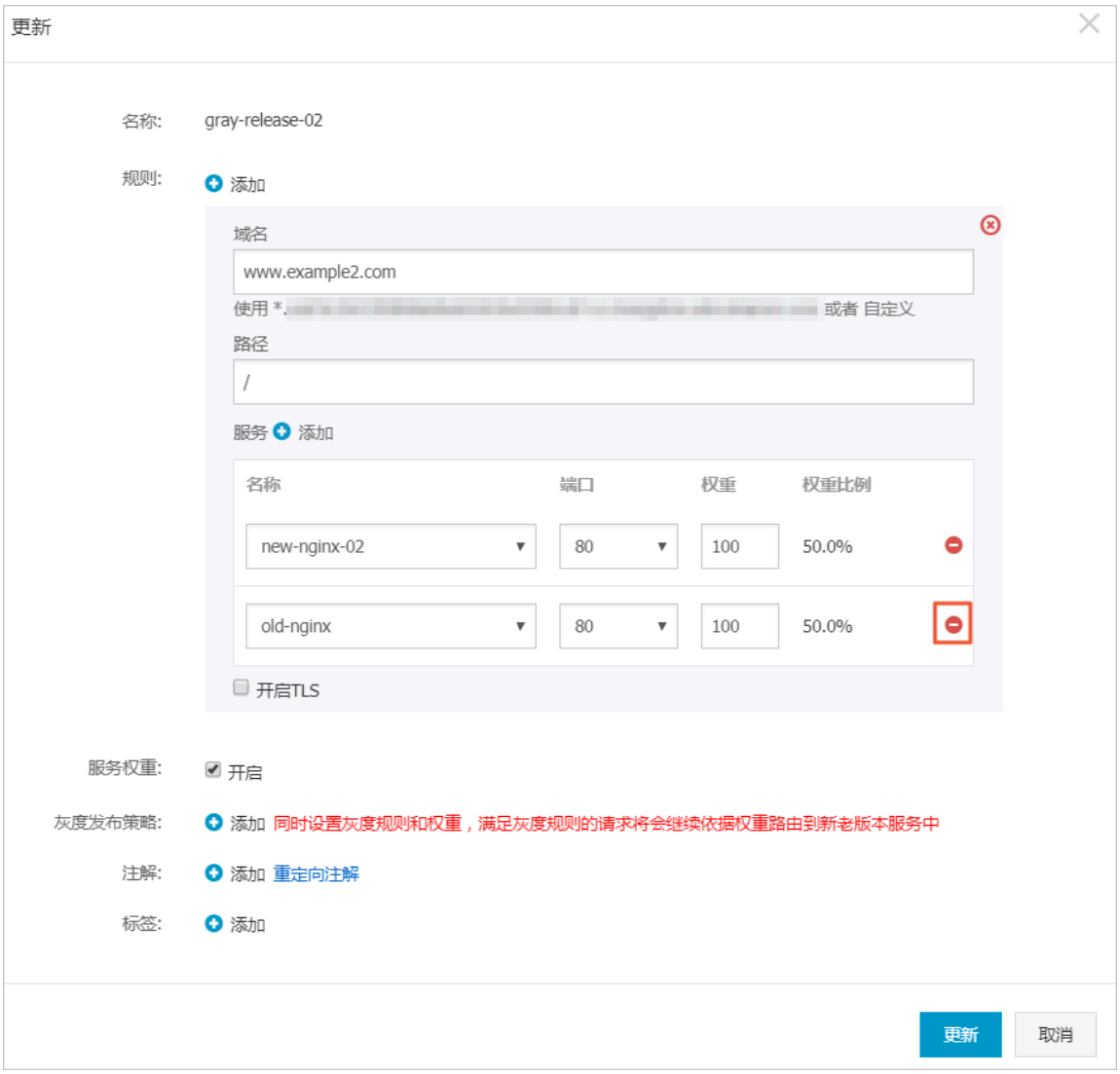
通过控制台删除

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 路由，进入路由列表页面。
3. 选择所需的集群和命名空间，选择前面创建的路由，并单击操作列的变更。



4. 在弹出的对话框中，对 Ingress 进行修改：

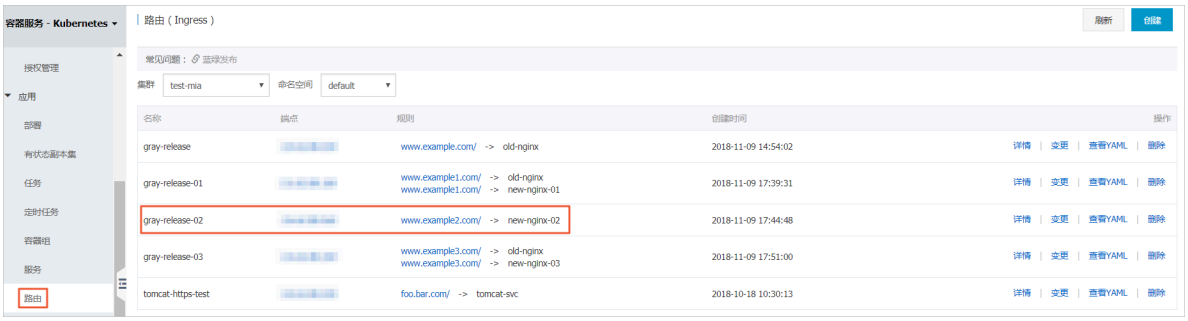
   a. 在规则 > 服务区域，删除老版本服务规则。



   b. 单击更新。

执行结果

1. 返回路由列表页面，您可看到只有一条路由规则，指向新版本的 new-nginx 服务。



2. 登录 Master 节点，执行 curl 命令，查看路由的访问情况。

```
$ curl -H "Host: www.example2.com" http://<EXTERNAL_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
```

可以看到，现在的请求全部被路由到了新版本的服务中，至此完成了灰度发布的整个周期。最后，您也可以删除老版本的deployment和service。

5 Istio

5.1 在Kubernetes上基于Istio实现Service Mesh智能路由

阿里云容器服务支持一键部署Istio，并支持多种扩展功能，本例中介绍如何通过Istio实现智能路由。Istio官方文档请参考*intelligent-routing*。

前提条件

- 您已成功部署一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您已成功部署了Istio，参见部署*Istio*。

> 说明:
> 本例中Isito的版本是1.0.2。

- 您需要拥有一个本地Linux环境，并已配置好Kubectl工具连接到集群，参见通过 *kubectl* 连接 *Kubernetes* 集群。
- 您已下载对应Istio版本的项目代码，并在Istio文件目录下执行相关命令。参见*https://github.com/istio/istio/releases*。

安装官方示例

首先快速安装Bookinfo官方示例，具体可以参见： *https://istio.io/docs/guides/bookinfo*。

快速部署Bookinfo示例应用

1. 首先为`default` 命名空间打上标签 `istio-injection=enabled`。

> 说明:
> 阿里云容器服务Kubernetes集群支持一键部署Istio，并支持自动Sidecar注入。

```
$ kubectl label namespace default istio-injection=enabled
```

2. 使用Kubectl命令部署Bookinfo示例应用。

```
$ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
```

上面的命令会启动全部的四个服务，其中也包括了 `reviews` 服务的三个版本（v1、v2 以及 v3）。

3. 确认所有的服务和 Pod 都已经正确的定义和启动。

```
$ kubectl get svc,pods
```

4. 你需要从外部访问应用程序，例如使用浏览器。您需要创建一个*Istio Gateway*。为应用程序定义入口网关。

```
$ kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```

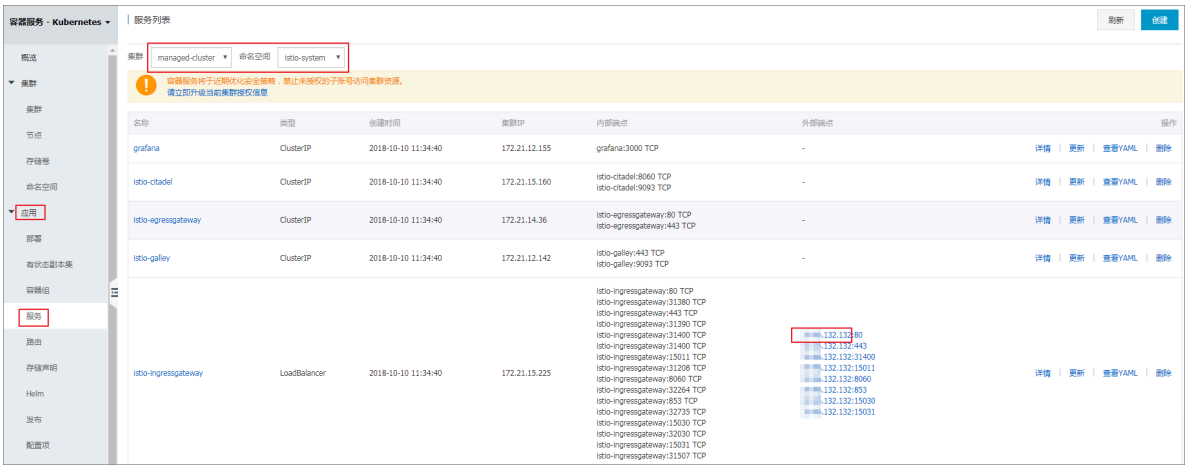
确认网关创建完成：

```
$ kubectl get gateway
NAME                 AGE
bookinfo-gateway   32s
```

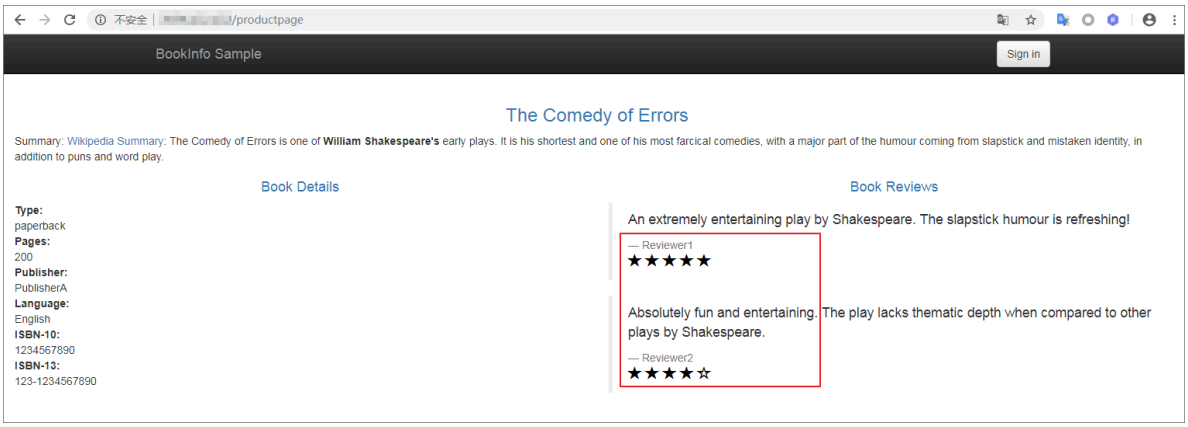
5. 然后确认`istio-ingressgateway`的访问地址。您可通过命令行快速查询。

```
$ kubectl get svc istio-ingressgateway -n istio-system
```

您也可通过容器服务管理控制台，在左侧导航栏单击应用 > 服务，选择集群和Istio-system命名空间，查看`istio-ingressgateway`服务的IP地址。



6. 访问BookInfo首页，地址是`http://{EXTERNAL-IP}/productpage`。



多次刷新浏览器，将在 productpage 中看到评论的不同的版本，它们会按照 round robin（红星、黑星、没有星星）的方式展现，因为目前为止还没有使用 Istio 来控制版本的路由。

请求路由

BookInfo示例部署了三个版本的reviews服务，因此需要设置一个缺省路由。否则当多次访问该应用程序时，会发现有时输出会包含带星级的评价内容，有时又没有。出现该现象的原因是当没有为应用显式指定缺省路由时，Istio会将请求随机路由到该服务的所有可用版本上。

在使用 Istio 控制 Bookinfo 版本路由之前，你需要在目标规则中定义好可用的版本 。

运行以下命令为 Bookinfo 服务创建的默认的目标规则：

- 如果不需要启用双向TLS，请执行以下命令：

```
$ kubectl apply -f samples/bookinfo/networking/destination-rule-all.
yaml
```

- 如果需要启用双向 TLS，请执行以下命令：

```
$ kubectl apply -f samples/bookinfo/networking/destination-rule-all-
mtls.yaml
```

等待几秒钟，等待目标规则生效。你可以使用以下命令查看目标规则：

```
$ kubectl get destinationrules -o yaml
```

将所有微服务的缺省版本设置为v1

通过运行如下命令，将所有微服务的缺省版本设置为v1：

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-all-v1.
yaml
```

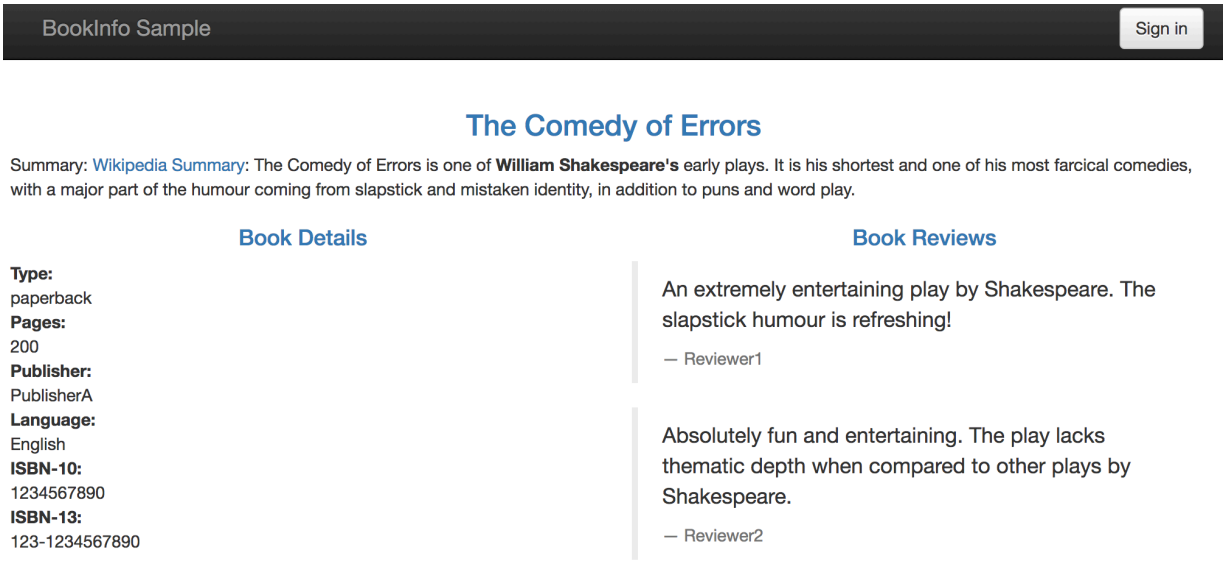
可以通过下面的命令来显示所有已创建的路由规则：

```
kubectl get virtualservices -o yaml
```

由于路由规则是通过异步方式分发到代理的，过一段时间后规则才会同步到所有pod上。因此需要等几秒钟后再尝试访问应用。

在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

可以看到BookInfo应用程序的productpage页面，显示的内容中不包含带星的评价信息，这是因为reviews:v1服务不会访问ratings服务。



将来自特定用户的请求路由到reviews:v2

通过运行如下命令，把来自测试用户"jason"的请求路由到reviews:v2，以启用ratings服务。

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-reviews
-test-v2.yaml
```

可以通过如下命令确认规则是否创建：

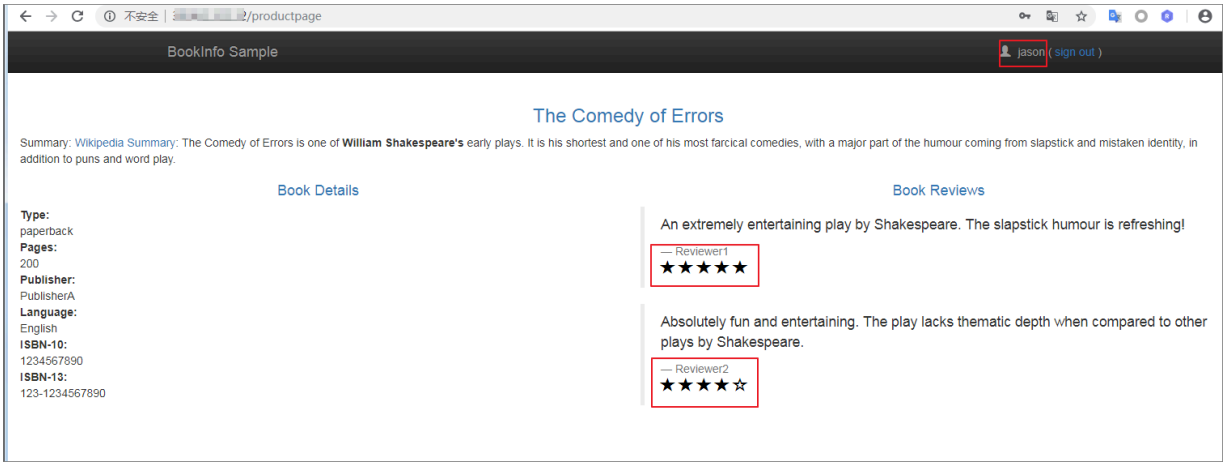
```
$ kubectl get virtualservice reviews -o yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
        subset: v2
  - route:
    - destination:
        host: reviews
        subset: v1
```

确认规则已创建之后，在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

以"jason"用户登录productpage页面，您可以在每条评价后面看到星级信息。

说明:

账号名称和密码都是`jason`。



说明:

本例中，首先使用Istio将100%的请求流量都路由到了BookInfo服务的v1版本；然后再设置了一条路由规则，路由规则基于请求的header（例如一个用户cookie）选择性地将特定的流量路由到了reviews服务的v2版本。

故障注入

为了测试BookInfo微服务应用的弹性，我们计划针对"jason"用户在`reviews:v2`和`ratings`服务之间 注入7秒的延迟 。由于 reviews:v2 服务针对调用ratings服务设置了10秒的超时，因此期望端到端的流程能无错持续。

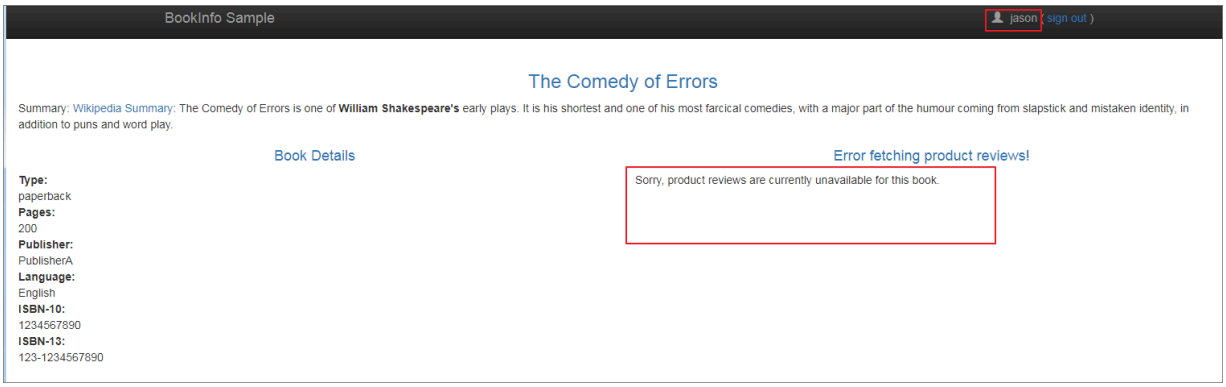
HTTP Delay

使用HTTP Delay创建一个故障注入规则，延迟来自用户jason的流量：

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-ratings-test-delay.yaml
```

确认规则已创建之后，在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

以"jason"用户登录productpage页面，您可看到如下画面：



说明:

整个review服务失败的原因：productpage和review服务之间的超时小于（3秒加上一次重试，总共6秒）review服务和rating服务之间的超时（10秒）。在由不同开发团队负责独立开发不同微服务的典型企业应用中，这类bug就会发生。Istio的故障注入规则有助于识别这些异常，而无需影响到最终用户。

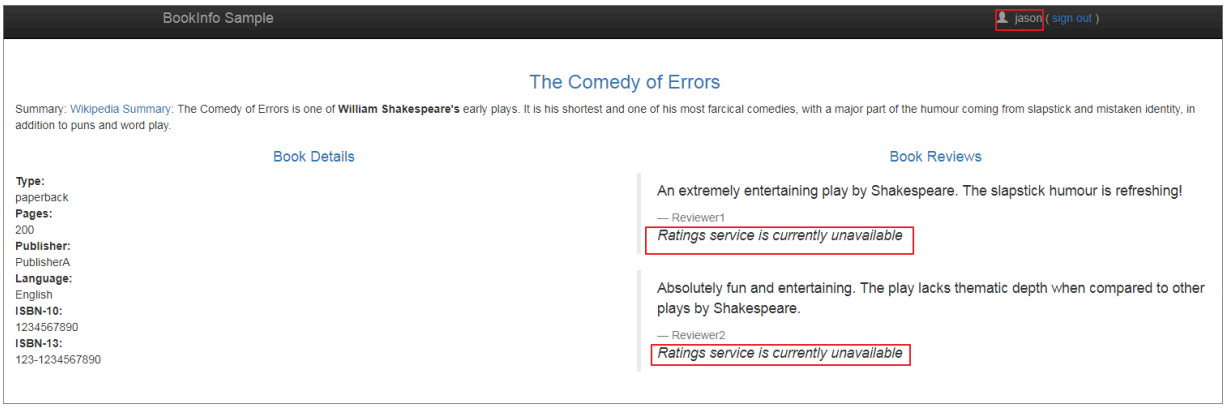
HTTP Abort

类似地，使用HTTP Abort创建一个故障注入规则：

```
$ kubectl apply -f samples/bookinfo/networking/virtual-service-ratings
-test-abort.yaml
```

确认规则已创建之后，在浏览器中打开BookInfo应用程序的URL: `http://{EXTERNAL-IP}/productpage`。

以"jason"用户登录productpage页面，应该可以看到如下画面：



流量转移

除了基于内容的路由，Istio还支持基于权重的路由规则。

首先，将所有微服务的缺省版本设置为v1：

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-all-
v1.yaml
```

其次，使用下面的命令把50%的流量从reviews:v1转移到reviews:v3:

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-
reviews-50-v3.yaml
```

在浏览器中多次刷新productpage页面，大约有50%的几率会看到页面中出现带红星的评价内容。

说明:

注意该方式和使用容器编排平台的部署特性来进行版本迁移是完全不同的。容器编排平台使用了实例scaling来对流量进行管理。而通过Istio，两个版本的reviews服务可以独立地进行扩容和缩容，并不会影响这两个版本服务之间的流量分发。

如果觉得 `reviews: v3` 微服务已经稳定，你可以通过以下命令， 将virtual service 100%的流量路由到 `reviews: v3`，从而实现一个灰度发布的功能。

```
$ kubectl replace -f samples/bookinfo/networking/virtual-service-reviews-v3.yaml
```

总结

我们可以利用阿里云Kubernetes容器服务，快速搭建一套用于连接、管理以及安全化微服务的开放平台Istio，为应用引入和配置多个相关服务。本文通过一个官方示例来尝试了Istio 的流量路由、故障注入、流量转移等功能。欢迎大家使用阿里云上的容器服务，快速搭建微服务的开放治理平台Istio，比较简单地集成到自己项目的微服务开发中。

5.2 kubernetes上实现 Istio 分布式追踪

本文通过一个示例演示了如何在启用了Istio的应用中使用分布式追踪系统Jaeger。

在由单体架构迁移至微服务时，传统的监视工具往往无法提供跨越不同服务的可见性。因此就有必要引入分布式跟踪的工具。

为了解决不同的分布式追踪系统 API 不兼容的问题，诞生了 OpenTracing 规范。OpenTracing 是一个轻量级的标准化层，它位于应用程序/类库和追踪或日志分析程序之间。OpenTracing 已进入 CNCF，正在为全球的分布式追踪，提供统一的概念和数据标准。它通过提供平台无关、厂商无关的 API，使得开发人员能够方便的添加（或更换）追踪系统的实现。

Jaeger 是*CNCF*下的一款开源分布式追踪系统，兼容 OpenTracing API。

前提条件

- 您已成功部署一个Kubernetes集群，参见*创建Kubernetes集群*。
- 您需要拥有一个本地Linux环境，并已配置好Kubectl工具连接到集群，参见*通过 kubectl 连接 Kubernetes 集群*。
- 您已下载对应Istio版本的项目代码，并在Istio文件目录下执行相关命令。参见*https://github.com/istio/istio/releases*。

步骤1 一键部署 Jaeger 追踪系统

1. 登录 容器服务管理控制台。
2. 单击左侧导航栏的集群，选择所需集群，单击更多 > 部署Istio。

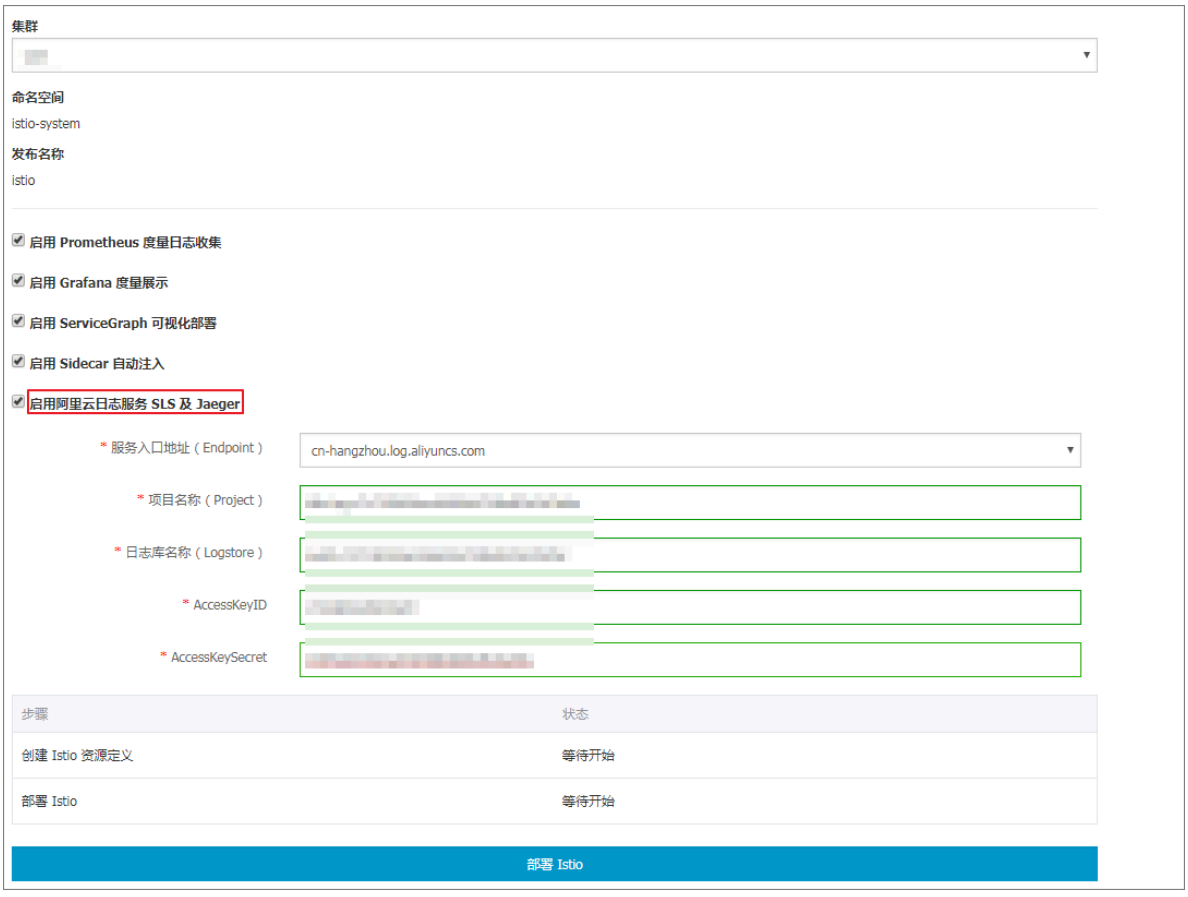
说明:

本例中Isito的版本是1.0，更多关于部署Istio的信息，请参见部署*Istio*。

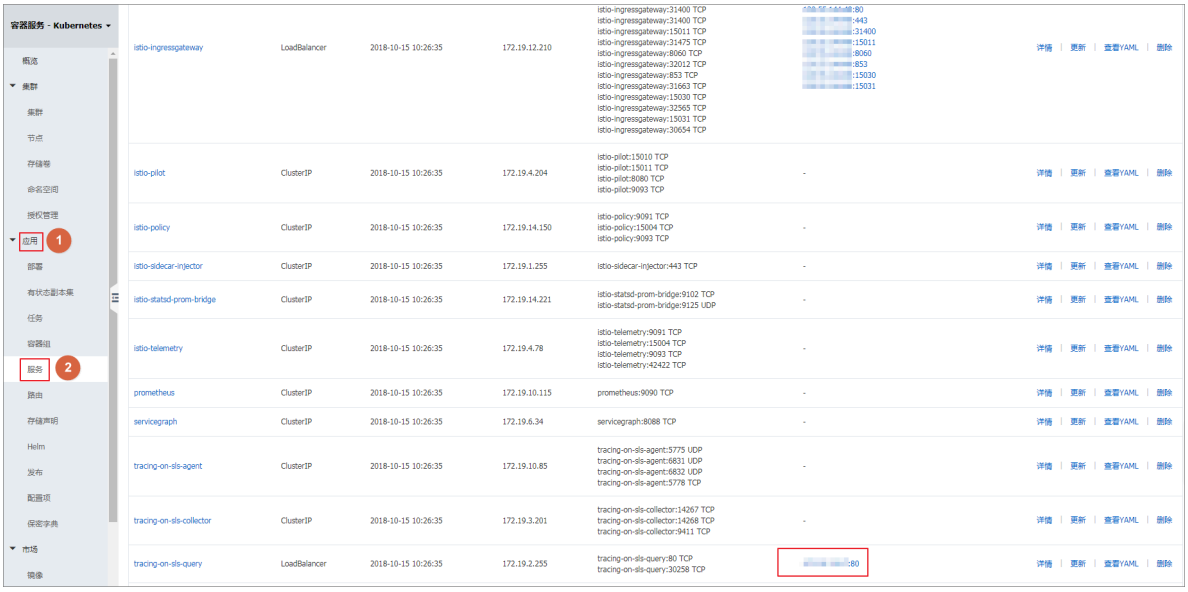
3. 在部署 Istio 页面中，选择启用阿里云日志服务 SLS 及 Jaeger。阿里云容器服务支持通过Web界面一键部署Jaeger追踪系统。

说明:

- 如果用户输入的项目名称（Project）不存在，Istio部署过程中会自动创建，用户无需手工构建日志服务的配置。
- 如果用户输入的项目名称（Project）存在，会检查用户输入的日志库名称（Logstore）是否存在于该项目下，如果不存在，Istio部署中会自动创建，用户无需手工构建日志服务的配置；如果日志库名称（Logstore）存在于该项目下，Istio部署中会创建该日志服务所需的索引(注意：会覆盖已有的索引)。



4. 部署完成后，在左侧菜单栏中单击应用 > 服务，选择所需集群和命名空间，找到tracing-on-sls-query服务。



5. 单击tracing-on-sls-query服务的外部地址，即可进入Jaeger UI页面。

   您可开始利用Jaeger观察服务间调用链，诊断性能问题、分析系统故障。

步骤2 部署分布式服务追踪示例Bookinfo

1. 首先为`default`命名空间打上标签`istio-injection=enabled`。

   > **说明:**
   > 阿里云容器服务Kubernetes集群支持一键部署Istio，并支持自动Sidecar注入。更多信息，参见BookInfo指南。

   ```
   $ kubectl label namespace default istio-injection=enabled
   ```

2. 使用Kubectl命令部署Bookinfo示例应用。

   ```
   $ kubectl apply -f samples/bookinfo/platform/kube/bookinfo.yaml
   ```

   上面的命令会启动全部的四个服务，其中也包括了`reviews`服务的三个版本（v1、v2 以及 v3）。

3. 确认所有的服务和 Pod 都已经正确的定义和启动。

```
$ kubectl get svc,pods
```

4. 你需要从外部访问应用程序，例如使用浏览器。您需要创建一个*Istio Gateway*。为应用程序定义入口网关。

```
$ kubectl apply -f samples/bookinfo/networking/bookinfo-gateway.yaml
```

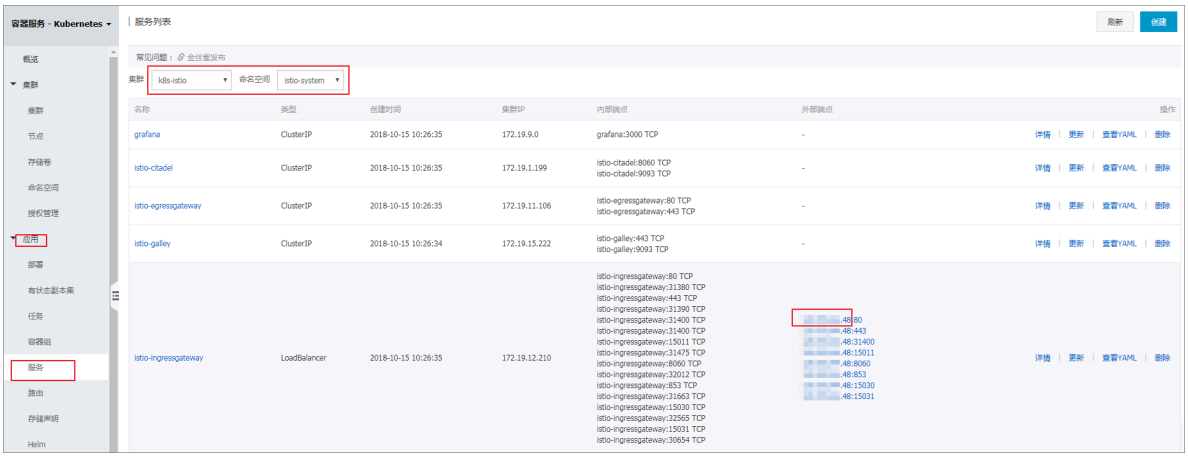
确认网关创建完成：

```
$ kubectl get gateway
NAME                 AGE
bookinfo-gateway   32s
```

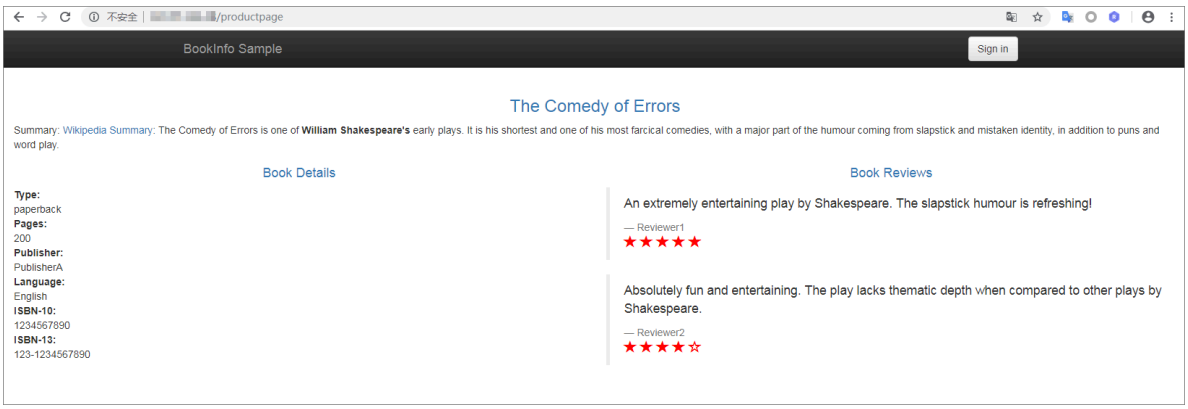
5. 然后确认`istio-ingressgateway`的访问地址。您可通过命令行快速查询。

```
$ kubectl get svc istio-ingressgateway -n istio-system
```

您也可通过容器服务管理控制台，在左侧导航栏单击应用 > 服务，选择集群和Istio-system命名空间，查看`istio-ingressgateway`服务的IP地址。



6. 访问BookInfo首页，地址是`http://{EXTERNAL-IP}/productpage`。



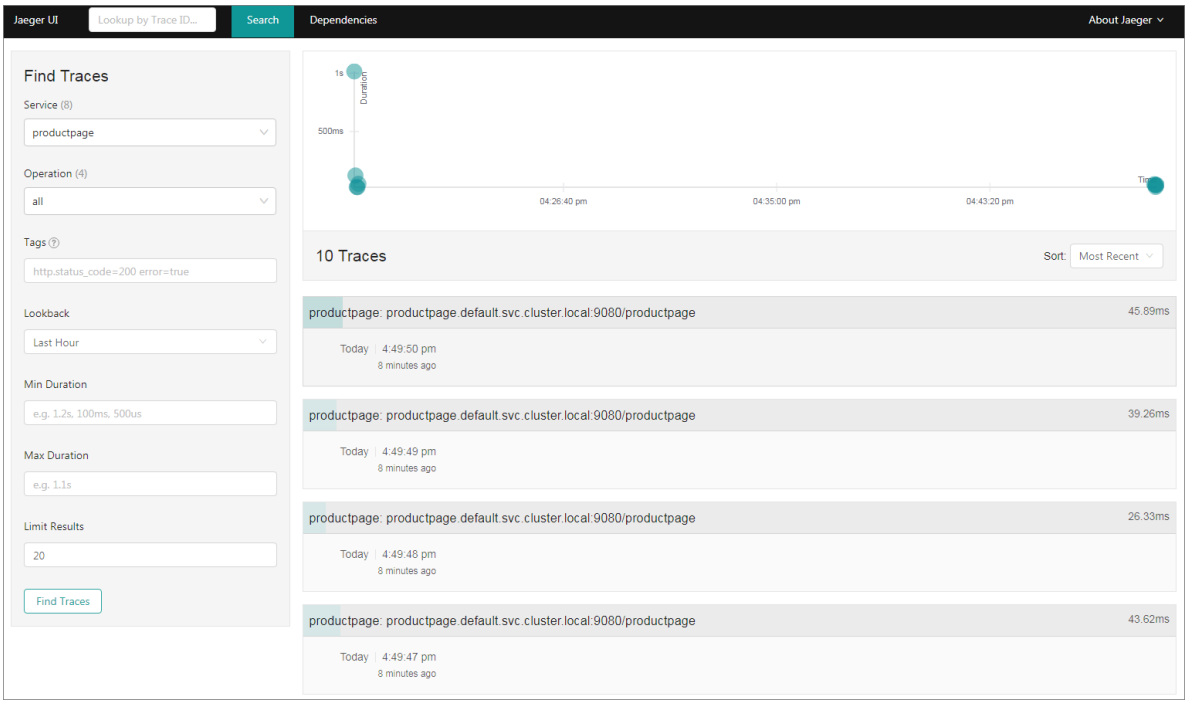
多次刷新浏览器，将在 productpage 中看到评论的不同的版本，它们会按照 round robin（红星、黑星、没有星星）的方式展现，因为目前为止还没有使用 Istio 来控制版本的路由。

步骤3 查看Jaeger追踪结果

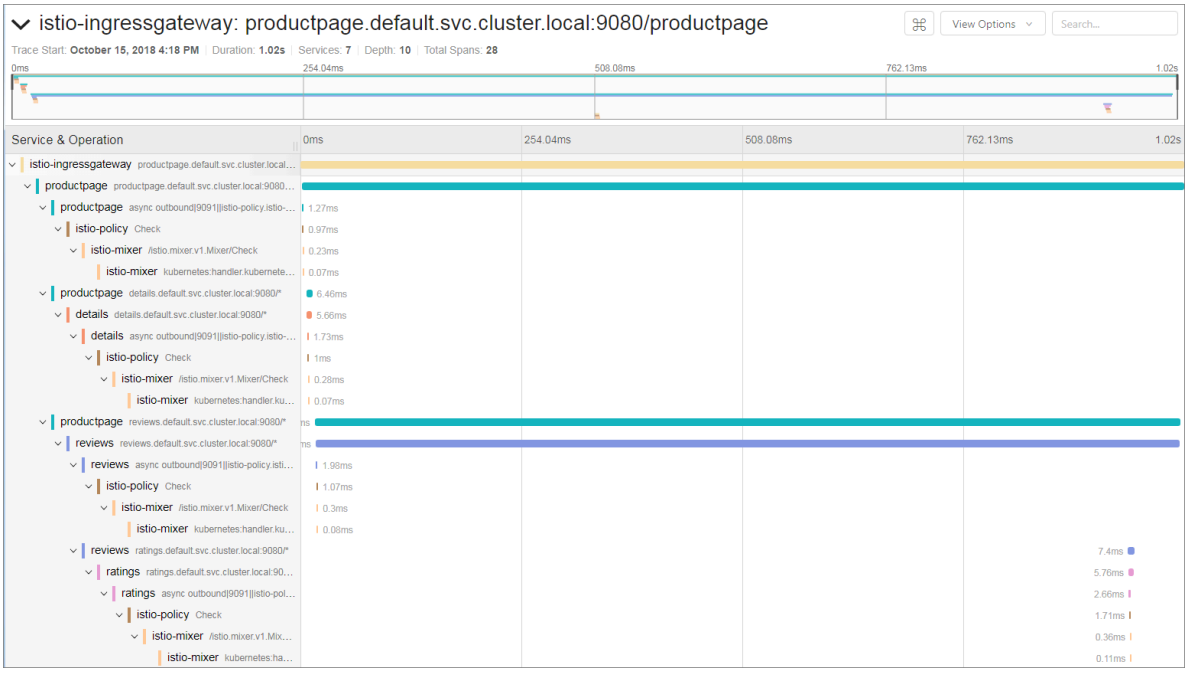
Jaeger UI显示了分布式服务追踪信息的结果。

1. 单击tracing-on-sls-query服务的外部地址，进入Jaeger UI 页面。
2. 在左侧选择服务，然后选择各项配置，最后单击Find Traces，右上角显示的时刻和持续时间散点图用可视化方式呈现了结果，并提供了向下挖掘能力。

> **说明:**
> 本例中以BookInfo示例应用为例，查看productpage服务的调用情况。



3. 用户可以选择用多种不同视图对追踪结果进行可视化，例如追踪时段内的直方图，或服务在追踪过程中的累积时间。



Istio 分布式追踪实现原理

尽管Istio代理能够自动发送spans，但他们需要一些标识来将整个调用链关系联系起来。应用程序需要传入合适的HTTP header信息，便于代理发送span信息到Jaeger时，span可以准确地把每次调用关联起来。

为此，应用程序需要从传入的请求中收集如下的header信息并将其传入到每个传出请求：

```
x-request-id
x-b3-traceid
x-b3-spanid
x-b3-parentspanid
x-b3-sampled
x-b3-flags
x-ot-span-context
```

示例服务中的productpage应用（Python应用）从HTTP请求中提取所需的header信息：

```
defgetForwardHeaders(request):
    headers = {}

    user_cookie = request.cookies.get("user")
    if user_cookie:
        headers['Cookie'] = 'user=' + user_cookie

    incoming_headers = [ 'x-request-id',
                         'x-b3-traceid',
                         'x-b3-spanid',
                         'x-b3-parentspanid',
                         'x-b3-sampled',
                         'x-b3-flags',
                         'x-ot-span-context'
```

```
    ]

    for ihdr in incoming_headers:
        val = request.headers.get(ihdr)
        if val isnotNone:
            headers[ihdr] = val
            #print "incoming: "+ihdr+":"+valreturn headers
```

在应用程序中调用其他服务时，这些header信息会被传播下去形成一个调用链。

具体代码请参见 Istio 文档*https://istio.io/docs/tasks/telemetry/distributed-tracing.html*。

总结

我们可以利用阿里云Kubernetes容器服务，快速搭建一套用于连接、管理以及安全化微服务的开放平台Istio，为应用引入和配置多个相关服务。本文通过一个示例演示了如何在启用了Istio的应用中使用分布式追踪系统Jaeger。 欢迎大家使用阿里云上的容器服务，快速搭建微服务的开放治理平台Istio，比较简单地集成到自己项目的微服务开发中。

5.3 Kubernetes上实现Istio集成阿里云日志服务

阿里云Kubernetes容器服务已经提供了Istio与日志服务Log Service的集成能力，本文通过一个官方示例来重点介绍Istio与基于阿里云日志服务的分布式追踪系统的整合能力。

说明:

在使用阿里云Kubernetes容器服务Istio 1.0的过程中，如果遇到类似CRD版本问题，请参考我们提供的问题分析。 我们会持续更新遇到的问题及其解决方法。

前提条件

- 您已成功部署一个Kubernetes集群，参见创建*Kubernetes*集群。
- 您需要拥有一个本地Linux环境，并已配置好Kubectl工具连接到集群，参见通过 *kubectl* 连接 *Kubernetes* 集群。
- 您已下载对应Istio版本的项目代码，并在Istio文件目录下执行相关命令。参见*https://github.com/istio/istio/releases*。
- 您已成功部署测试应用 BookInfo，参见安装官方示例或者*BookInfo*指南。

回顾 OpenTracing

为了解决不同的分布式追踪系统 API 不兼容的问题，诞生了 OpenTracing 规范。OpenTracing 是一个轻量级的标准化层，它位于应用程序/类库和追踪或日志分析程序之间。OpenTracing 已进入 CNCF，正在为全球的分布式追踪，提供统一的概念和数据标准。它通过提供平台无关、厂商无关的 API，使得开发人员能够方便添加（或更换）追踪系统的实现。

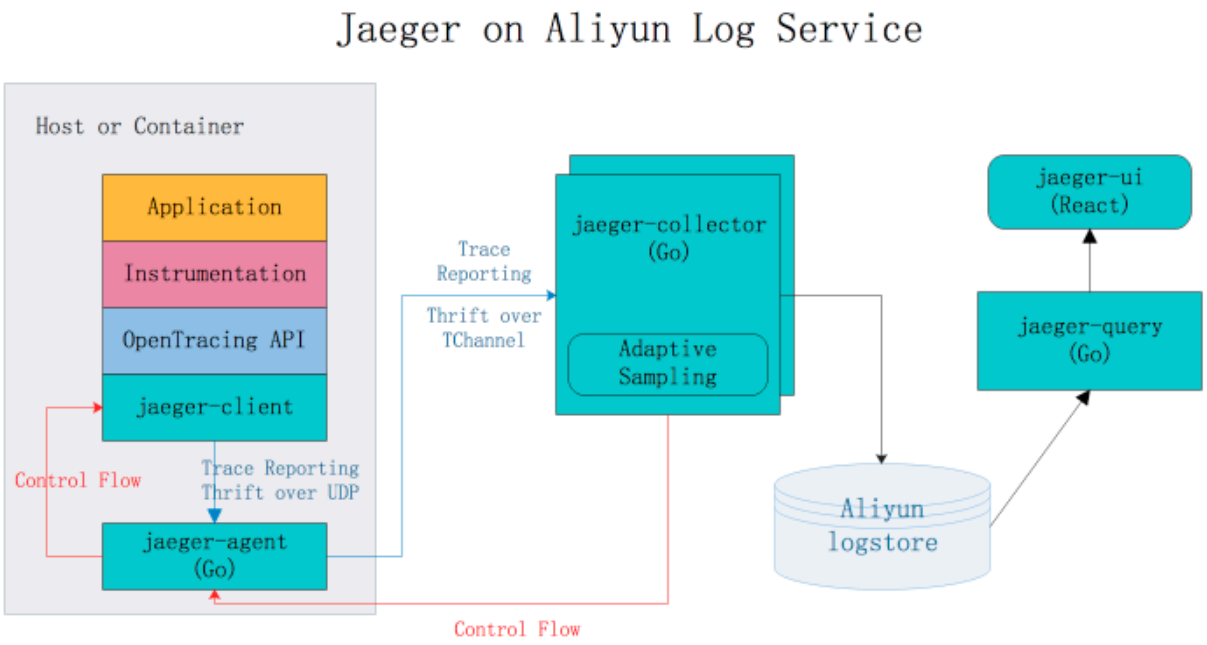
Jaeger 是CNCF下的一款开源分布式追踪系统，兼容 OpenTracing API。

阿里云日志服务Log Service与分布式追踪系统Jaeger

日志服务（Log Service，简称LOG/原SLS）是针对实时数据一站式服务，在阿里集团经历大量大数据场景锤炼而成。提供日志类数据采集、消费、投递及查询分析功能，全面提升海量日志处理/分析能力。

Jaeger 是 Uber 推出的一款开源分布式追踪系统，为微服务场景而生。它主要用于分析多个服务的调用过程，图形化服务调用轨迹，是诊断性能问题、分析系统故障的利器。

Jaeger on Aliyun Log Service 是基于 Jeager 开发的分布式追踪系统，支持将采集到的追踪数据持久化到阿里云日志服务中，并通过 Jaeger 的原生接口进行查询和展示。



组件	说明
Jaeger-client	Jaeger-client 为不同语言实现了符合 OpenTracing 标准的 SDK。应用程序通过 API 写入数据，客户端库library 把 trace 信息按照应用程序指定的采样策略传递给 jaeger-agent。数据使用 Thrift 序列化，通过 UDP 进行通信。
Jaeger-agent	Jaeger-agent 是一个监听在 UDP 端口上接收 span 数据的网络守护进程，它会将数据批量发送给Jaeger-collector。它被设计成一个基础组件，部署到所有的宿主机上。Jaeger-agent 代理将 Jaeger-client 和 Jaeger-collector 解耦，为 Jaeger-client library 屏蔽了路由和发现 collector 的细节。

组件	说明
Jaeger-collector	接收 jaeger-agent 发送来的数据，然后将数据写入后端存储。后端存储是一个可插拔的组件，Jaeger on Aliyun Log Service 增加了对阿里云日志服务的支持。
阿里云日志服务Log Service	Jaeger-collector 会将接收到的 span 数据持久化到日志服务Log Service中。Query 会从日志服务中检索数据。
Query&UI	接收查询请求，从后端存储系统中检索 trace 并通过 UI 进行展示。

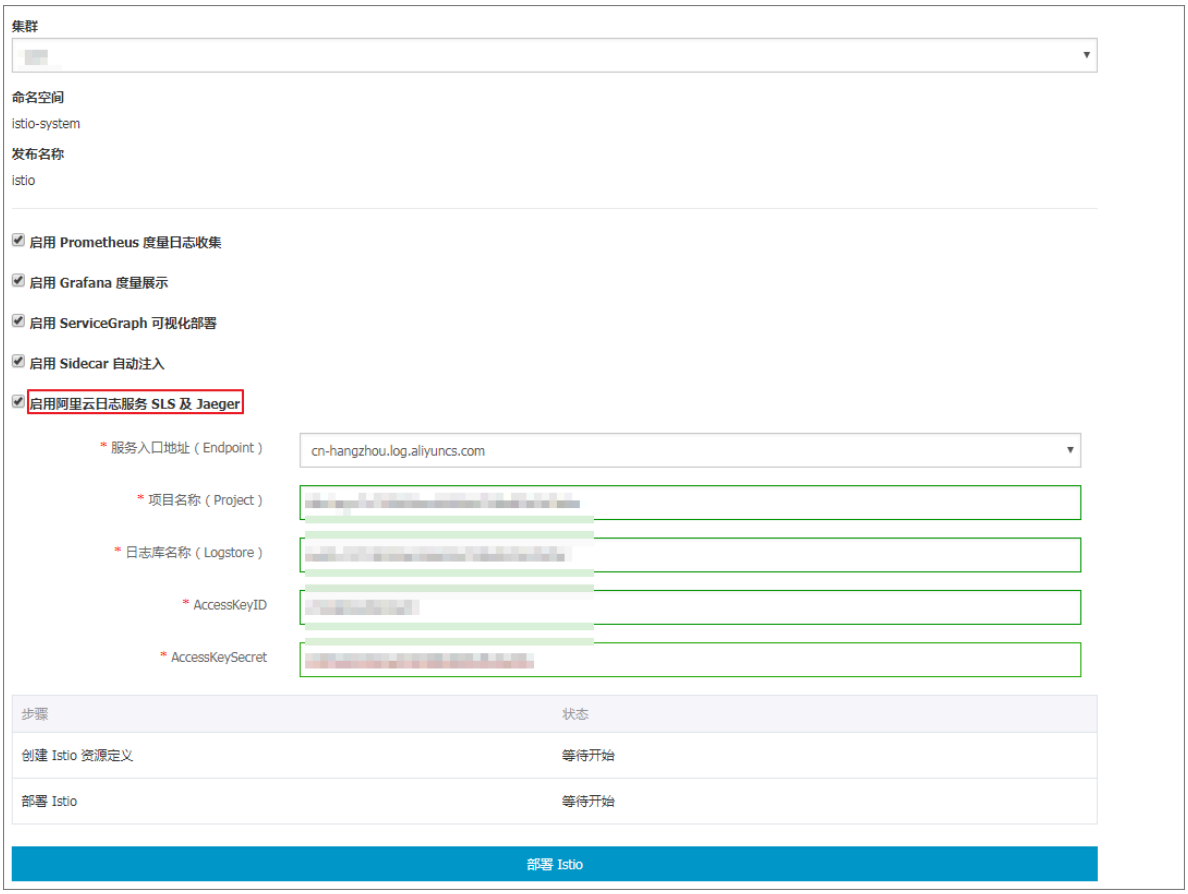
步骤1 通过Web界面一键部署Istio

1. 登录容器服务管理控制台。
2. 单击左侧导航栏的集群，选择所需集群，单击更多 > 部署Istio。

> **说明:**
> 本例中启用阿里云日志服务 SLS 及 Jaeger，更多信息参见部署Istio，Isito的版本是1.0。

3. 在部署 Istio 页面中，选择启用阿里云日志服务 SLS 及 Jaeger。

   阿里云Kubernetes容器服务中整合了日志服务Log Service， 其分布式追踪数据会保存到阿里云日志服务Log Store中。



   启用日志服务的参数说明如下:

参数名	对应参数	描述
服务入口地址（Endpoint）	`storage.aliyun_sls.endpoint`	指定用于存储 Span 的 Project 所在的 Endpoint
项目名称（Project）	`storage.aliyun_sls.project`	指定用于存储 Span 的 Project，项目名称只能包含小写字母、数字和连字符（-），且必须以小写字母和数字开头和结尾，长度为 3~63 个字节

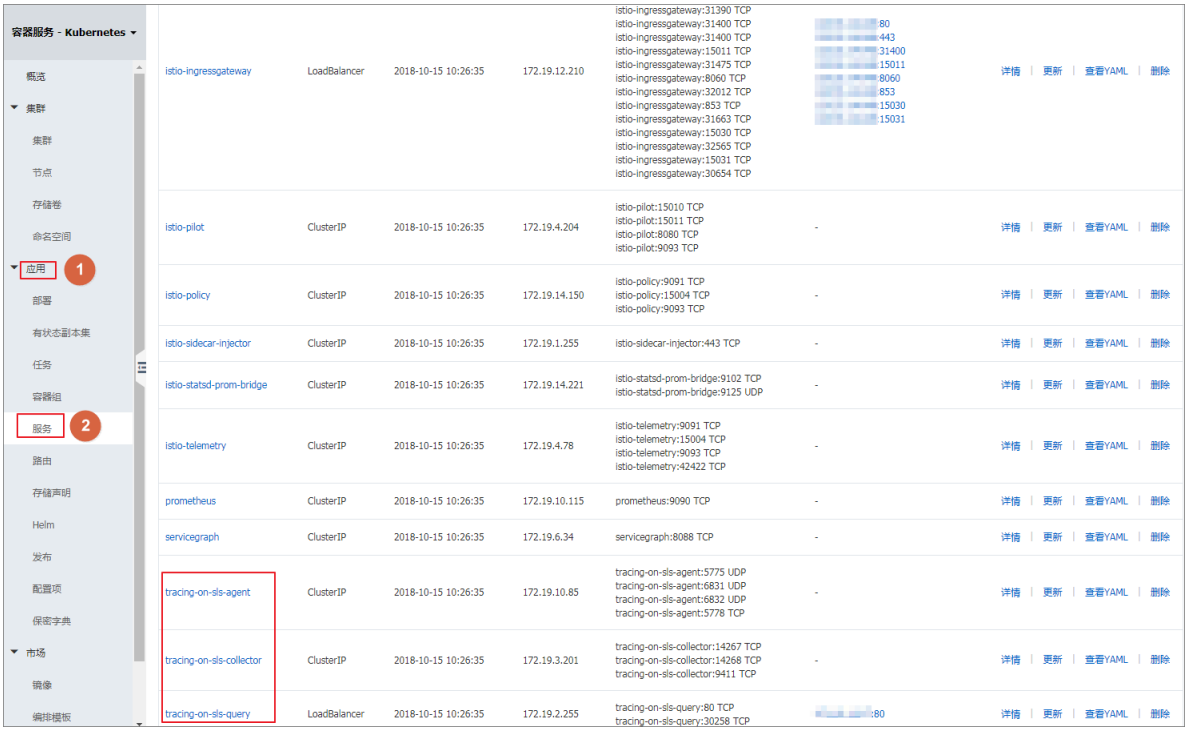
参数名	对应参数	描述
日志库名称（Logstore）	storage.aliyun_sls.logstore	指定用于存储 Span 的 Logstore，须由小写字母、数字、连字符（-）和下划线（_）组成，且以小写字母或者数字开头和结尾，长度为3-63字节。Logstore 名称在其所属项目内必须唯一
AccessKeyID	storage.aliyun_sls.accesskey.id	指定用户标识 Access Key ID
AccessKeySecret	storage.aliyun_sls.accesskey.secret	指定用户标识 Access Key Secret

说明:

- 如果用户输入的项目名称（Project）不存在，Istio部署过程中会自动创建，用户无需手工构建日志服务的配置。
- 如果用户输入的项目名称（Project）存在，会检查用户输入的日志库名称（Logstore）是否存在于该项目下，如果不存在，Istio部署中会自动创建，用户无需手工构建日志服务的配置；如果日志库名称（Logstore）存在于该项目下，Istio部署中会创建该日志服务所需的索引(注意：会覆盖已有的索引)。

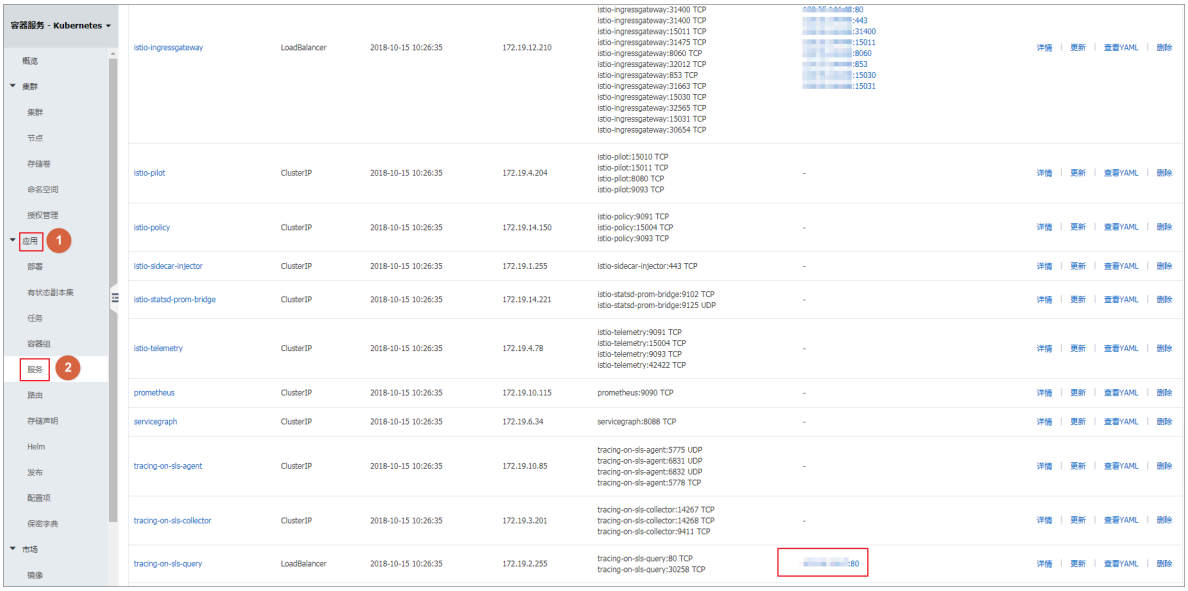
4. 最后单击部署 Istio。几分钟之后，一套集成阿里云日志服务的Istio实例就已创建完毕。

5. 单击左侧导航栏的应用 > 服务，在右侧可以查看到刚创建的Istio相关服务，其中包括集成的日志相关服务。



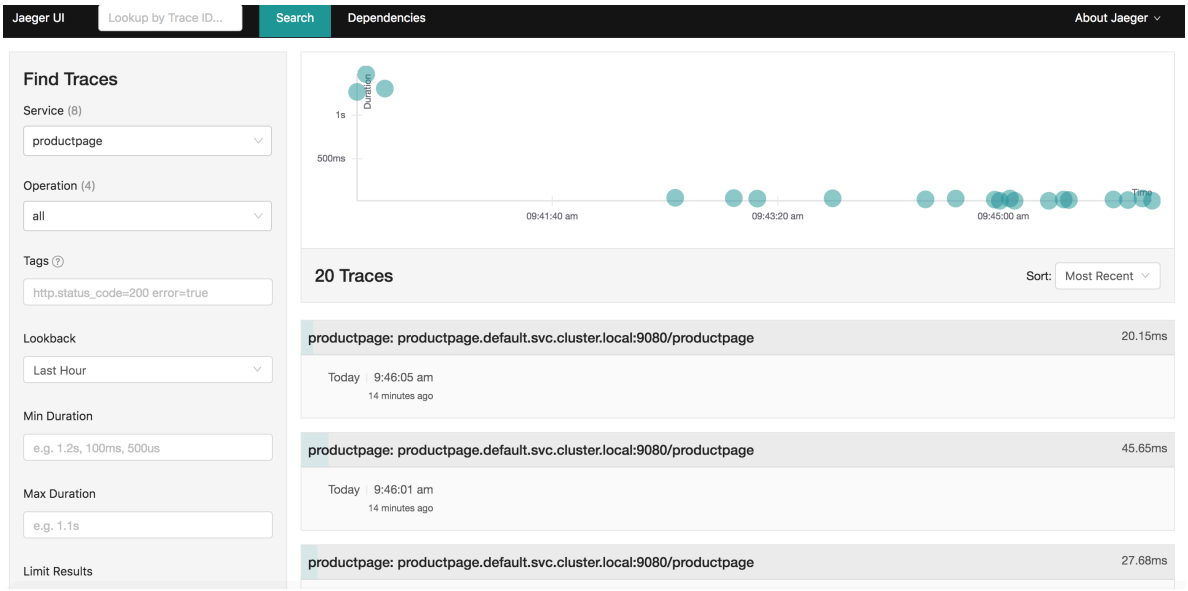
步骤2 查看分布式调用链跟踪Jaeger on Aliyun Log Service

1. 登录容器服务管理控制台。
2. 单击左侧导航栏的应用 > 服务，选择所需集群和istio-system命名空间，单击tracing-on-sls-query服务的外部地址，进入Jaeger UI 页面。

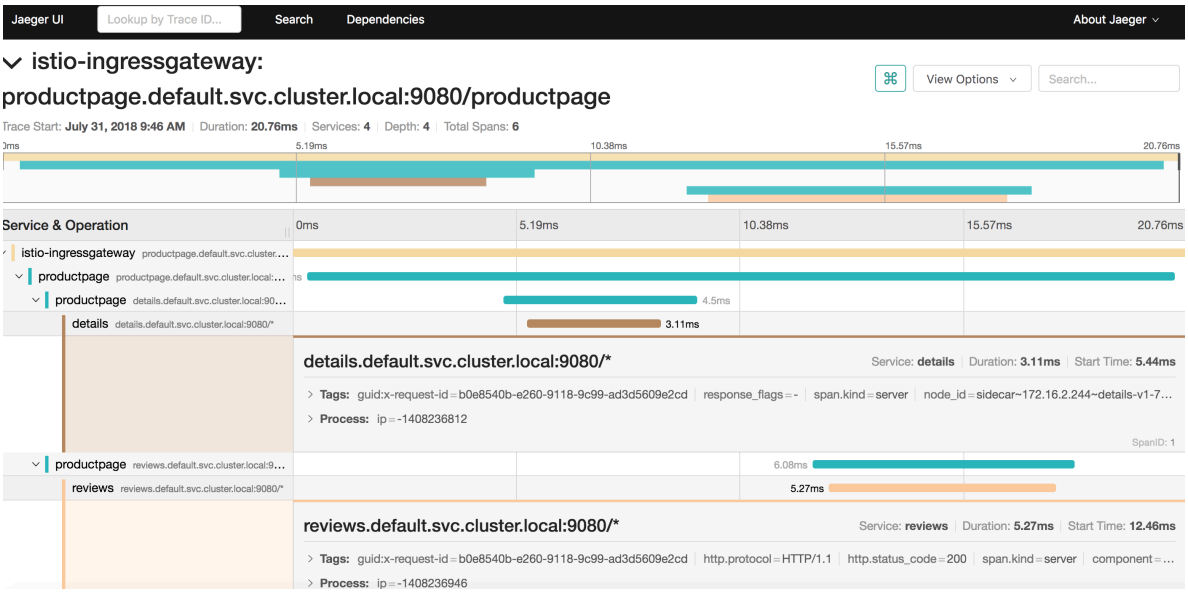


3. 在左侧选择Service，然后选择各项配置，最后单击Find Traces，右上角显示的时刻和持续时间散点图用可视化方式呈现了结果，并提供了向下挖掘能力。

> **说明：**
> 本例中以BookInfo示例应用为例，查看productpage服务的调用情况。



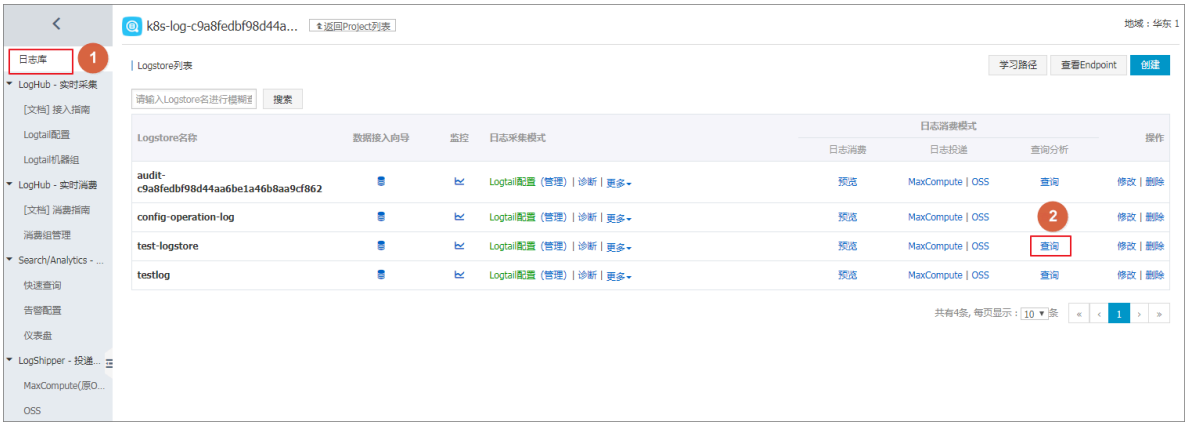
4. 用户可以选择用多种不同视图对追踪结果进行可视化，例如追踪时段内的直方图，或服务在追踪过程中的累积时间。



步骤3 查看阿里云日志服务Log Service

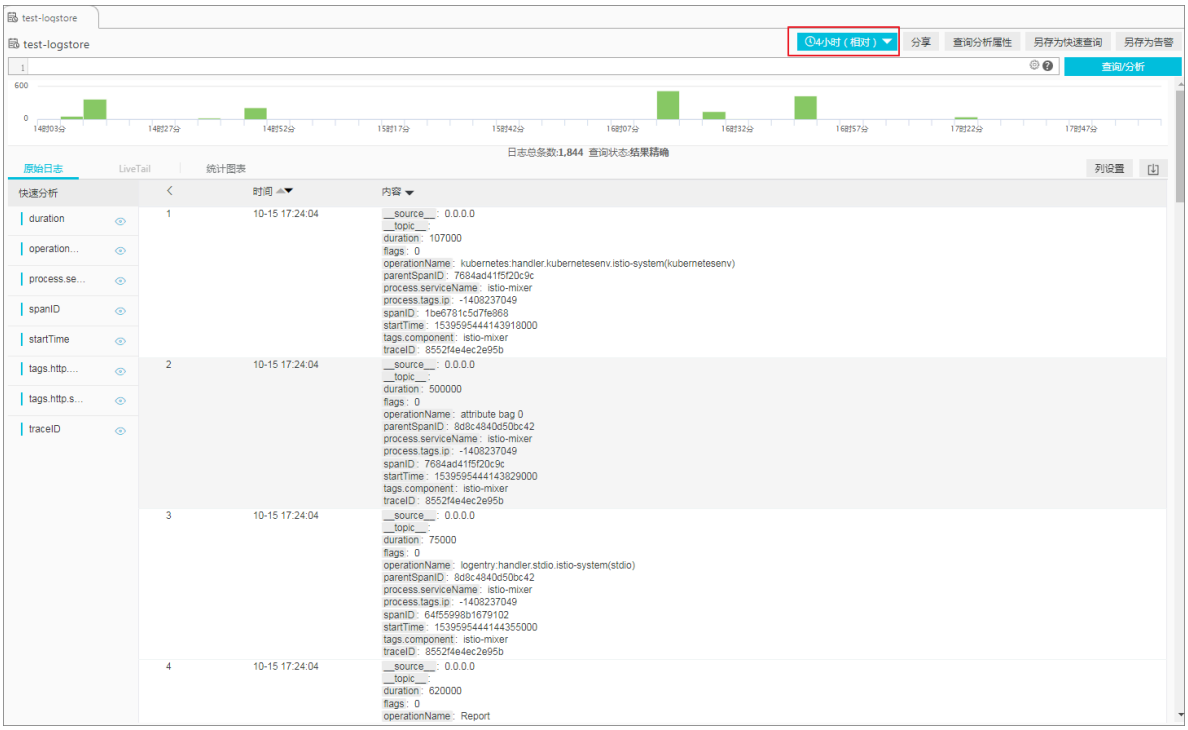
1. 登录 日志服务管理控制台。
2. 单击左侧导航栏中Project管理，选择所需的Project，并单击Project项目名称。

3. 进入Project详情页，单击查询。

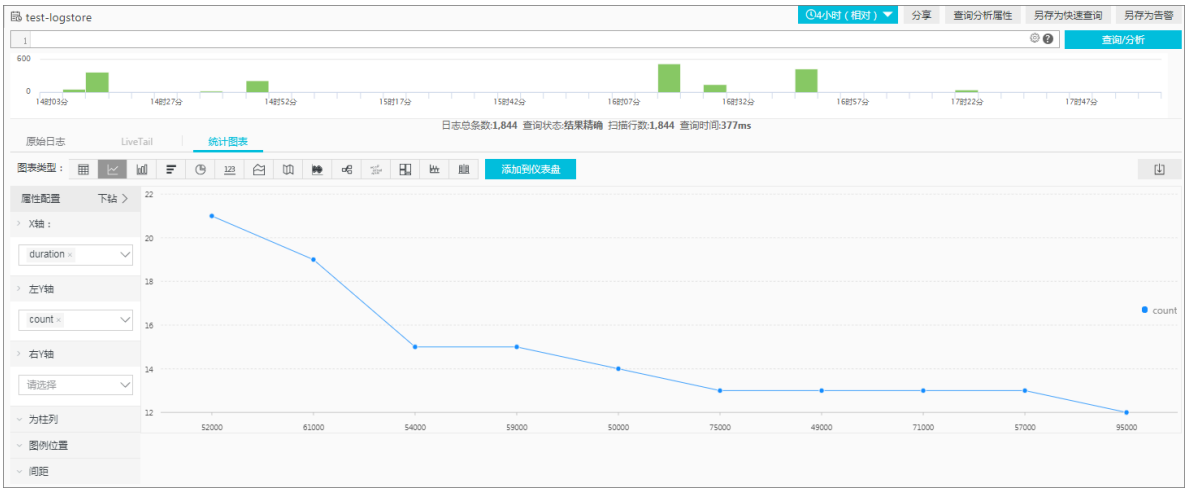


4. 选择日志查询时间，然后单击查询/分析。

日志服务查询分析功能除了提供日志内容的各种语句查询能力以外，还提供多种扩展功能优化您的查询，如支持各种图表功能。



统计图表：



总结

阿里云Kubernetes容器服务已经提供了Istio与日志服务Log Service的集成能力，本文通过一个官方示例来重点介绍了Istio与基于阿里云日志服务的分布式追踪系统的整合能力。

欢迎大家使用阿里云上的容器服务，快速搭建微服务的开放治理平台Istio，比较简单地集成到自己项目的微服务开发中。

5.4 基于Istio实现Kubernetes与ECS上的应用服务混合编排

Istio从0.2开始就提供了一个称之为Mesh Expansion(中文大多称之为网格扩展)的功能。它的主要功能是可以把一些非Kubernetes服务（这些服务往往是运行在其他一些虚拟机或物理裸机中）集成到运行在Kubernetes集群上的Istio服务网格中。

阿里云Kubernetes容器服务已经提供了Istio Mesh Expansion整合能力，本文通过一个官方示例来重点介绍如何使用Istio提供Kubernetes与阿里云ECS上的应用服务混合编排能力。

网格扩展Mesh Expansion

Mesh Expansion就是指部署在 Kubernetes 之中的 Istio 服务网格提供的一种将虚拟机或物理裸机集成进入到服务网格的方法。

Mesh Expansion对于用户从遗留系统往云上迁移过程中有着非常重要的作用，在微服务体系结构中，无法要求所有的工作负载都在Kubernetes中运行，用户的一些应用程序可能在Kubernetes中运维，而另外一些可能在虚拟机或物理裸机中运行。

通过一套Istio控制面板就可以管理跨Kubernetes与虚拟机或物理裸机的若干服务。这样既能保证原有业务的正常运转，又能实现Kubernetes与虚拟机上的应用服务混合编排的能力。

准备Kubernetes集群并安装Istio

阿里云容器服务Kubernetes 1.10.4目前已经上线，可以通过容器服务管理控制台非常方便地快速创建 Kubernetes 集群。具体过程可以参考创建Kubernetes集群。

> 说明:
> 确保通过kubectl 能够连接上Kubernetes 集群，参见通过 kubectl 连接 Kubernetes 集群。

接着，就像前面系列文章中介绍的步骤，通过应用目录简便部署Istio。首先通过命令行或者控制台创建命名空间`istio-system`。

1. 登录容器服务管理控制台。
2. 单击左侧的市场 > 应用目录，在右侧选中`ack-istio`。
3. 在打开的页面中选择命名空间 `istio-system`，并单击参数，可以通过修改参数配置进行定制化安装。

> 说明:
> 说明文档提供了安装和卸载的一些重要信息，特别是常见的CRD（custom resource definition）版本问题。

> **说明:**
> 在使用阿里云Kubernetes容器服务Istio 1.0的过程中，如果遇到类似CRD版本问题，请参考我们提供的 阿里云Kubernetes容器服务Istio实践之常见问题分析。 我们会持续更新遇到的问题及其解决方法。

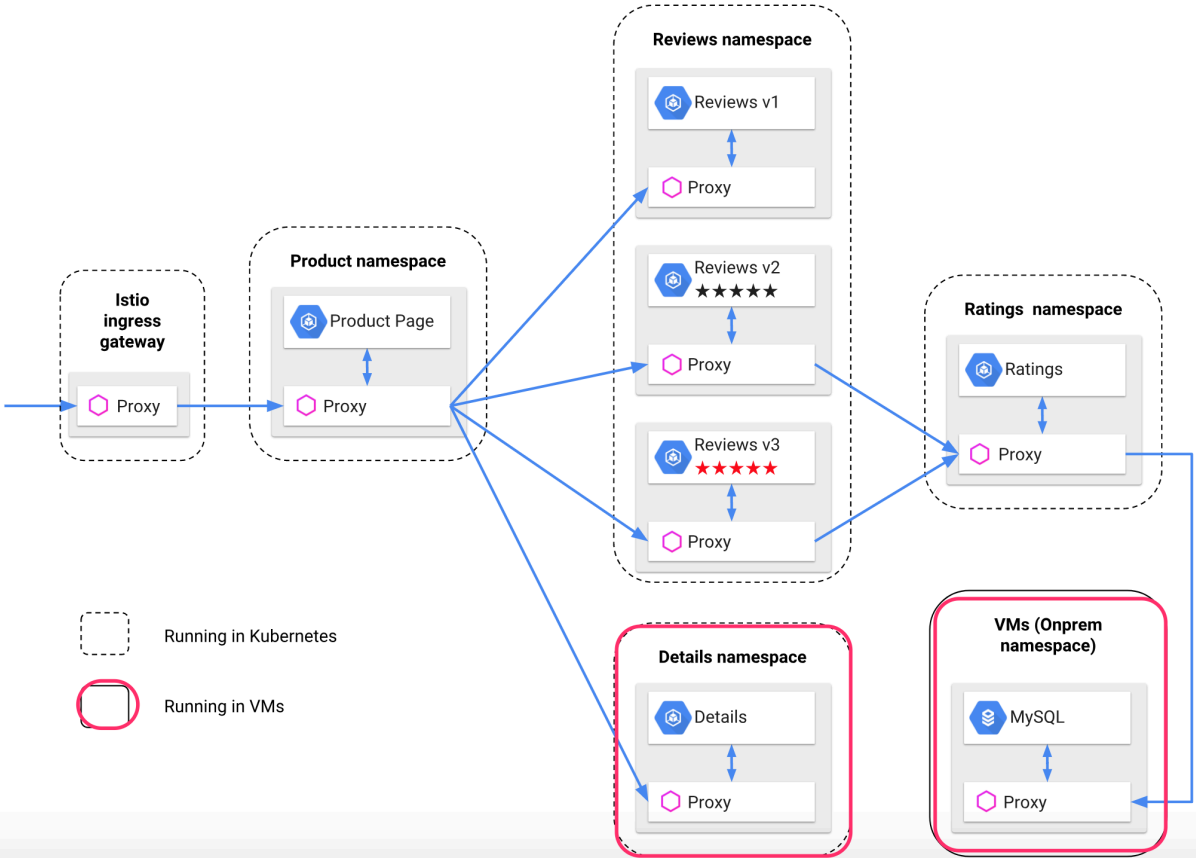
安装示例到Kubernetes集群中

首先通过如下命令行或者控制台创建命名空间 `bookinfo`，并部署我们修改之后的应用。在这个修改后的版本中去掉了`details`组件，并定义了`ingressgateway`。

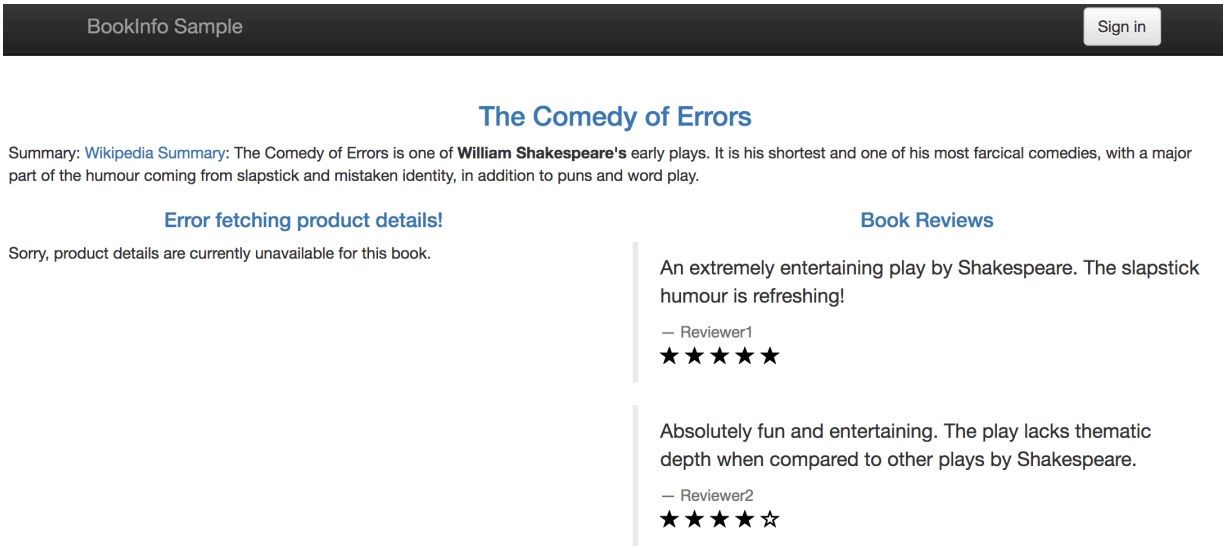
本示例中涉及到的文件可以从示例获取到。

```
kubectl create ns bookinfo
kubectl label namespace bookinfo istio-injection=enabled
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-without-details.yaml
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-gateway.yaml
```

基于官方示例修改的部署，其中`details`组件和数据库运行在Kubernetes之外的ECS上：



正常运行之后，通过`ingressgateway`暴露的地址访问`/productpage`页面，效果应该如下所示，`details`部分应该不能正常显示：



设置 Kubernetes

1. **首先，如果在安装Istio时没有为 Kube DNS、Pilot、Mixer 以及 Citadel 设置内部负载均衡器的话，则需要进行进行这步设置，命令如下：**

```
kubectl apply -f ./mesh-expansion.yaml
```

4个服务创建如下：

```
ali-1c36bbed0b91:meshexpansion wangxn$ kubectl apply -f ./mesh-expansion.yaml
service "istio-pilot-ilb" created
service "dns-ilb" created
service "mixer-ilb" created
service "citadel-ilb" created
```

2. 接着，生成 Istio 的配置*cluster.env*与 DNS 配置文件*kubedns*，用来在虚拟机上进行配置。其中*cluster.env*文件包含了将要拦截的集群 IP 范围，*kubedns*文件则是让虚拟机上的应用能够解析集群的服务名称，然后被 Sidecar 劫持和转发。

   执行命令如下：

   ```
   ./setupMeshEx.sh generateClusterEnvAndDnsmasq
   ```

   生成的*cluster.env*配置文件的示例：

   ```
   ali-1c36bbed0b91:meshexpansion wangxn$ cat cluster.env
   ISTIO_SERVICE_CIDR=
   ISTIO_SYSTEM_NAMESPACE=istio-system
   ISTIO_CP_AUTH=MUTUAL_TLS
   ```

   生成的*kubedns*文件的示例：

   ```
   ali-1c36bbed0b91:meshexpansion wangxn$ cat kubedns
   server=/svc.cluster.local/
   address=/istio-policy/
   address=/istio-telemetry/
   address=/istio-pilot/
   address=/istio-citadel/
   address=/istio-ca/
   address=/istio-policy.istio-system/
   address=/istio-telemetry.istio-system/
   address=/istio-pilot.istio-system/
   address=/istio-citadel.istio-system/
   address=/istio-ca.istio-system/
   ```

设置ECS

配置你自己的工作环境能与ECS实例的授权，生成SSH key并分发到ECS中。可以通过`ssh root@<ECS_HOST_IP>`命令确认是否可以成功连上ECS虚机。

生成公钥：

```
ssh-keygen -b 4096 -f ~/.ssh/id_rsa -N ""
```

说明：

为保证ECS能与Kubernetes网络上可通，可以将ECS与Kubernetes加入到同一个安全组。

阿里云容器服务针对ECS的设置步骤提供了较好的用户体验，通过运行以下脚本即可完成配置：

```
export SERVICE_NAMESPACE=default
```

```
./setupMeshEx.sh machineSetup root@<ECS_HOST_IP>
```

检查运行的进程：

```
ps aux |grep istio
```

```
root@remotevm:~# ps aux |grep istio
root     19460  0.0  0.0  52284  3404 ?        Ss   11:59   0:00 su -s /bin/bash -c INSTANCE_IP=192.168.3.70 POD_NAME=remotevm POD_NAMESPACE=default exec /u
sr/local/bin/pilot-agent proxy      --serviceCluster rawvm     --discoveryAddress istio-pilot.istio-system:8080     --controlPlaneAuthPolicy MUTUAL_TLS
2> /var/log/istio/istio.err.log > /var/log/istio/istio.log istio-proxy
istio-p+ 19508  0.0  0.0  45276  4592 ?        Ss   11:59   0:00 /lib/systemd/systemd --user
istio-p+ 19510  0.0  0.0  61324  2064 ?        S    11:59   0:00 (sd-pam)
istio-p+ 19516  0.0  0.2  31204 17252 ?        Ssl  11:59   0:00 /usr/local/bin/pilot-agent proxy --serviceCluster rawvm --discoveryAddress istio-pilot.isti
o-system:8080 --controlPlaneAuthPolicy MUTUAL_TLS
istio-p+ 19537  7.1  0.5 133208 41572 ?        Sl   11:59   0:02 /usr/local/bin/envoy -c /etc/istio/proxy/envoy-rev1.json --restart-epoch 1 --drain-time-s 2
 --parent-shutdown-time-s 3 --service-cluster rawvm --service-node sidecar~192.168.3.70~remotevm.default~default.svc.cluster.local --max-obj-name-len 189 -l
 warn --v2-config-only
```

Istio 认证使用的 Node Agent 健康运行：

```
sudo systemctl status istio-auth-node-agent
```

在ECS上运行务

由前面示例部署图知道，有2个服务需要运行在ECS上，一个是Details服务，另一个是数据库服务。

在ECS上运行Details服务

通过以下命令模拟(仅仅是用Docker模拟而已)一个`Details`服务，运行在ECS上并暴露端口9080。

```
docker pull istio/examples-bookinfo-details-v1:1.8.0
docker run -d -p 9080:9080 --name details-on-vm istio/examples-
bookinfo-details-v1:1.8.0
```

配置 Sidecar 来拦截端口，这一配置存在于*/var/lib/istio/envoy/sidecar.env*，使用环境变量`ISTIO_INBOUND_PORTS`。

示例 (在运行服务的虚拟机上)：

```
echo"ISTIO_INBOUND_PORTS=9080,8080" > /var/lib/istio/envoy/sidecar.env
```

```
systemctl restart istio
```

注册Details服务到Istio

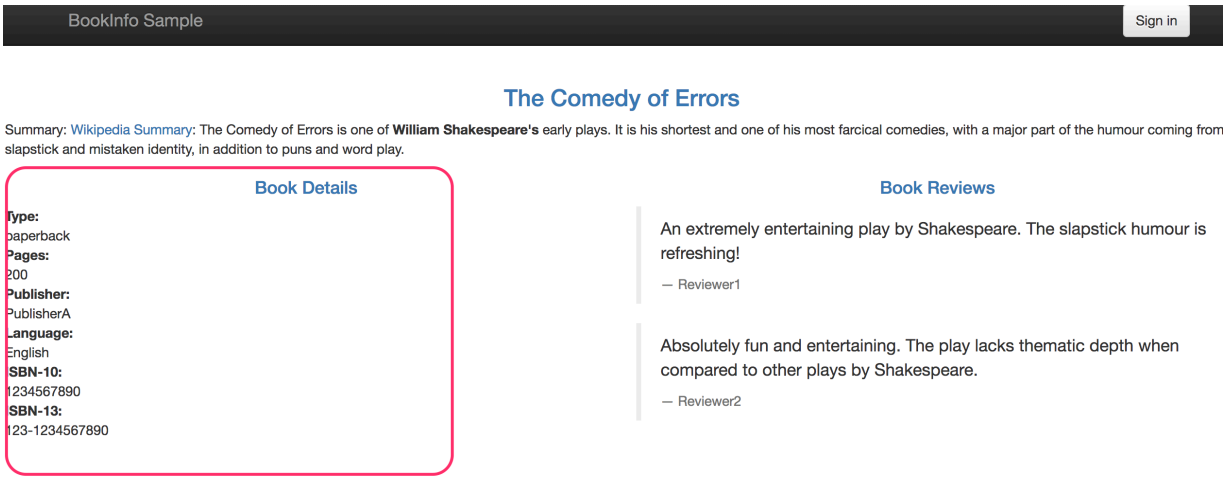
查找虚拟机的 IP 地址，用来加入服务网格：

```
hostname -I
```

手工配置一个没有选择器的服务和端点，用来承载没有对应 Kubernetes Pod 的服务。例如在一个有权限且能够使用 istioctl 命令的服务器上，注册`Details`服务：

```
istioctl -n bookinfo register details 192.168.3.202 http:9080
```

再次访问`/productpage`页面，效果应该如下所示，`details`部分应该可以正常显示

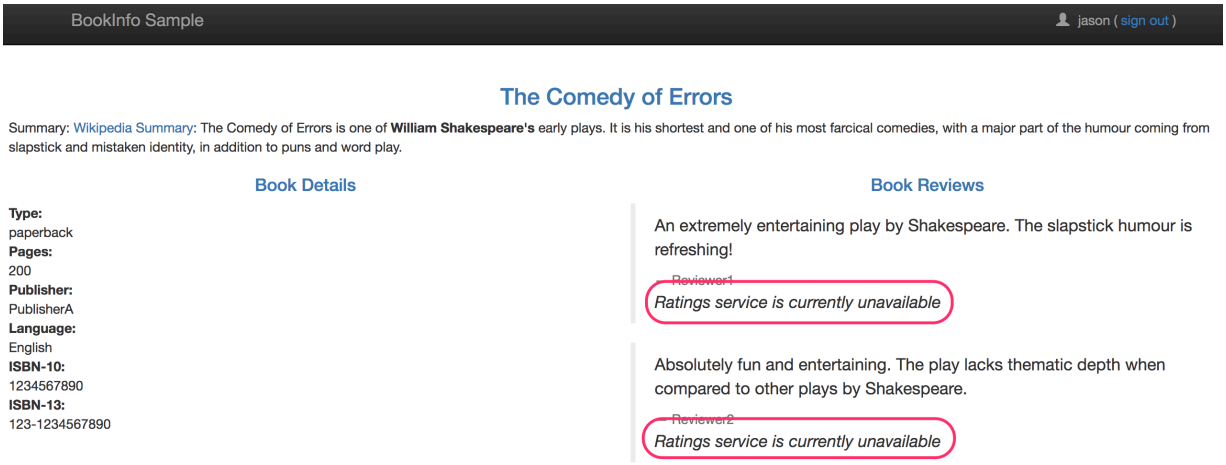


将ratings服务切到数据库版本

默认ratings服务不访问数据库，通过如下命令更新版本，使得ratings服务切换到访问数据库的版本：

```
kubectl apply -f ./bookinfo/bookinfo-ratings-v2-mysql-vm.yaml
kubectl apply -f ./bookinfo/virtual-service-ratings-mysql-vm.yaml
```

此时访问`/productpage`页面，效果应该如下所示，`ratings`部分不能正常显示，下一步就是在ECS上搭建数据库服务，并将之加入到Istio中：



在ECS上运行数据库服务

在虚拟机上运行 MariaDB，将其作为 ratings 服务的后端；并配置MariaDB可以支持远程访问。

```
apt-get update && apt-getinstall -y mariadb-server
sed -i 's/127\.0\.0\.1/0\.0\.0\.0/g' /etc/mysql/mariadb.conf.d/50-
server.cnf
sudo mysql
# 授予 root 权限
GRANT ALL PRIVILEGESON *.* TO'root'@'localhost'IDENTIFIEDBY'password'
WITHGRANTOPTION;
quit;
sudo systemctl restart mysql
```

在虚拟机上把初始化 ratings 数据库。

```
curl -q https://raw.githubusercontent.com/istio/istio/master/samples/
bookinfo/src/mysql/mysqldb-init.sql | mysql -u root -ppassword
```

为了更清晰的观察 Bookinfo 应用在输出方面的差异，可以用下面的命令来修改评级记录，从而生成不同的评级显示：

```
mysql -u root -ppassword test -e "select * from ratings;"
mysql -u root -ppassword test -e  "update ratings set rating=2;select
 * from ratings;"
```

注册数据库服务到Istio

配置 Sidecar 来拦截端口，这一配置存在于*/var/lib/istio/envoy/sidecar.env*，使用环境变量ISTIO_INBOUND_PORTS 。

示例 (在运行服务的虚拟机上):

```
echo"ISTIO_INBOUND_PORTS=3306,9080,8080" > /var/lib/istio/envoy/
sidecar.env
```

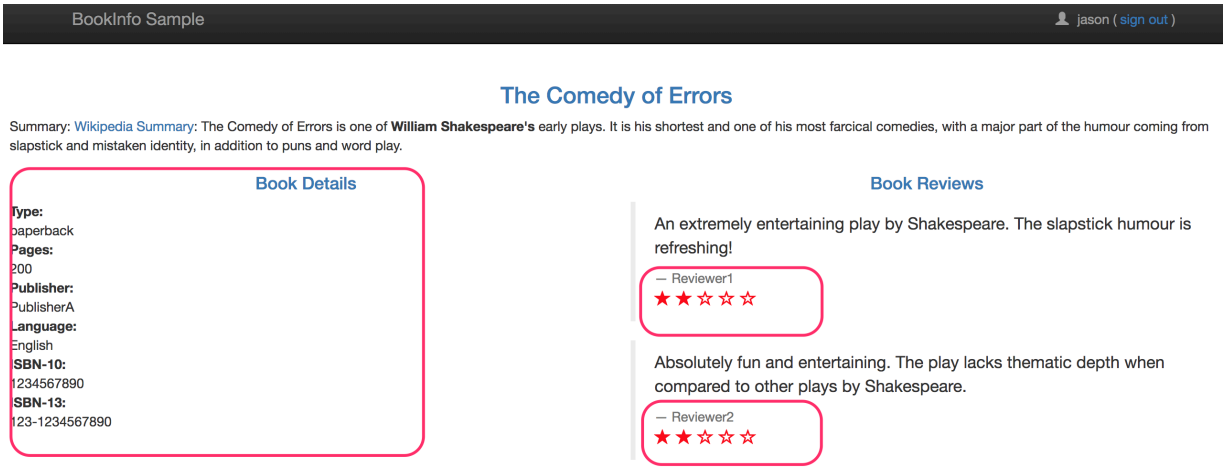
```
systemctl restart istio
```

同样地，在一个有权限且能够使用 istioctl 命令的服务器上，注册数据库服务：

```
istioctl-nbookinforegistermysqldb 192.168.3.202 3306
```

经过这个步骤，Kubernetes Pod 和其他网格扩展包含的服务器就可以访问运行于这一服务器上的数据库服务了。

此时访问/productpage页面，效果应该如下所示，details 与 ratings 部分均能正常显示，而且这2个服务都是来自于ECS：



总结

阿里云Kubernetes容器服务已经提供了Istio Mesh Expansion整合能力，本文通过一个官方示例来重点介绍如何使用Istio打通Kubernetes与阿里云ECS上的应用服务混合编排能力。

欢迎大家使用阿里云上的容器服务，快速搭建微服务的开放治理平台Istio，比较简单地集成到自己项目的微服务开发中。

5.5 基于Istio实现多Kubernetes集群上的应用服务混合编排

Istio 一个重要目标是支持混合型环境，例如您可以将应用运行在阿里云容器服务、本地 Kubernetes 集群，或是其他公有云的服务上。Istio 能够为整体服务平台提供一个统一的视图，管理这些环境之间的连接并确保安全性。Istio多集群支持的架构，允许将多个 Kubernetes 集群加入一个单独的服务网格（Service Mesh）中，并启用跨集群的服务发现功能。按照官方介绍，它还将支持全球化的集群级别的负载均衡，并通过 Gateway 对等互联提供对非扁平网络的支持。

在基于Istio实现Kubernetes与ECS上的应用服务混合编排中介绍阿里云Kubernetes容器服务提供的Istio Mesh Expansion整合能力。

本文通过一个官方示例来重点介绍在阿里云容器服务上如何基于Istio实现多Kubernetes集群上的应用服务混合编排。你可以看到在一个 Kubernetes 集群上安装 Istio 控制平面，然后Istio连接了2个 Kubernetes 集群之后，就会生成一个跨越多个 Kubernetes 集群的网格网络。

准备Kubernetes集群

阿里云容器服务Kubernetes 1.10.4目前已经上线，可以通过容器服务管理控制台非常方便地快速创建 Kubernetes 集群。具体过程可以参考创建Kubernetes集群。

确保安装配置kubectl 能够连接上Kubernetes 集群，并满足以下网络要求：

- 各集群的 Pod CIDR 范围和 Service CIDR 范围必须是唯一的，不允许相互重叠。
- 每个集群中的所有的 Pod CIDR 需要能够互相路由。
- 所有的 Kubernetes 控制平面 API Server 互相可路由。

集群	ContainerCIDR	ServiceCIDR
Istio-control-plane	172.16.0.0/16	172.19.0.0/20
Istio-remote1	172.20.0.0/16	172.21.0.0/20

本示例中涉及到的文件可以从示例 获取。

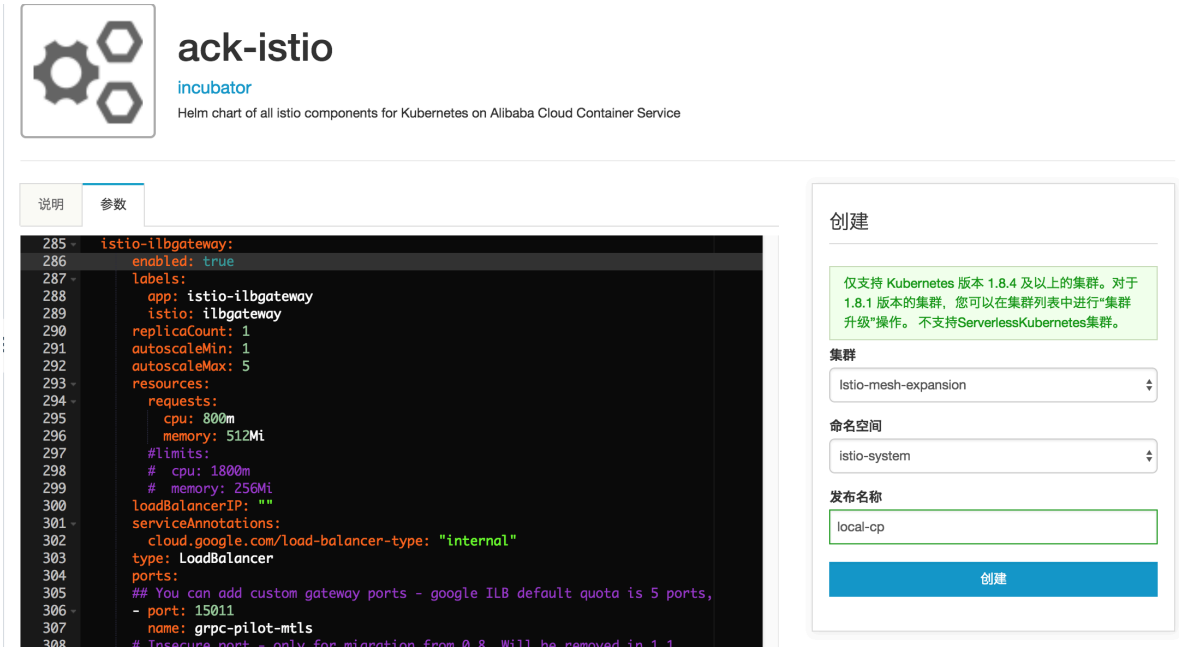
安装Istio控制平面

接着，就像前面系列文章中介绍的步骤，通过应用目录简便部署Istio。

1. 登录容器服务管理控制台。
2. 单击左侧的市场 > 应用目录，在右侧选中ack-istio。
3. 在打开的页面中选择命名空间istio-system，并单击参数，可以通过修改参数配置进行定制化安装：

```
......
   istio-ilbgateway:
      enabled: true
```

```
.....
```



说明:

说明文档提供了安装和卸载的一些重要信息，特别是常见的CRD（custom resource definition）版本问题。

说明:

在使用阿里云Kubernetes容器服务Istio 1.0的过程中，如果遇到类似CRD版本问题，请参考我们提供的 阿里云Kubernetes容器服务Istio实践之常见问题分析 。 我们会持续更新遇到的问题及其解决方法。

安装 Istio 远程组件

通过以下脚本可以获取到控制平面的连接信息：

```
export PILOT_POD_IP=$(kubectl -n istio-system get pod -l istio=pilot -
o jsonpath='{.items[0].status.podIP}')
export POLICY_POD_IP=$(kubectl -n istio-system get pod -l istio=mixer
 -o jsonpath='{.items[0].status.podIP}')
export STATSD_POD_IP=$(kubectl -n istio-system get pod -l istio=statsd
-prom-bridge -o jsonpath='{.items[0].status.podIP}')
export TELEMETRY_POD_IP=$(kubectl -n istio-system get pod -l istio-
mixer-type=telemetry -o jsonpath='{.items[0].status.podIP}')
export ZIPKIN_POD_IP=$(kubectl -n istio-system get pod -l app=jaeger -
o jsonpath='{.items[0].status.podI
P}')
echo "remotePilotAddress: $PILOT_POD_IP"
echo "remotePolicyAddress: $POLICY_POD_IP"
echo "remoteStatsdPromBridge: $STATSD_POD_IP"
echo "remoteTelemetryAddress: $TELEMETRY_POD_IP"
```

```
echo "remoteZipkinAddress: $ZIPKIN_POD_IP"
```

1. 登录容器服务管理控制台。
2. 单击左侧的市场 > 应用目录，在右侧选中ack-istio-remote。
3. 在打开的页面中选择命名空间istio-system，并单击参数，将这些信息填充到参数配置中进行定制化安装：

```
  .....
  envoyStatsd:
    enabled: false
    host: "$remoteStatsdPromBridge"
```

```
# Remote Istio endpoints. Can be hostnames or IP addresses# The
Pilot address is required. The others are optional.
remotePilotAddress: "$remotePilotAddress"
remotePolicyAddress: "$remotePolicyAddress"
remoteTelemetryAddress: "$remoteTelemetryAddress"
remoteZipkinAddress: "$remoteZipkinAddress"
```

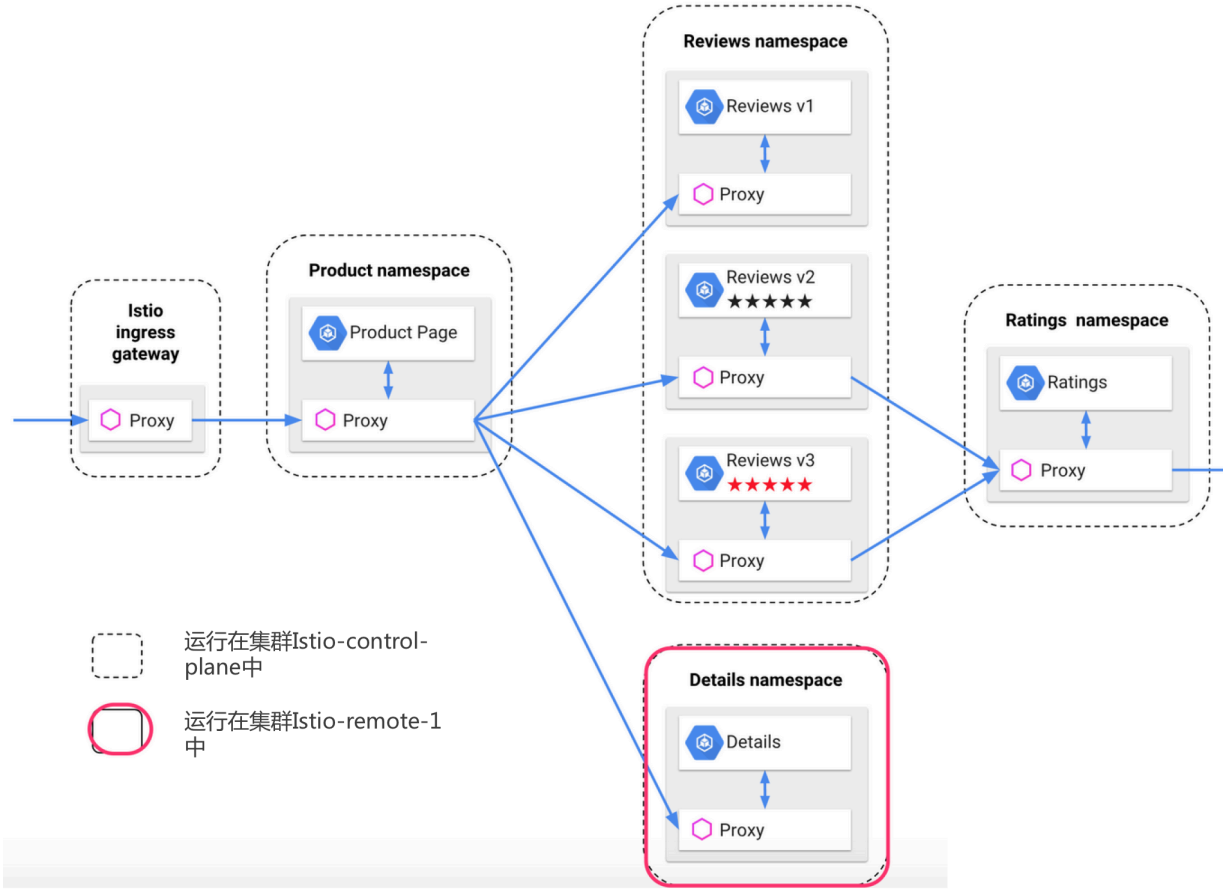
安装示例到集群Istio-control-plane

首先通过如下命令行或者控制台创建命名空间 bookinfo，并部署我们修改之后的应用。在这个修改后的版本中去掉了details组件，并定义了ingressgateway 。

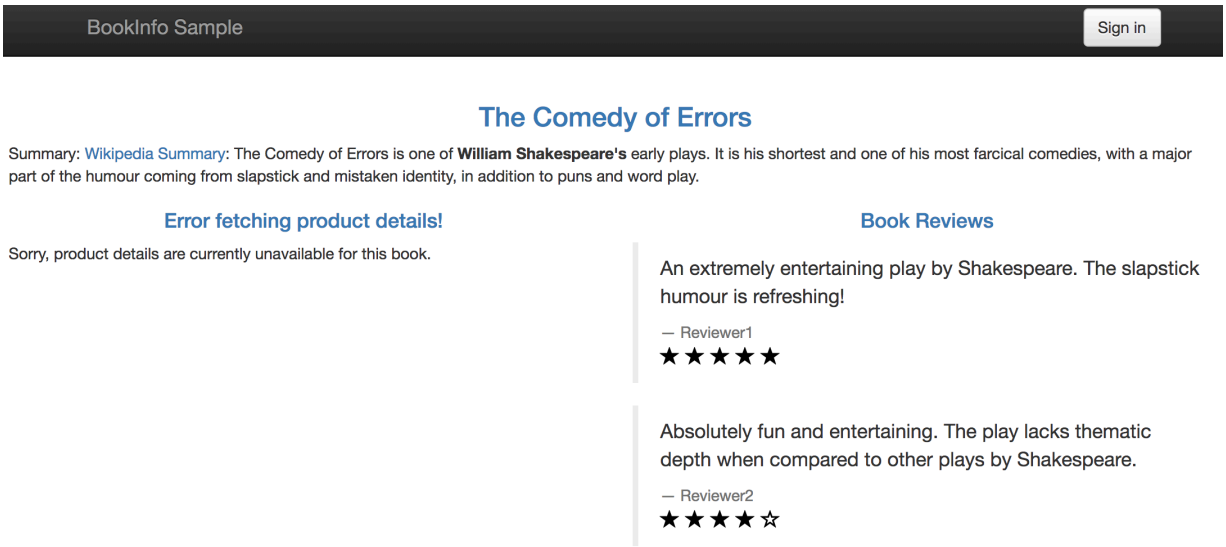
本示例中涉及到的文件可以从 这里 获取到。

```
kubectl create ns bookinfo

kubectl label namespace bookinfo istio-injection=enabled
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-without-details.yaml
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-gateway.yaml
```

基于官方示例修改的部署，其中除了details组件运行在安装Istio本地控制平面的Kubernetes集群Istio-control-plane中，而details组件运行在另外一个Kubernetes集群Istio-remote-1中：



正常运行之后，通过`ingressgateway`暴露的地址访问`/productpage`页面，效果应该如下所示，`details`部分应该不能正常显示（因为此时`details`组件还没有安装到集群Istio-remote-1并加入到同一网格中）：



远程集群配置

Istio 控制平面需要访问网格中的所有集群，来完成服务发现的目的。下面描述了如何在远程集群中创建一个 Service account，并赋予它必要的 RBAC 权限；后面还会使用这个 Service account 的凭据为远程集群生成一个 kubeconfig 文件，这样就可以访问远程集群了。

下面的过程应该在每一个要加入到服务网格中的集群上执行。这个过程需要对应集群的管理员用户来完成。运行命令`generate-kubeconfig.sh`，参数为描述所在集群的唯一标示：

```
./generate-kubeconfig.sh myremote1
```

完成这些步骤之后，就在当前目录中创建了远程集群的 kubeconfig 文件。集群的文件名和原始的 kubeconfig 集群名称一致。

控制平面集群配置

在运行 Istio 控制平面的集群上为每个远程集群创建一个 Secret：

```
./create-secret.sh myremote1
```

安装示例到集群Istio-remote-1

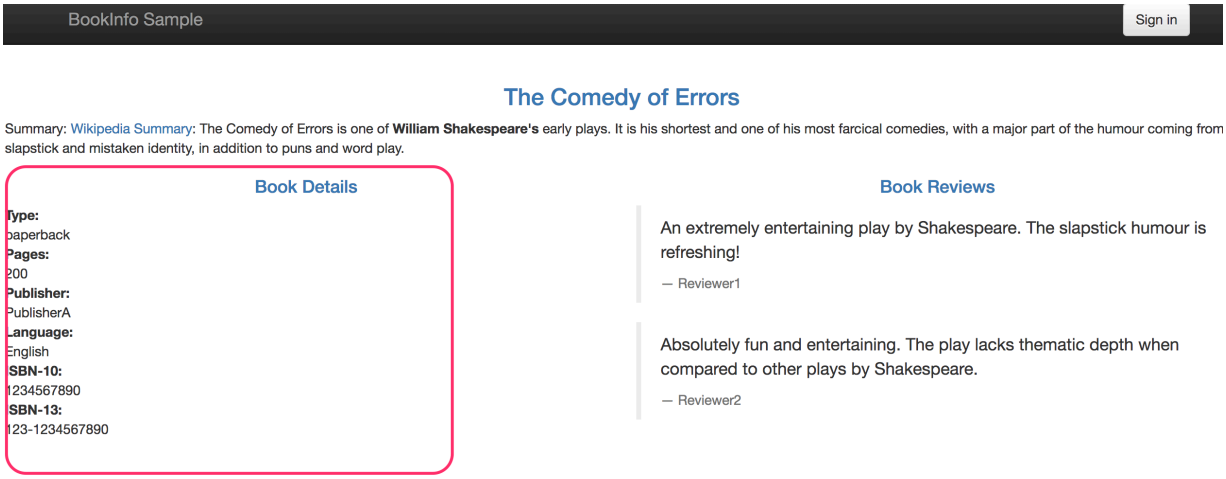
同样的步骤，安装示例中`details`部分到集群Istio-remote-1，YAML文件可以从 示例文件 获取到。

```
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-details.yaml
```

之后，`details`服务将会被注册到控制平面中，可以查看`{pilot-ipAddress}:8080/v1/registration`返回结果是否包含 `details`服务：

```
{
 "service-key": "details.bookinfo.svc.cluster.local|http",
 "hosts": [
  {
   "ip_address": "[redacted]",
   "port": 9080
  }
 ]
},
```

再次访问`/productpage`页面，效果应该如下所示，`details`部分应该可以正常显示：



总结

本文通过一个官方示例，来重点介绍在阿里云容器服务上如何基于Istio实现多Kubernetes集群上的应用服务混合编排。你可以看到在 一个 Kubernetes 集群上安装 Istio 控制平面，然后Istio连接了2个 Kubernetes 集群之后，就会生成一个跨越多个 Kubernetes 集群的网格网络。

欢迎大家使用阿里云上的容器服务，快速搭建微服务的开放治理平台Istio，比较简单地集成到自己项目的微服务开发中。

6 监控

6.1 Kubernetes基于ARMS的应用监控

本文介绍容器服务Kubernetes基于业务实时监控服务ARMS（Application Real-Time Monitoring Service）如何进行应用监控。

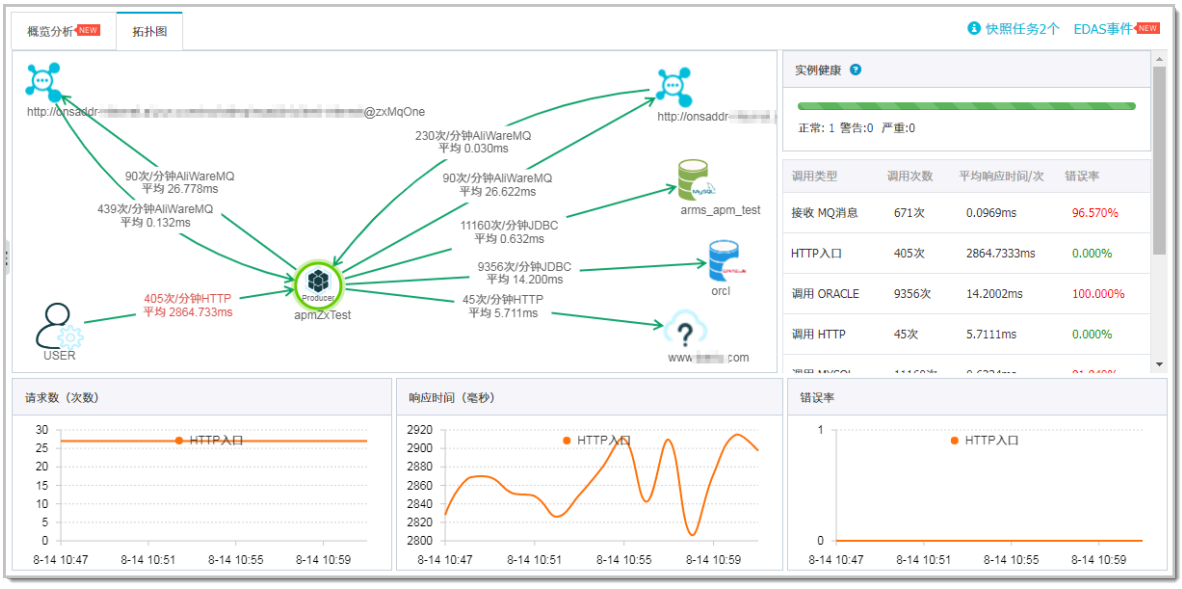
背景信息

业务实时监控服务ARMS（Application Real-Time Monitoring Service）是一款阿里云应用性能管理（APM）类监控产品。

ARMS应用监控是一款针对 Java 应用的性能管理（Application Performance Management，简称 APM）软件。您无需修改任何代码，只需要在 Java 应用的启动脚本中挂载一个探针，该探针就能够对您的 Java 应用进行全方位监控，帮助您更快速地定位出错接口和慢接口、重现调用参数、检测内存泄漏、发现系统瓶颈，从而大幅提升线上问题诊断问题的效率。ARMS产品详情，可参考业务实时监控服务 ARMS。

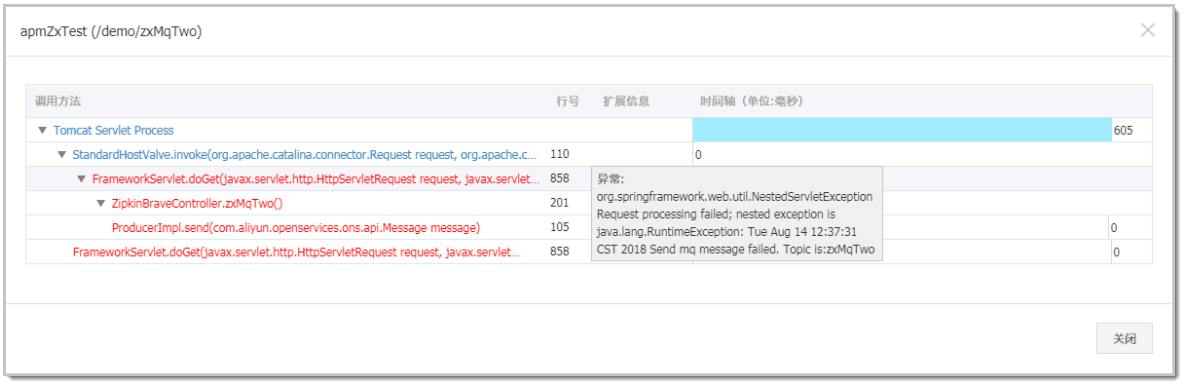
主要功能

- 自动发现应用拓扑



- 自动发现并监控接口
- 捕获异常事务和SQL分析、慢事务
- Java异常报表

- 查询基于调用链的事务快照



- 即席多维排查（多维度调用链搜索，异常调用链搜索）
- PaaS平台集成

前提条件

- 您已成功创建一个Kubernetes集群，参见创建Kubernetes集群。
- 您已开通ARMS应用监控服务，参见开通 ARMS 服务。

安装组件

1. 登录容器服务管理控制台。
2. 单击左侧导航栏市场 > 应用目录，在应用目录页面选择ack-arms-pilot。
3. 在应用目录-ack-arms-pilot页面，单击右侧的创建。

单击左侧导航栏应用 > 无状态，选择目标集群和命名空间，可以看到名称为*ack-arms-pilot-default-ack-arms-pilot*的应用已成功创建。

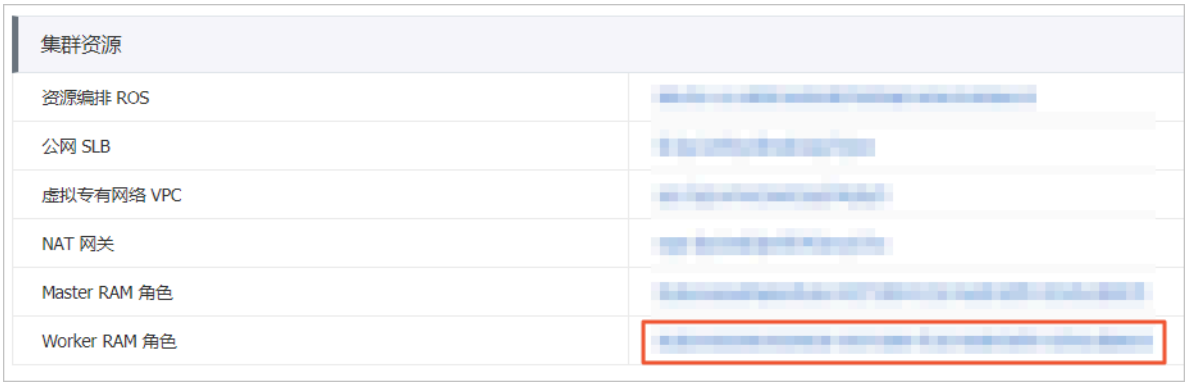
授权

1. 在容器服务管理控制台，单击左侧导航栏集群，进入集群列表页面。

   说明:

   请以主账号登录容器服务管理控制台进行授权操作。

2. 单击目标集群的名称，查看集群的详细信息。
3. 单击集群资源区域的Worker RAM角色，进入RAM访问控制台的RAM角色管理页面。



说明:

本文以新版RAM访问控制台为例进行介绍。

若您使用的是旧版的RAM访问控制台：

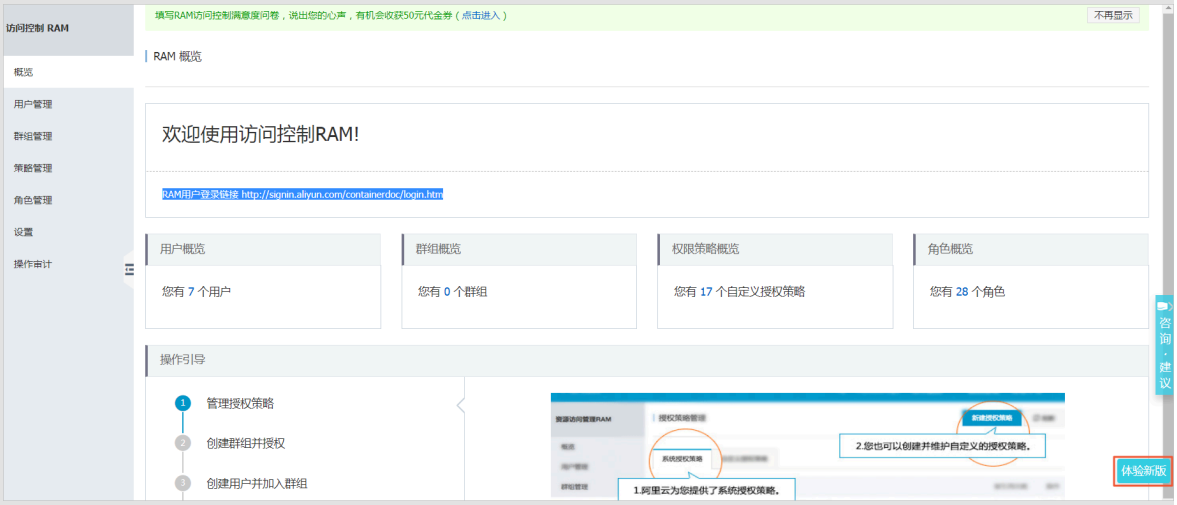
方法一

a. 单击左侧导航栏角色管理，在角色名中输入Worker RAM角色的名称进行搜索。单击角色名称。

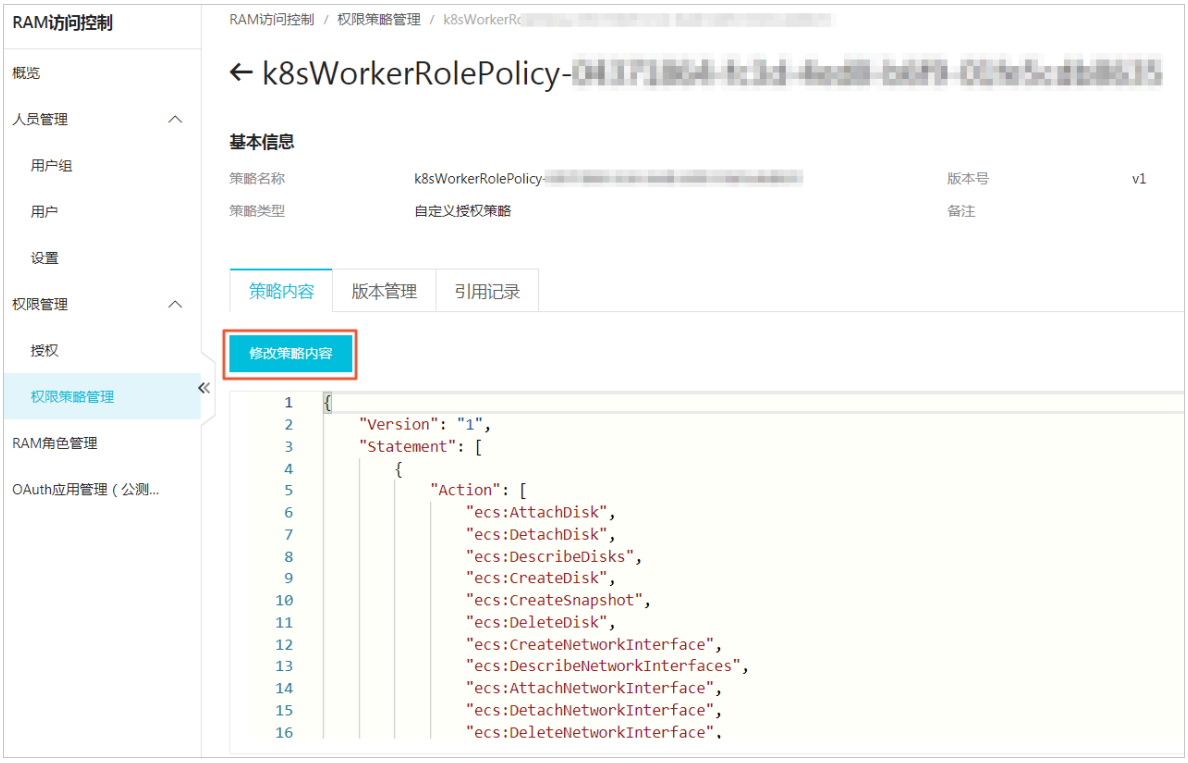
b. 在基本信息区域，单击页面右上角的编辑基本信息。

方法二

单击页面右下角体验新版，切换到新版RAM访问控制台。在容器服务管理控制台单击Worker RAM 角色重新登录RAM访问控制台。



4. **在RAM角色管理页面，单击权限管理区域的权限策略名称，查看具体的权限策略。**
5. **在权限策略管理页面，单击策略内容区域的修改策略内容。**



6. **在策略内容区域增加以下字段后，单击确定。**

```
{
    "Action": "arms:*",
    "Resource": "*",
    "Effect": "Allow"
```

```
}
```

部署ARMS应用监控

在创建Deployment的yaml文件中，通过添加以下`annotations`部署ARMS应用监控。

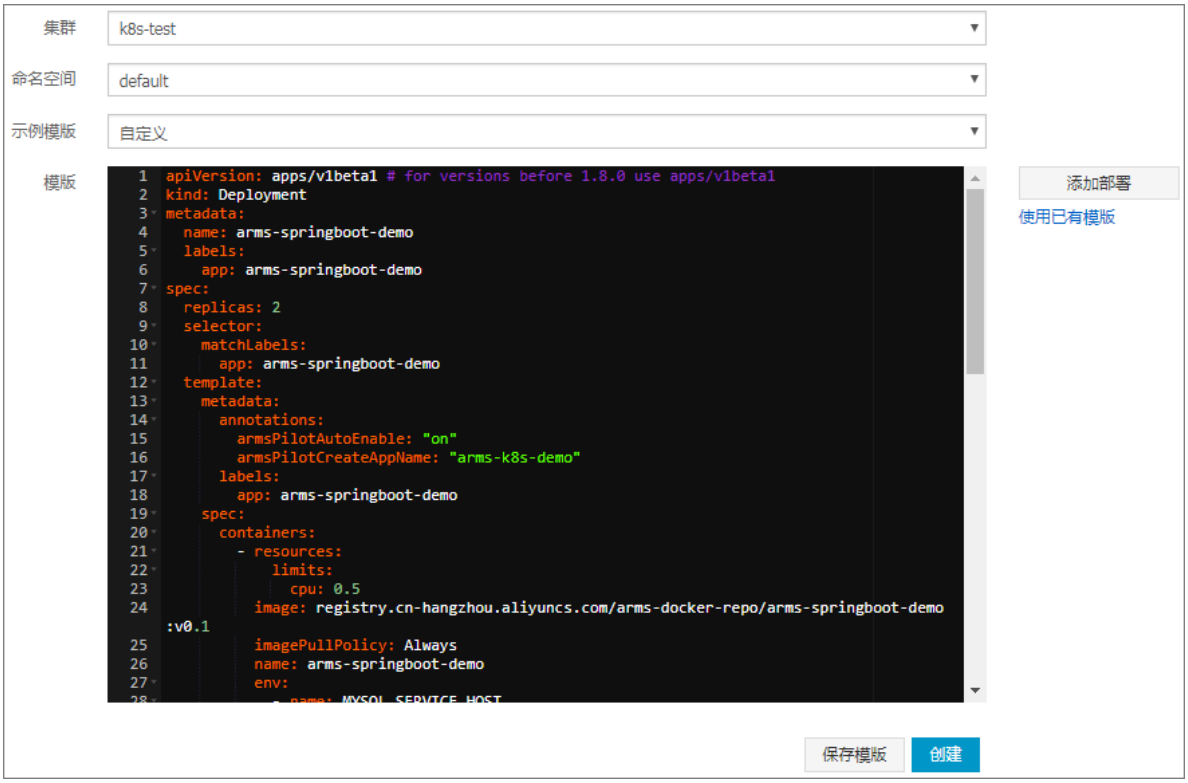
```
annotations:
  armsPilotAutoEnable: "on"
  armsPilotCreateAppName: "<your-deployment-name>"
```

说明:

- `annotations`添加到yaml文件`spec`字段`template`的`metadata`下。

- armsPilotCreateAppName的内容为ARMS中应用的名称。

1. 在容器服务管理控制台，单击左侧导航栏应用 > 无状态。
2. 单击页面右上角使用模板创建。
3. 选择目标集群和命名空间，创建Deployment。




```
apiVersion: apps/v1beta1 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: arms-springboot-demo
  labels:
    app: arms-springboot-demo
spec:
  replicas: 2
  selector:
    matchLabels:
      app: arms-springboot-demo
  template:
    metadata:
      annotations:
        armsPilotAutoEnable: "on"
        armsPilotCreateAppName: "arms-k8s-demo"
      labels:
        app: arms-springboot-demo
    spec:
      containers:
        - resources:
            limits:
              cpu: 0.5
          image: registry.cn-hangzhou.aliyuncs.com/arms-docker-repo/
arms-springboot-demo:v0.1
          imagePullPolicy: Always
          name: arms-springboot-demo
```

```
          env:
            - name: MYSQL_SERVICE_HOST
              value: "arms-demo-mysql"
            - name: MYSQL_SERVICE_PORT
              value: "3306"
---
apiVersion: apps/v1beta1 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: arms-demo-mysql
  labels:
    app: mysql
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - resources:
            limits:
              cpu: 0.5
          image: registry.cn-hangzhou.aliyuncs.com/arms-docker-repo/
arms-demo-mysql:v0.1
          name: mysql
          ports:
            - containerPort: 3306
              name: mysql
---
apiVersion: v1
kind: Service
metadata:
  labels:
    name: mysql
  name: arms-demo-mysql
spec:
  ports:
    # the port that this service should serve on
    - name: arms-mysql-svc
      port: 3306
      targetPort: 3306
  # label keys and values that must match in order to receive
traffic for this service
  selector:
    app: mysql
---
```

执行结果

1. **单击左侧导航栏应用 > 无状态，选择目标集群和命名空间，可看到刚刚部署的Deployment。**
2. **单击目标Deployment操作列的ARMS控制台，登录ARMS管理控制台查看应用的详细信息，例如：应用健康概览、接口调用等。**

说明:

若操作列无ARMS控制台，请检查您是否授权容器服务调用ARMS，可参考授权。

7 DevOps

7.1 通过在Eclipse中安装Alibaba Cloud Toolkit插件部署应用

本文档将向您介绍如何在Eclipse中安装Cloud Toolkit，并使用Cloud Toolkit快速部署一个应用。

背景信息

Alibaba Cloud Toolkit for Eclipse（下文简称 Cloud Toolkit）是一个免费的 IDE 插件，帮助阿里云用户更高效的使用阿里云。

您只需注册或使用一个已有的阿里云账号，即可免费下载Cloud Toolkit。下载完成后，您可以将该插件安装到Eclipse中。

当您在本地完成应用程序的开发、调试及测试后，通过该插件即可轻松将应用程序部署到阿里云。

安装Cloud Toolkit

前提条件

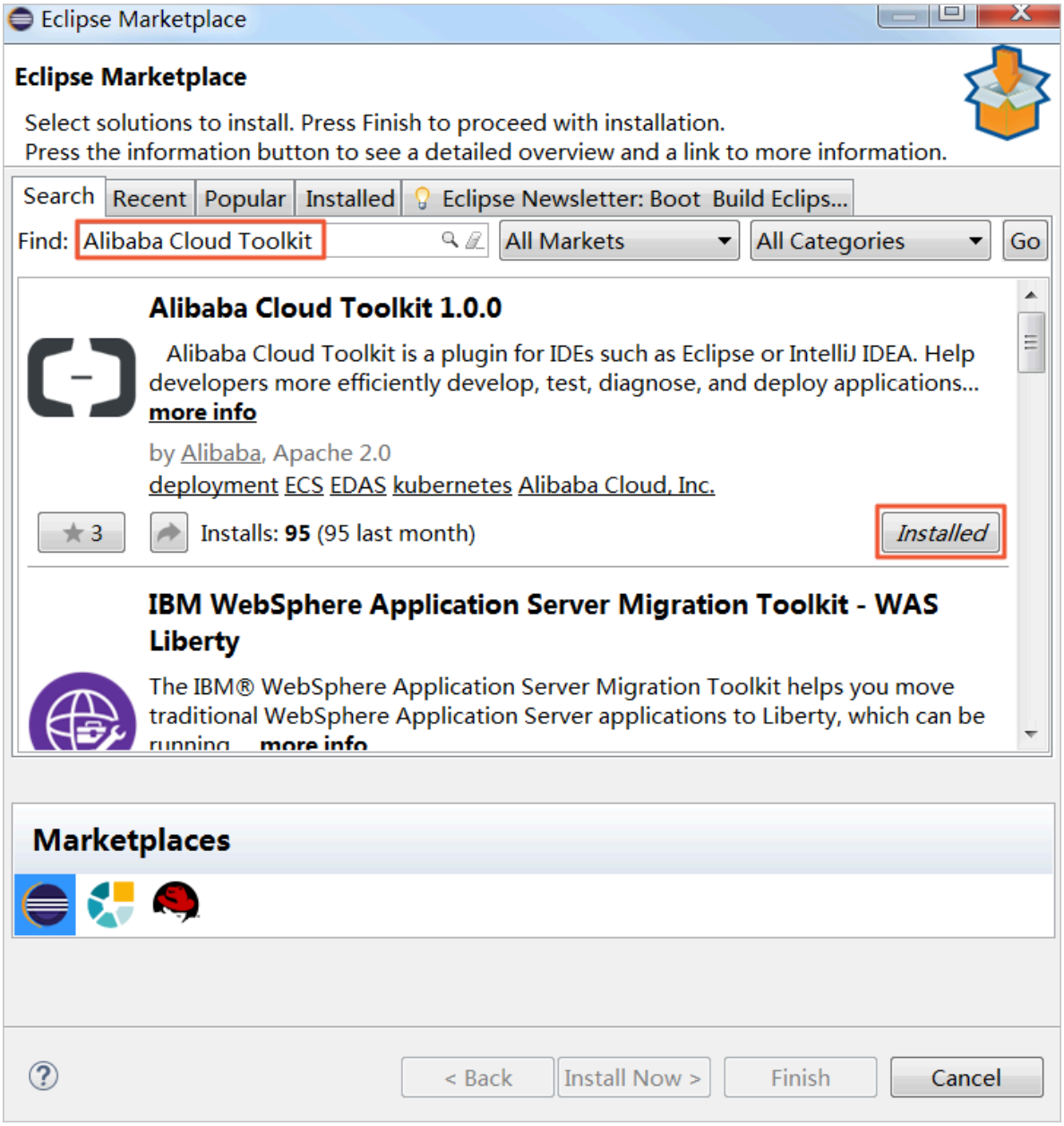
- 下载并安装*JDK 1.8 或更高版本*。
- 下载并安装适用于 Java EE 开发人员的*Eclipse IDE、4.5.0#代号#Mars#或更高版本*。

有两种安装Cloud Toolkit 方式：在Eclipse Marketplace中下载安装和直接通过Cloud Toolkit的官方地址安装。

在Eclipse Marketplace中下载安装

1. 启动Eclipse。
2. 在菜单栏中选择Help > Eclipse Marketplace...。
3. 在Eclipse Marketplace对话框中Find右侧的文本框中输入Alibaba Cloud Toolkit，然后回车。

4. 单击搜索结果中Alibaba Cloud Toolkit右下角的installed。



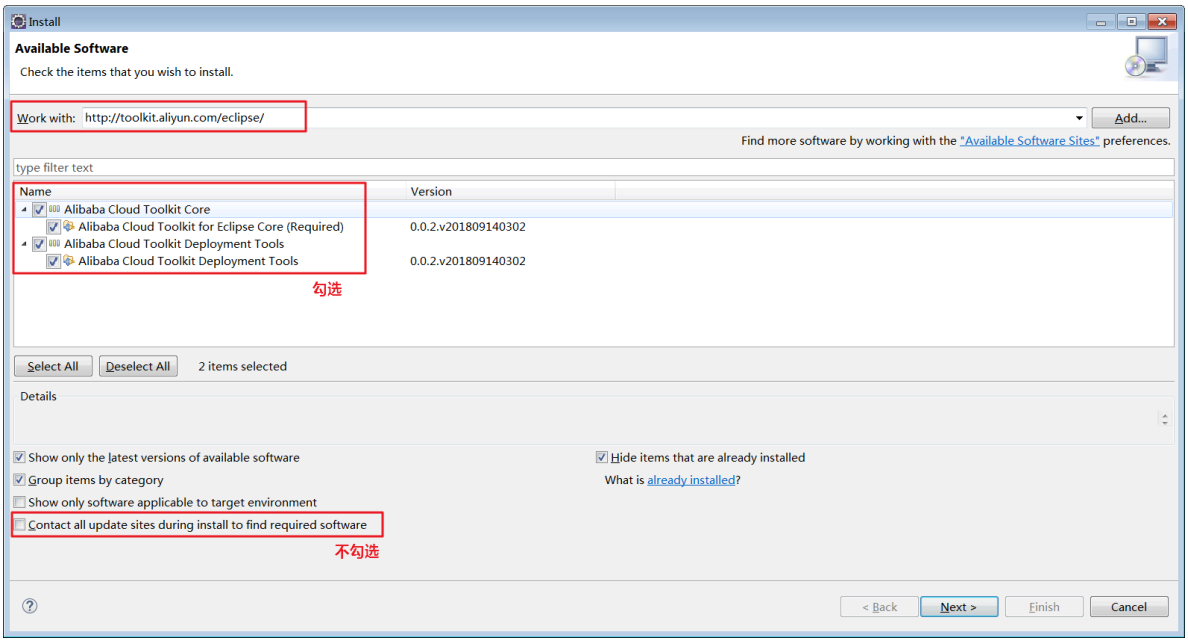
5. 按照Eclipse安装页面的提示，完成后续安装步骤。

直接通过Cloud Toolkit 的官方地址安装

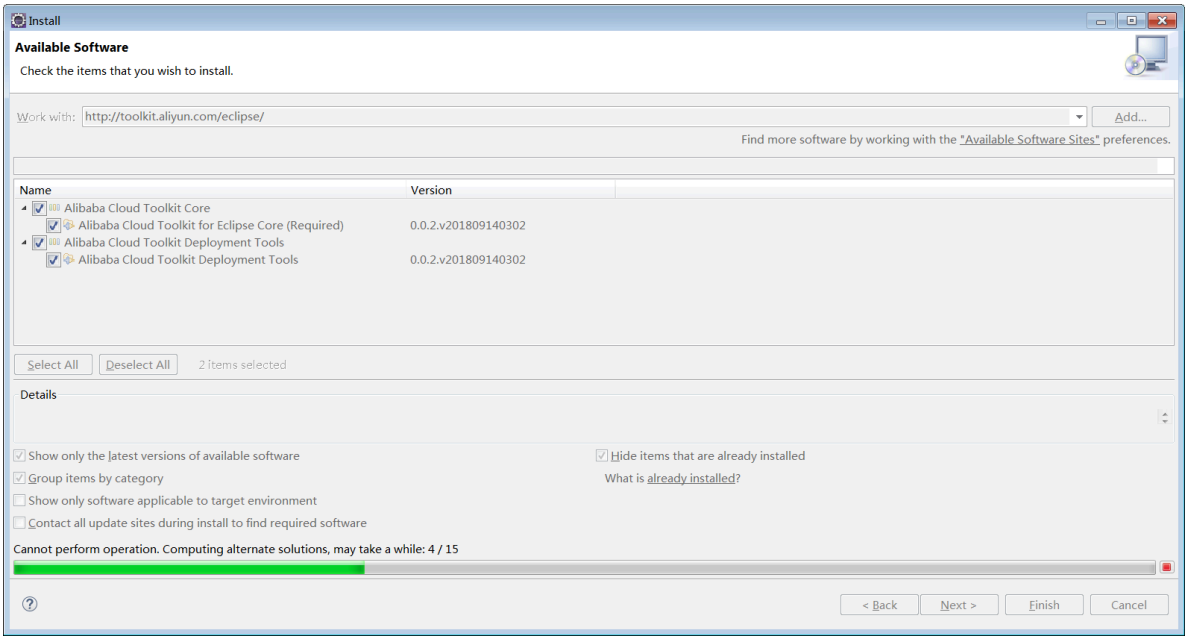
> **说明:**
> 如果您无法连接Eclipse Market服务器，请选择这种安装方式。

1. 启动Eclipse。
2. 在菜单栏中选择Help > Install New Software。
3. 在Available Software对话框的Work with文本框输入Cloud Toolkit for Eclipse的URL `http://toolkit.aliyun.com/eclipse/`。

4. 在下面的列表区域中勾选需要的组件`Alibaba Cloud Toolkit Core`和`Alibaba Cloud Toolkit Deployment Tools`，并在下方的Details区域中不勾选`Connect all update sites during install to find required software.`



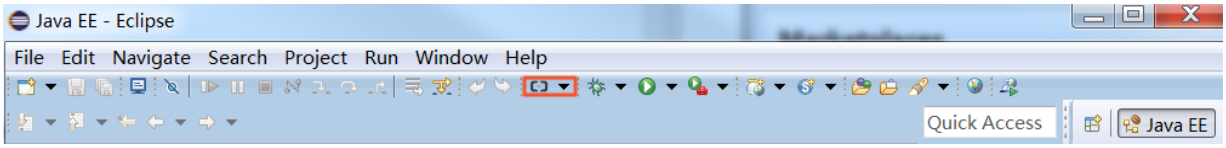
5. 配置完成后，单击Next，Elipse开始安装插件，并显示安装进度。



6. 按照Eclipse安装页面的提示，完成后续安装步骤。

预期结果

插件安装成功后，重启Eclipse，您可以在工具栏看到Alibaba Cloud Toolkit的图标。



获取Access Key ID和Access Key Secret

您本地的应用部署到云端时，都需要使用阿里云上的资源、应用。所以在部署前，需要设置您的阿里云账号信息，以保证拥有使用、管理相关资源、应用的权限。

使用阿里云主账号获取Access Key ID和Access Key Secret

1. 登录容器服务管理控制台。
2. 将光标滑动（非单击）到控制台页面右上角您的头像上，在弹出的下拉菜单中单击accesskeys。

3. 在安全提示对话框中单击继续使用AccessKey。

4. 在安全信息管理页面，用户 AccessKey区域右侧单击创建AccessKey，在手机验证对话框中单击点击获取后输入验证码。
5. 记录该账号的Access Key ID和Access Key Secret。

使用 RAM 子账号获取Access Key ID和Access Key Secret

1. 登录*RAM*子账号登录页面，输入您的子账号，单击下一步，再输入密码，单击登录。
2. 将光标滑动（非单击）到控制台页面右上角您的头像上，在弹出的下拉菜单中单击AccessKey管理。

3. 在安全信息管理页面，用户 AccessKey区域右侧单击创建AccessKey。

说明:

- 如果您当前子账号的创建AccessKey置灰不可用，请使用主账号对该子账号授权，可参考使用子账号。
- 在新建用户AccessKey对话框中，单击AccessKey详情右侧下拉箭头，记录当前子账号的AccessKeyID和AccessKeySecret。
- 这是用户AccessKey可供下载的唯一机会，请及时保存！

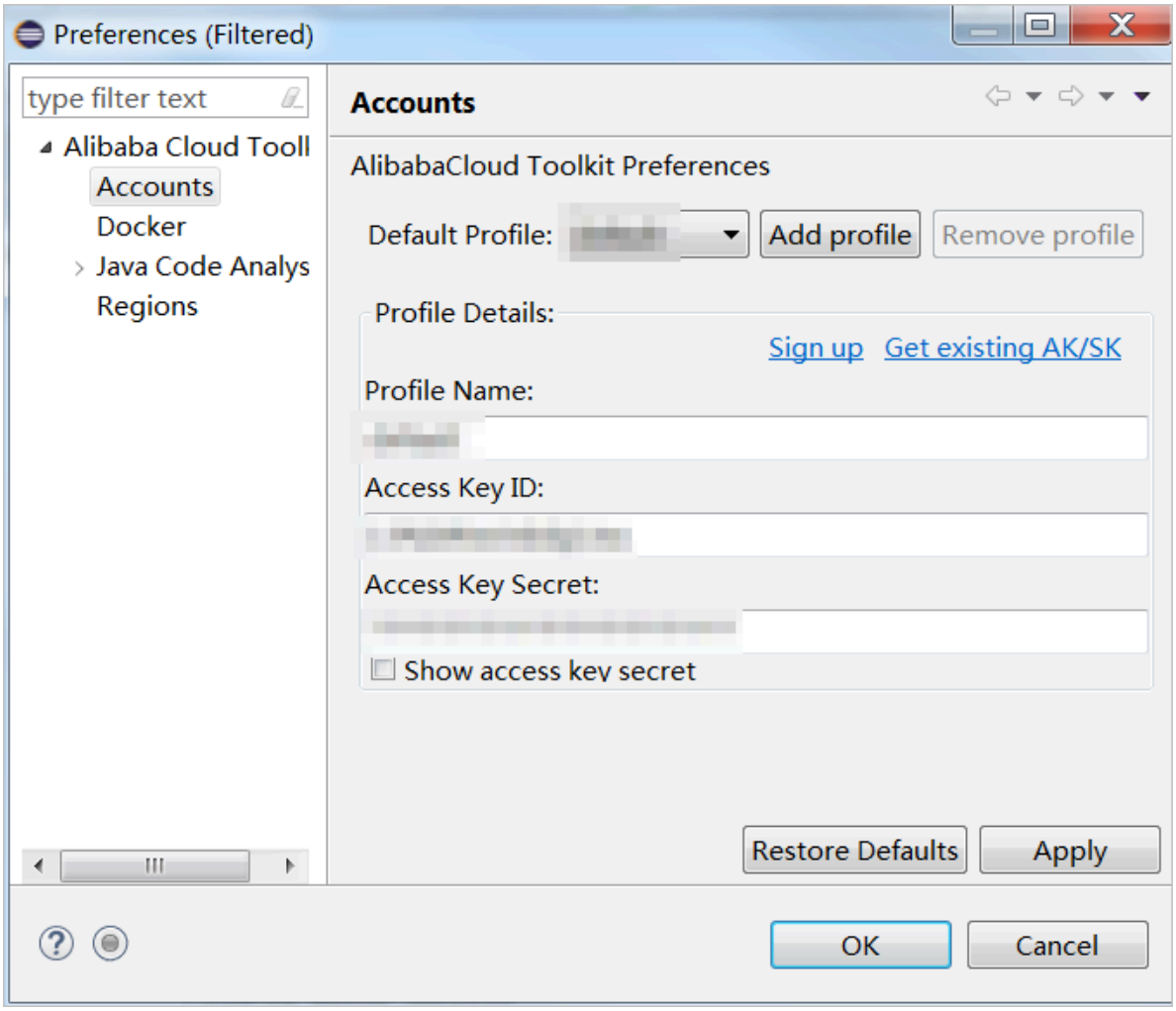
设置Access Key ID和Access Key Secret

1. 启动Eclipse。
2. 在工具栏单击Alibaba Cloud Toolkit图标，在下拉菜单中单击Alibaba Cloud Preference...。
3. 在Preference (Filtered)对话框的左侧导航栏中单击Accounts。
4. 在Accounts区域设置Access Key ID和Access Key Secret，然后单击OK。

说明:

- 如果您已有阿里云账号，单击Get existing AK/SK， 参考文档获取Access Key ID和Access Key Secret。

- 如果您没有阿里云账号，单击Sign up，进入阿里云账号注册页面，注册账号。注册完成后按照上述方式获取Access Key ID和Access Key Secret。



部署应用

前提条件

- 您已使用设置Access Key ID 和 Access Key Secret的阿里云账号进入容器镜像服务控制台，创建容器镜像仓库，可参考仓库构建。
- 您已使用设置Access Key ID 和 Access Key Secret的阿里云账号进入容器服务管理控制台，并使用镜像创建应用，可参考使用镜像创建无状态Deployment应用或使用镜像创建有状态StatefulSet应用。

1. 设置用于打包本地镜像的Docker环境。

   a. 在 Eclipse 工具栏单击Alibaba Cloud Toolkit， 在下拉菜单中单击Alibaba Cloud Preference...。

   b. 在Preference (Filtered)对话框的左侧导航栏中单击Docker。

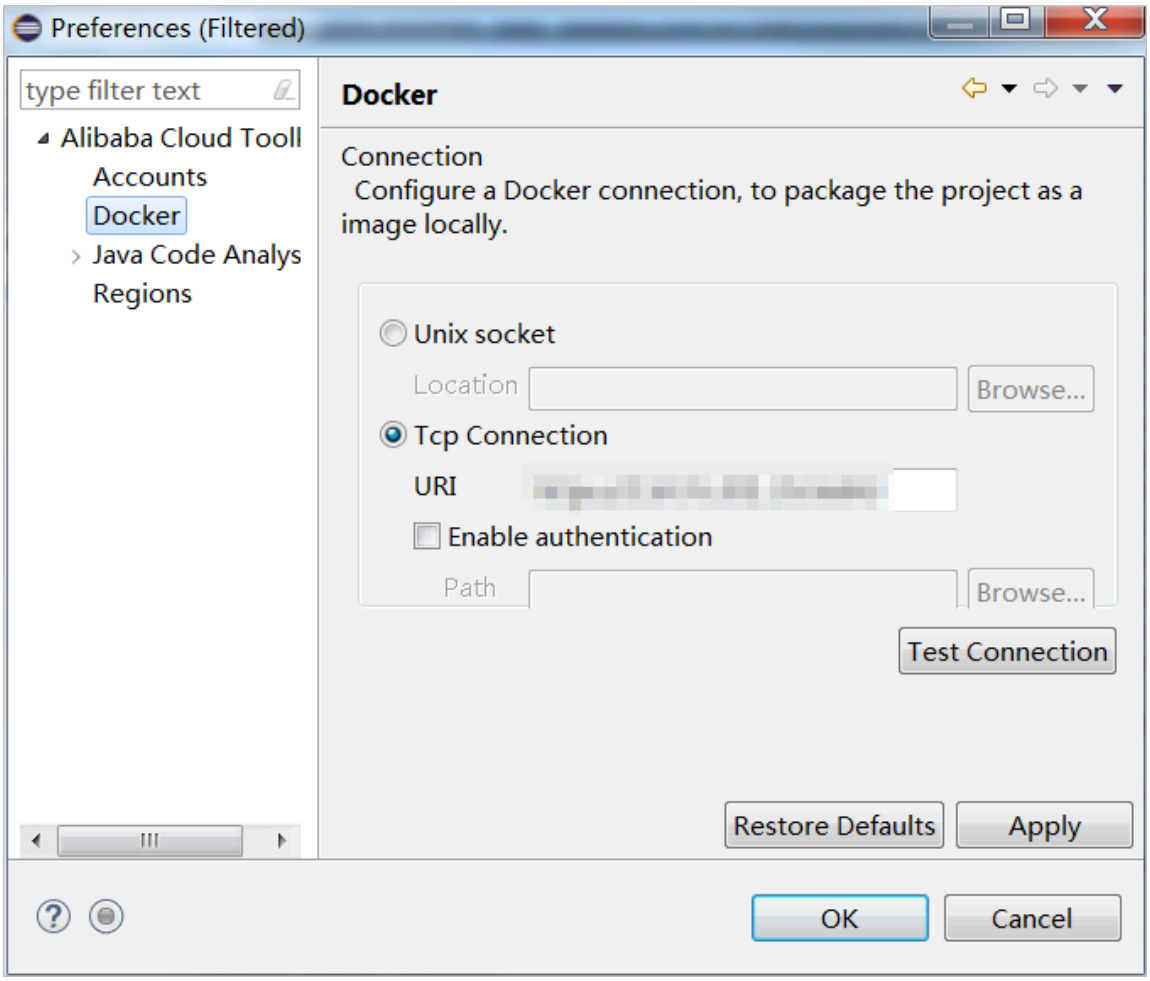
   c. 在Docker界面中设置可连接的 Docker 环境，包括本地和远程两种方式，然后单击OK。

   本地Docker环境

   - 如果您本地为Mac或Linux操作系统，勾选Unix Socket后单击Browse...，选择本地的Docker安装目录。
   - 如果您本地为Windows操作系统，勾选Tcp Connection后在 URI 右侧文档框输入本地 Docker 的 URI，如`http://127.0.0.1:2375`。

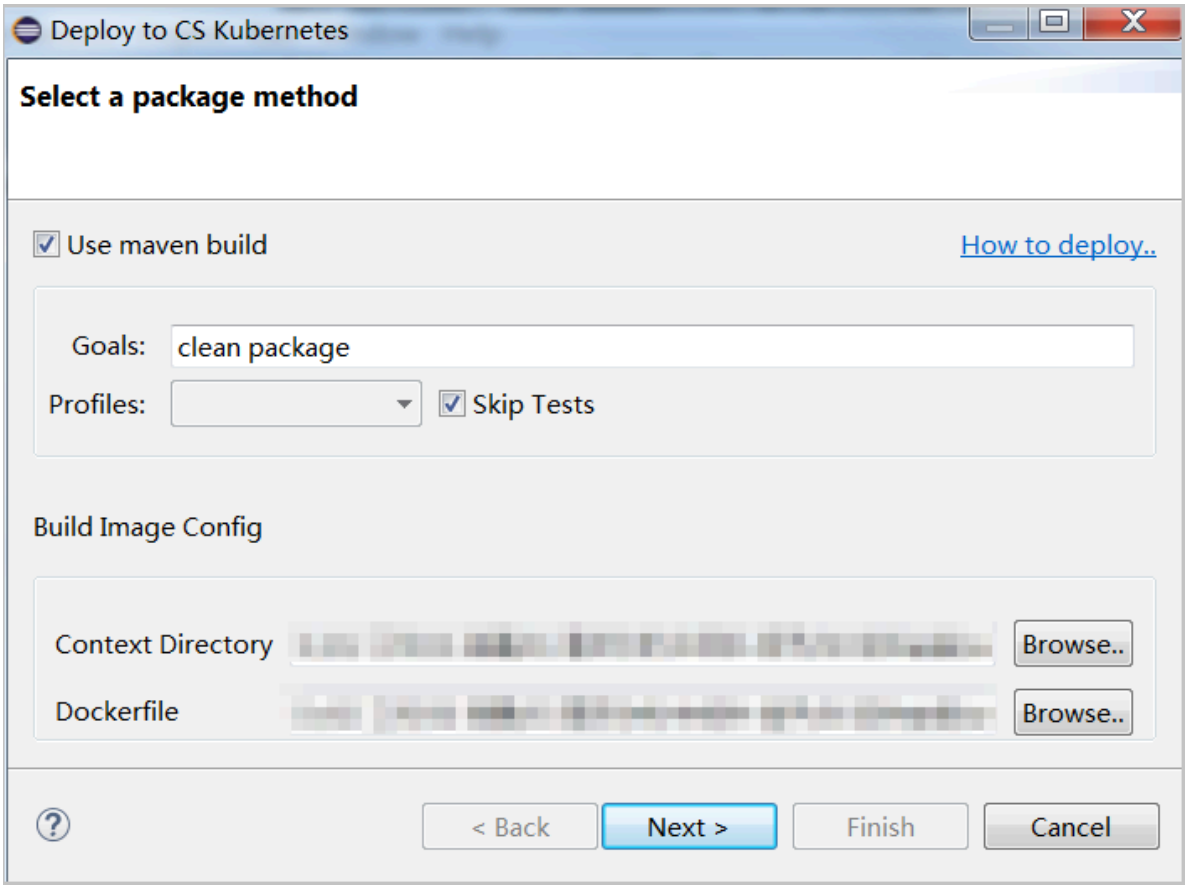
   远程Docker环境

   勾选Tcp Connection后在 URI 右侧文档框输入远端Docker环境的URI（包括 IP 地址和端口），如`http://127.0.0.1:2375`，并确保远程主机的 HTTP 服务开启。



2. 在 Eclipse 界面左侧的Package Explorer中右键单击您的 Docker 应用工程名，在弹出的下拉菜单中选择Alibaba Cloud > Deploy to CS Kubernetes... 。
3. 在Select a package method对话框选择本地应用程序的Context Directory和Dockerfile（通常会根据您本地的应用工程自动识别并设置），然后单击Next。

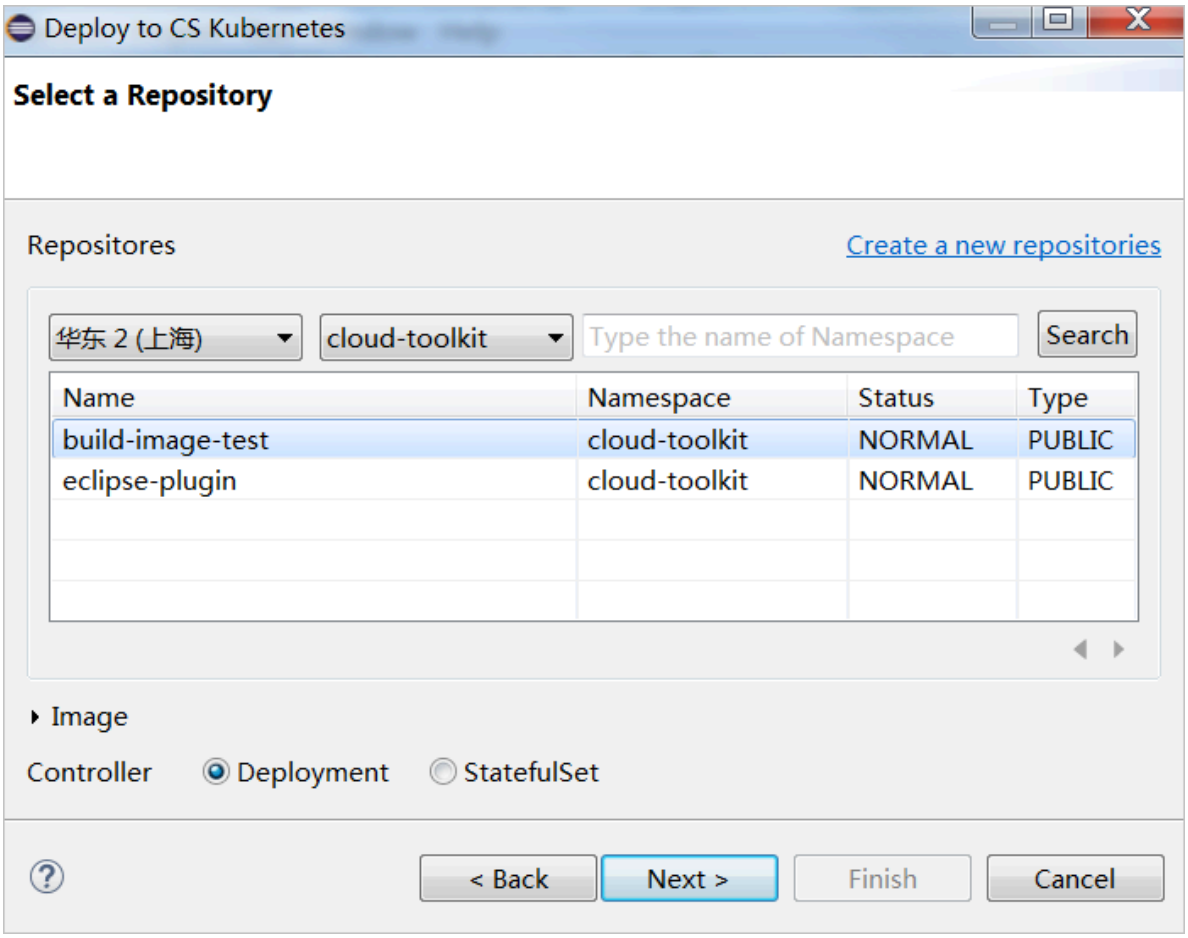
> **说明:**
> 您可以根据您的需要决定是否勾选Use maven build使用Maven构建应用工程。



4. 在Select a Repository对话框选择容器镜像服务的地域，命令空间和镜像仓库，然后单击Next。

> **说明:**

如果您还没有镜像仓库，在对话框右上角单击Create a new repository跳转到容器镜像仓库创建镜像仓库。创建步骤可参考容器镜像服务文档。

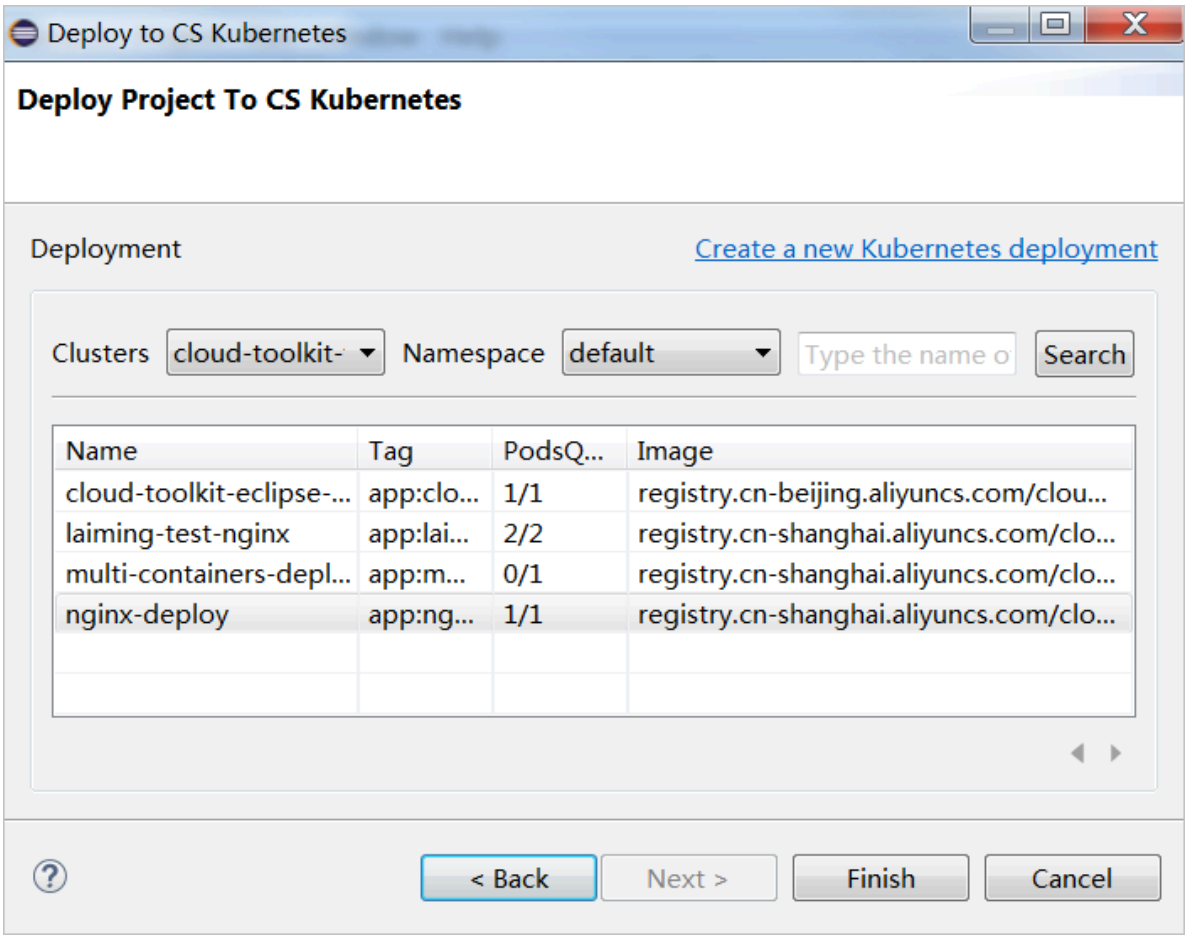


5. 在Deploy Project to CS Kubernetes对话框选择容器服务Kubernetes的Cluster、Namespace和Deployment，然后单击Finish。

说明：

- 如果您还没有创建容器服务Kubernetes的Deployment，在对话框右上角单击Create a new Kubernetes deployment，跳转到容器服务管理控制台创建Deployment。创建步骤可参考容器服务 Kubernetes 版文档。

- Alibaba Cloud Toolkit目前只支持以更新方式部署您的应用，需要您预先在控制台手动创建一个部署（Deployment），然后基于该部署将镜像替换为您的镜像。



预期结果

部署开始后，Eclipse的Console区域会打印部署日志。您可以根据日志信息检查部署结果。

说明:

如果您在使用Cloud Toolkit过程中有任何疑问，欢迎您扫描下面的二维码加入钉钉群进行反馈。

7.2 在Kubernetes服务上快速搭建Jenkins环境并完成应用构建到部署的流水线作业

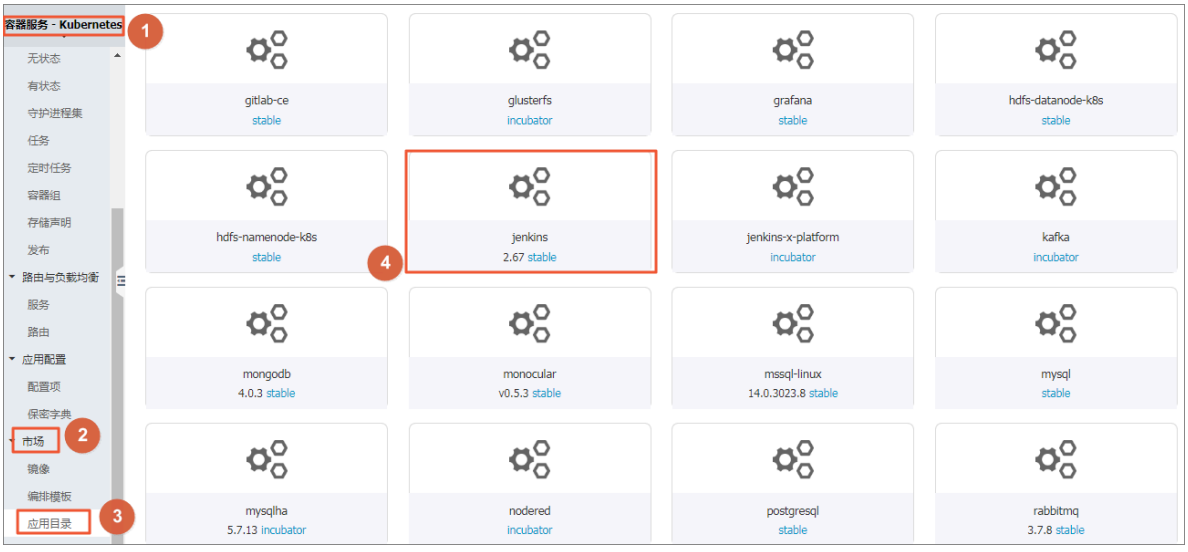
本文主要演示如何在阿里云Kubernetes服务上快速搭建Jenkins持续集成环境，并基于提供的示例应用快速完成应用源码编译、应用镜像构建和推送以及应用部署的流水线。

前提条件

您已经成功创建了一个Kubernetes集群。有关详细信息，请参见创建Kubernetes集群。

部署Jenkins

1. 登录容器服务管理控制台。
2. 在左侧导航栏中，选择容器服务 - Kubernetes > 市场 > 应用目录。选择jenkins。



3. 选择参数页签。

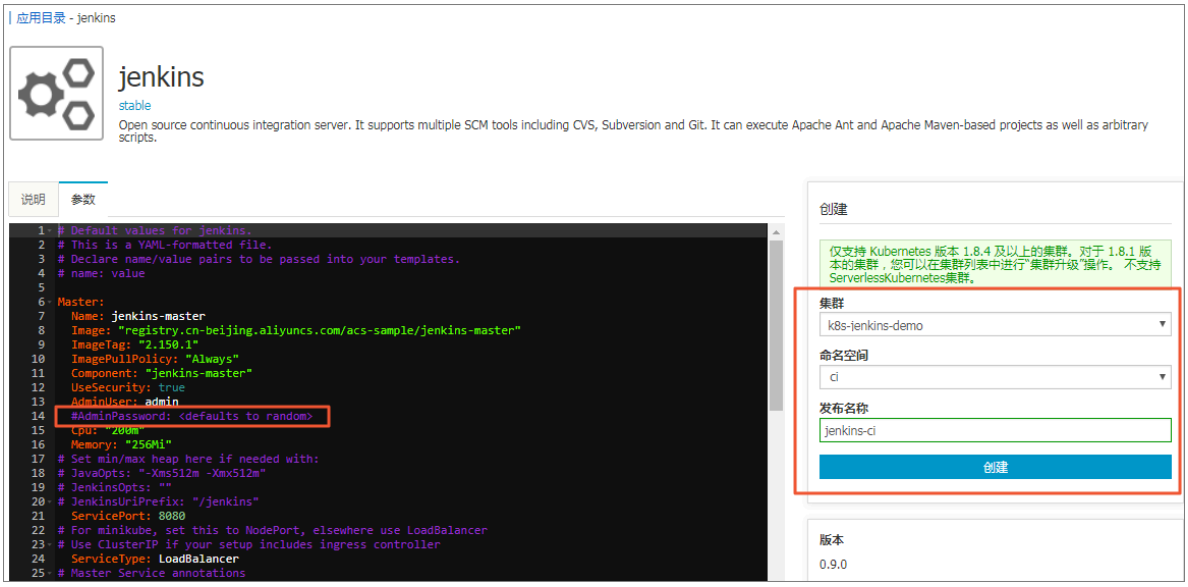
4. 修改AdminPassword字段。

> **说明:**
> 为确保密码生效，您需要去掉AdminPassword字段前的#。

如果您未设置登录密码，即未修改AdminPassword字段，Jenkins部署成功后系统会自动生成密码，您可以通过以下命令进行查看：

```
$ printf $(kubectl get secret --namespace ci jenkins-ci-jenkins -o
jsonpath="{.data.jenkins-admin-password}" | base64 --decode);echo
```

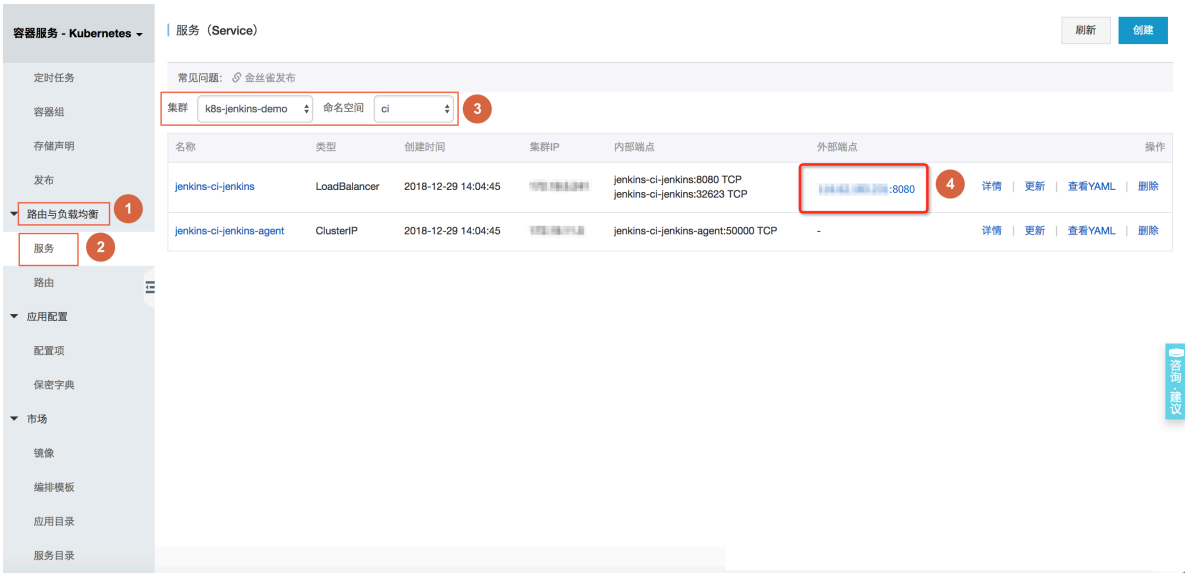
5. 选择Kubernetes集群以及命名空间，填写发布名称，并单击创建。



> **说明:**
> 建议选择自定义命名空间或default命名空间。本示例中选择自定义命名空间ci。

6. 在左侧导航栏中，选择路由与负载均衡 > 服务。
7. 选择部署了Jenkins的集群以及相应的命名空间。

8. 单击Jenkins服务的外部端点，访问并登录Jenkins。



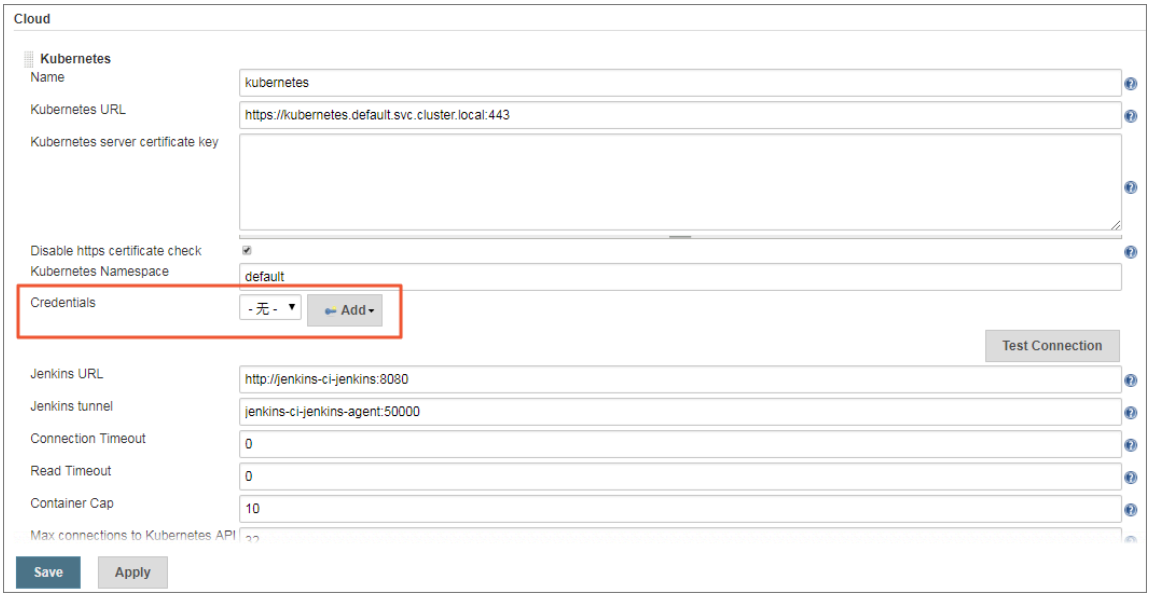
创建集群证书和镜像仓库证书，构建和部署示例应用

1. **创建Kubernetes集群证书。**

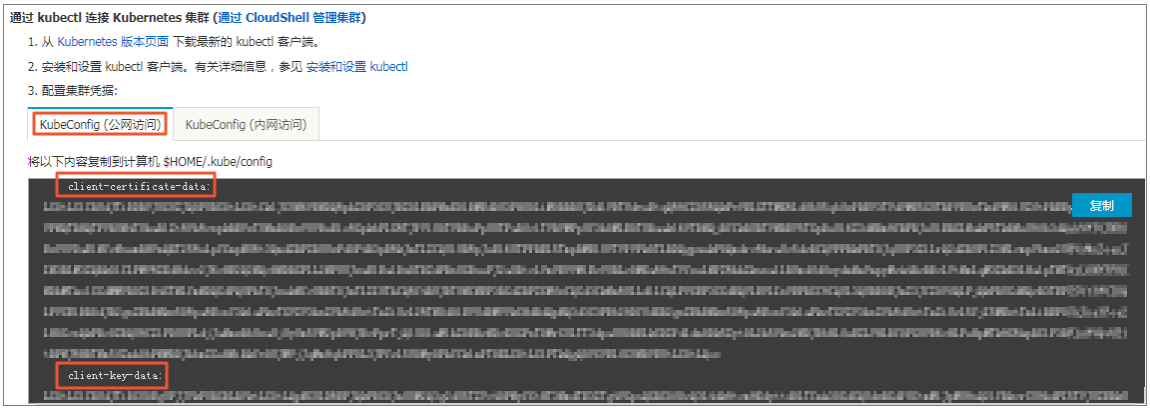
   a. **在Jenkins的左侧导航栏中，选择系统管理。**

   b. **在右侧的Manage Jenkins页面，单击系统设置。**

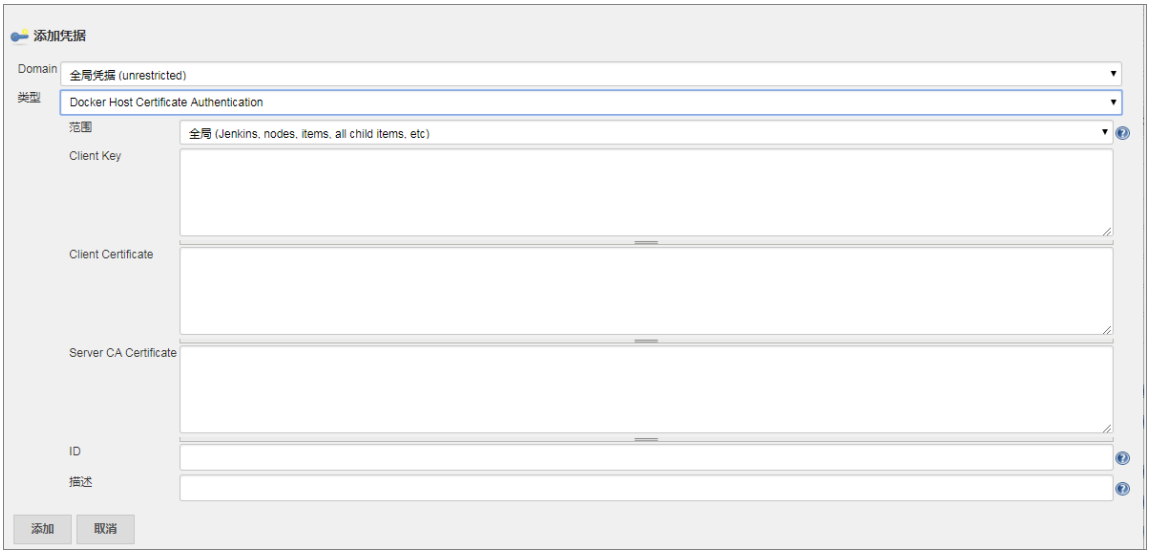
   c. **在Cloud区域中，单击Credentials右侧的Add。**



   在添加凭据前，先在Kubernetes集群的基本信息页面，找到配置集群凭据中提供的KubeConfig。



   d. **在弹出的添加凭据对话框中，完成以下配置：**

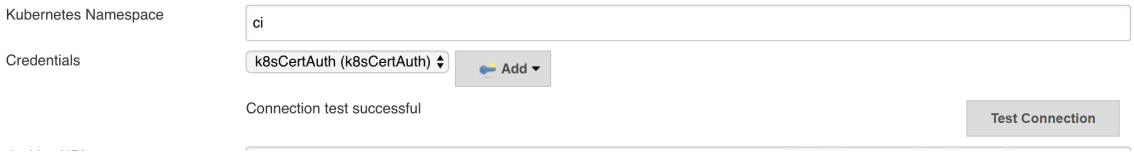


- 类型：选择Docker Host Certificate Authentication。
- Client Key: 填写KubeConfig中client-key-data对应的内容。
- Client Certificate: 填写KubeConfig中client-certificate-data对应的内容。
- ID：填写证书ID。本示例中填写k8sCertAuth。
- 描述：填写描述。

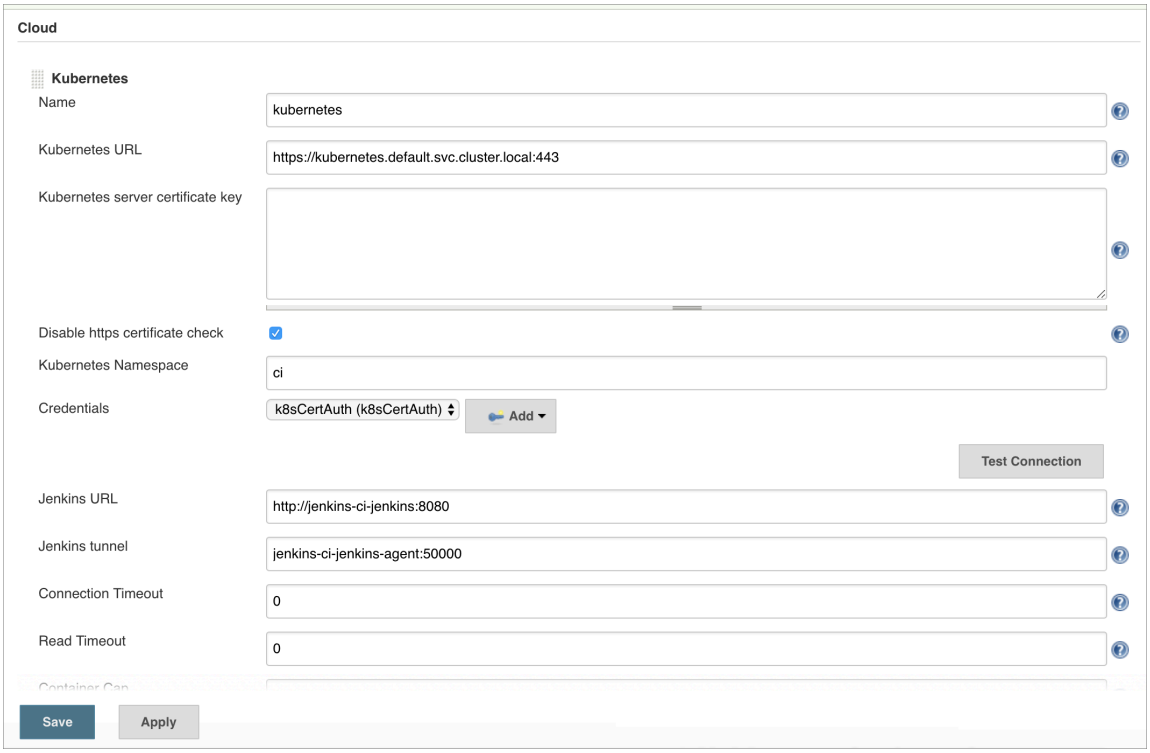
e. 单击添加。

f. 测试连通性。

从Credentials下拉框中选择上一步添加的凭据，单击Test Connection。



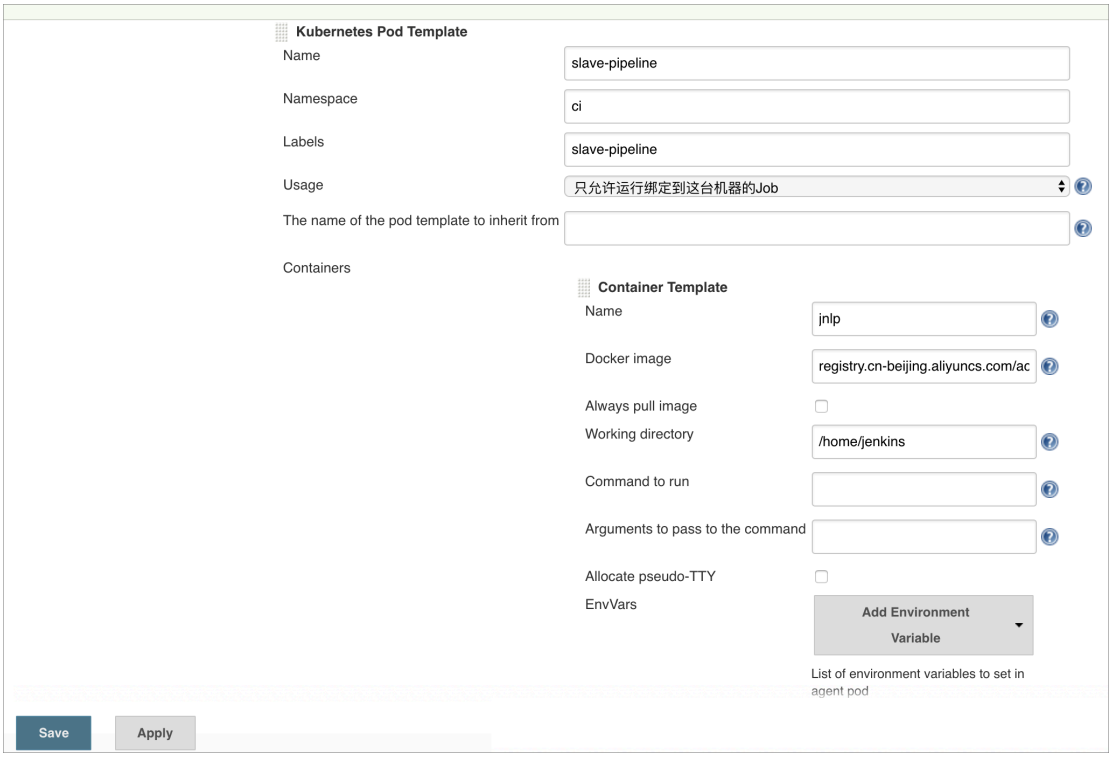
g. Kubernetes集群动态分配构建pod，其配置如下图所示。



h. 配置Kubernetes Pod Template。

slave-pipeline使用了4个container分别完成流水线中各个stage的构建。

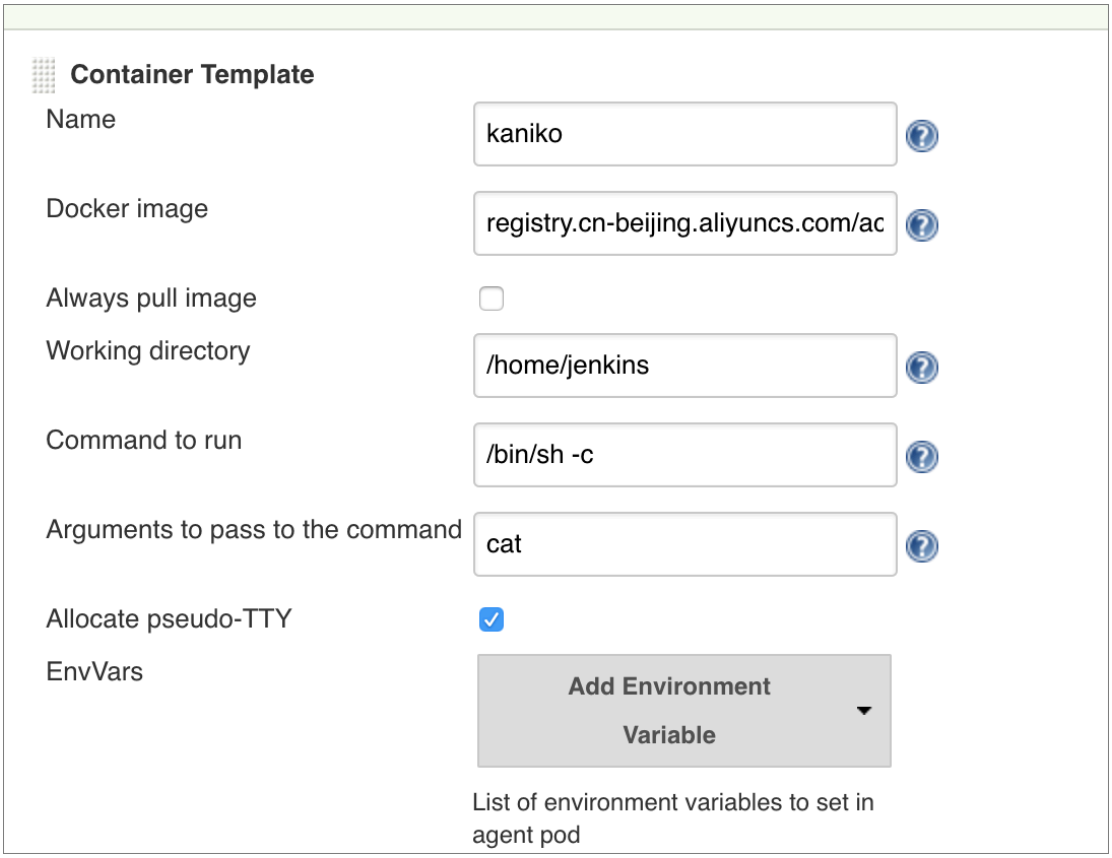
- **Container jnlp的配置如下图所示。**



使用jenkins-slave-jnlp作为Docker镜像。jenkins-slave-jnlp用于构建节点jnlp连接master。

```
registry.cn-beijing.aliyuncs.com/acs-sample/jenkins-slave-jnlp:3.14-1
```

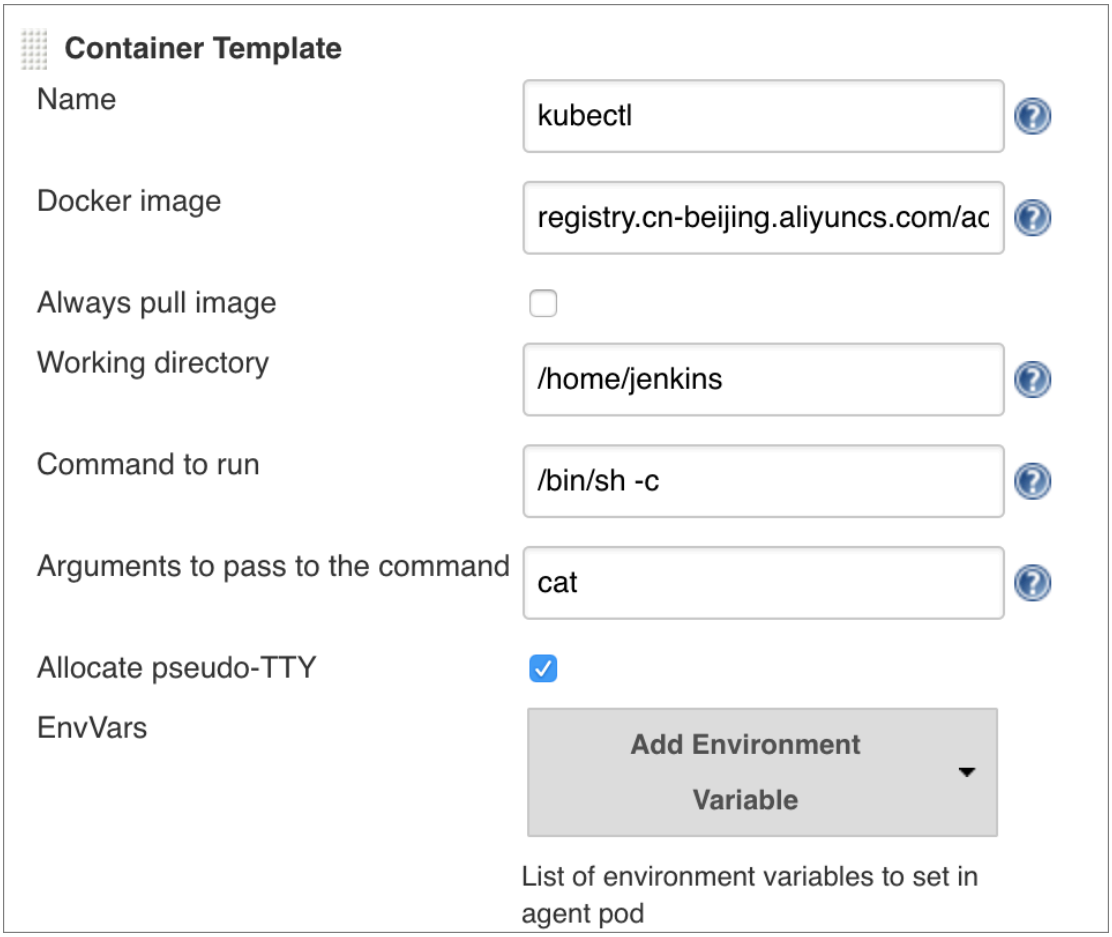
- Container kaniko的配置如下图所示。



使用jenkins-slave-kaniko作为Docker镜像。jenkins-slave-kaniko用于构建和推送应用的镜像。

```
registry.cn-beijing.aliyuncs.com/acs-sample/jenkins-slave-kaniko:0.6.0
```

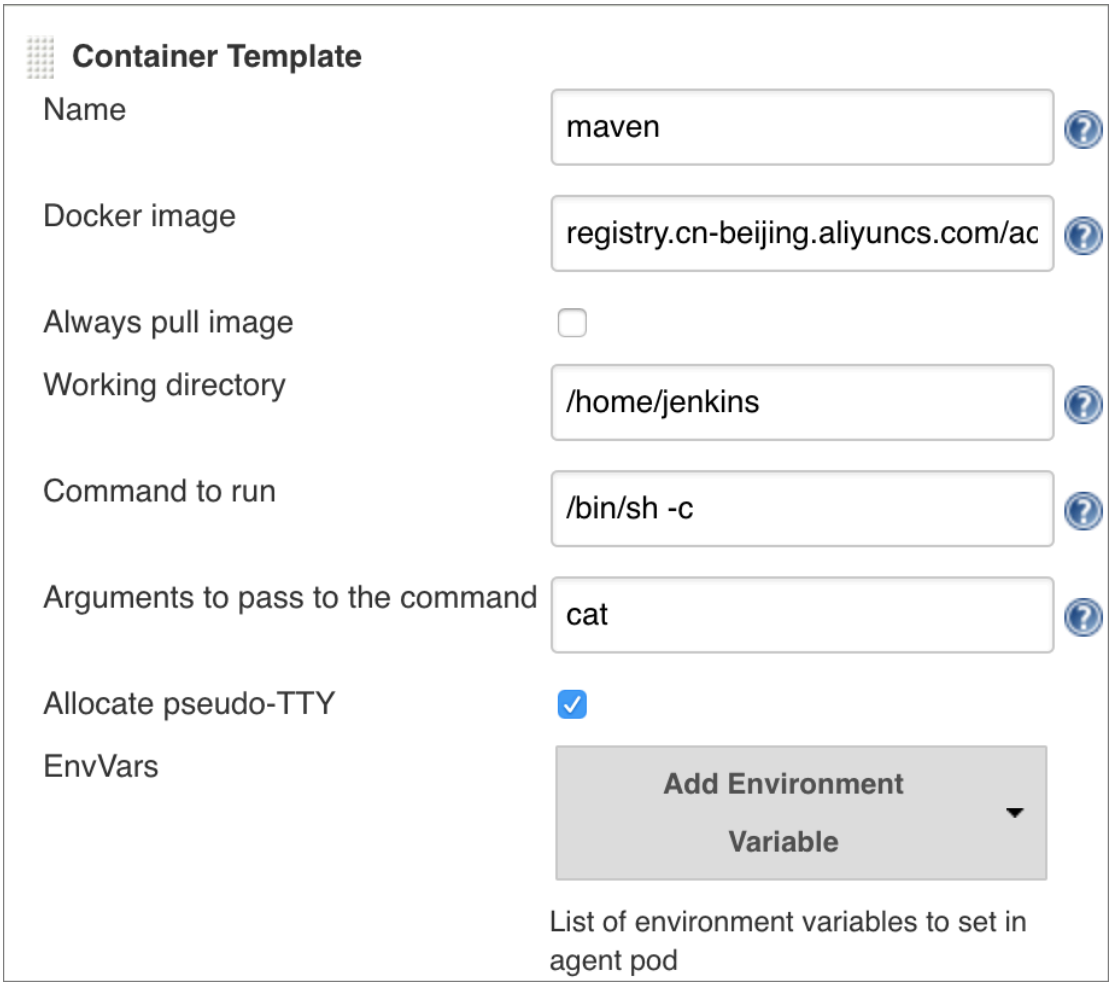
- Container kubectl的配置如下图所示。



使用jenkins-slave-kubectl作为Dokcer镜像。jenkins-slave-kubectl 用于kubectl部署应用。

```
registry.cn-beijing.aliyuncs.com/acs-sample/jenkins-slave-
kubectl:1.11.5
```

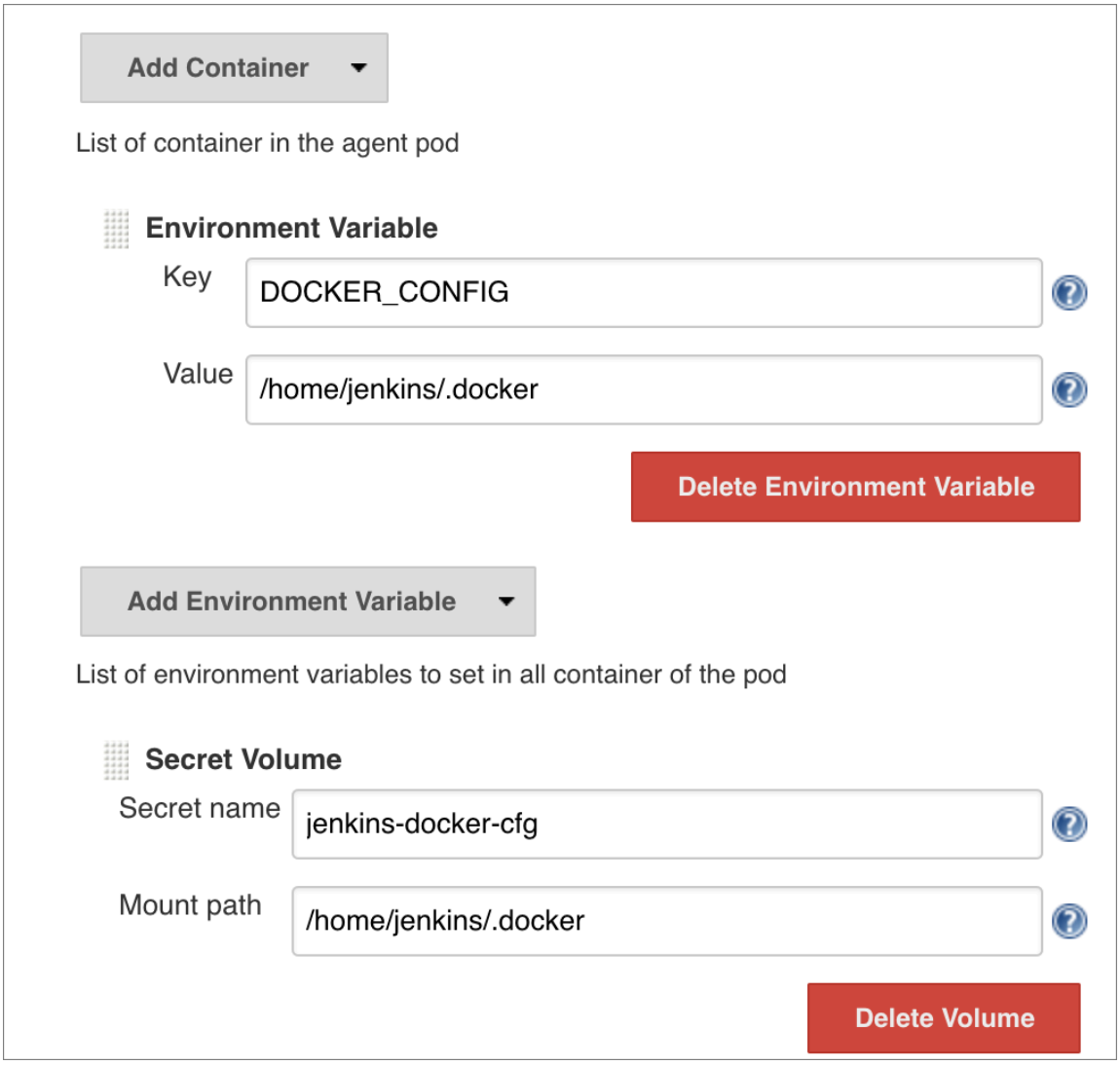
- Container maven的配置如下图所示。



使用jenkins-slave-maven作为Docker镜像。jenkins-slave-maven用于mvn打包构建。

```
registry.cn-beijing.aliyuncs.com/acs-sample/jenkins-slave-maven
:3.3.9-jdk-8-alpine
```

i. **kaniko需要配置镜像仓库权限。具体配置如下图所示。**



j. 单击Save保存配置。

2. 通过kubectl在Kubernetes集群中创建jenkins-docker-cfg secret，用于设置镜像仓库权限。

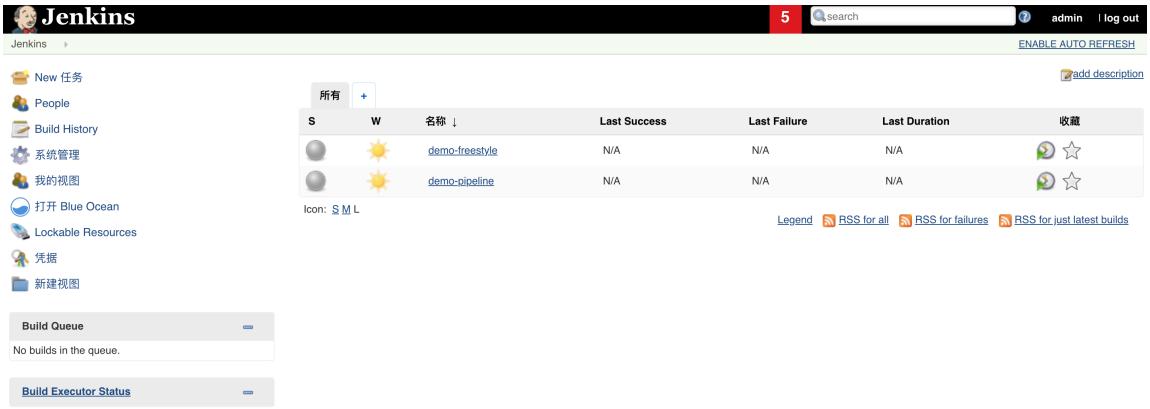
   本示例中使用阿里云镜像服务提供的北京区域镜像仓库：

```
$ docker login -u xxx -p xxx registry.cn-beijing.aliyuncs.com
Login Succeeded
```

```
$ kubectl create secret generic jenkins-docker-cfg -n ci --from-file
=/root/.docker/config.json
```

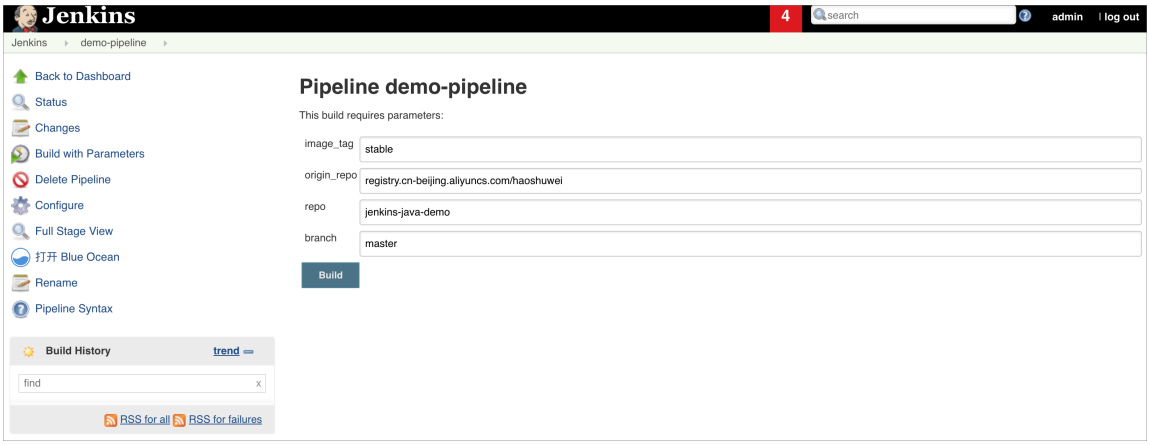
3. 构建demo-pipeline并访问应用服务。

   a. 在Jenkins首页，单击demo-pipeline。



   b. 在左侧导航栏中，选择Build with Parameters。

   c. 根据自己的镜像仓库信息修改构建参数。本示例中源码仓库分支为master，镜像为registry.cn-beijing.aliyuncs.com/haoshuwei:stable。



   d. 单击Build。

   e. 查看Build History。下图表示构建成功。

f. 构建成功后，登录容器服务管理控制台，查看应用的服务地址。

点击这里获取示例项目中使用的源码仓库。

了解更多容器服务的相关内容，请参见容器服务。

了解更多kaniko的相关内容，请参见*kaniko*。

7.3 使用GitLab CI在Kubernetes服务上运行GitLab Runner并执行Pipeline

本文主要演示如何在Kubernetes集群中安装、注册GitLab Runner，添加Kubernetes类型的executor来执行构建，并以此为基础完成一个Java源码示例项目从编译构建、镜像打包到应用部署的CICD过程。

背景信息

本文以构建一个Java软件项目并将其部署到阿里云容器服务Kubernetes集群中为例，说明如何使用GitLab CI在阿里云Kubernetes服务上运行GitLab Runner、配置Kubernetes类型的executor，并执行Pipeline。

创建GitLab源码项目并上传示例代码

1. 创建GitLab源码项目。

   本示例中创建的GitLab源码项目地址为：

   ```
   http://xx.xx.xx.xx/demo/gitlab-java-demo.git
   ```

2. 执行以下命令获取示例代码并上传至GitLab。

   ```
   $ git clone https://code.aliyun.com/CodePipeline/gitlabci-java-demo.git
   ```

```
$ git remote add gitlab http://xx.xx.xx.xx/demo/gitlab-java-demo.git
```

```
$ git push gitlab master
```

在Kubernetes集群中安装GitLab Runner

1. 获取GitLab Runner的注册信息。

   a. 获取项目专用Runner的注册信息。

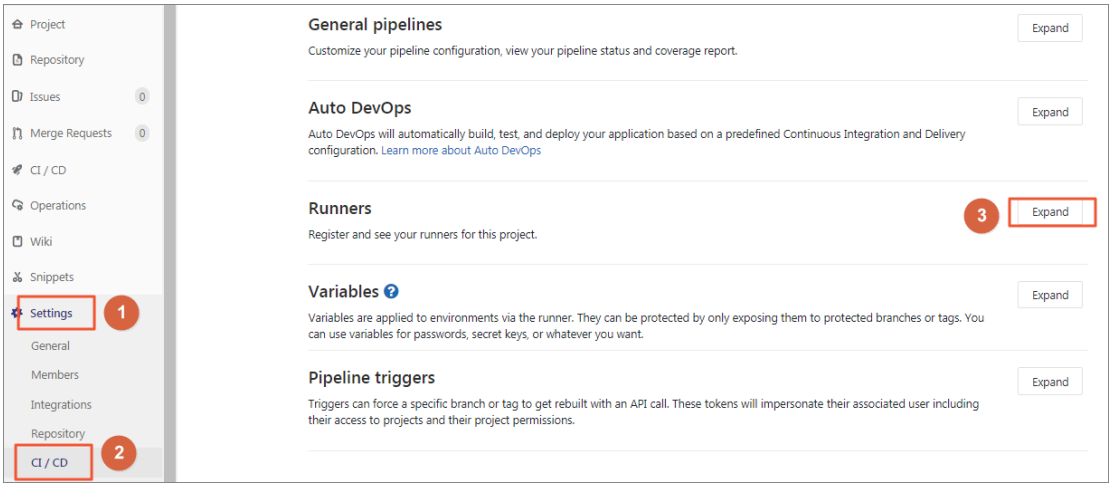
      A. 登录GitLab。

      B. 在顶部导航栏中，选择Projects > Your projects。

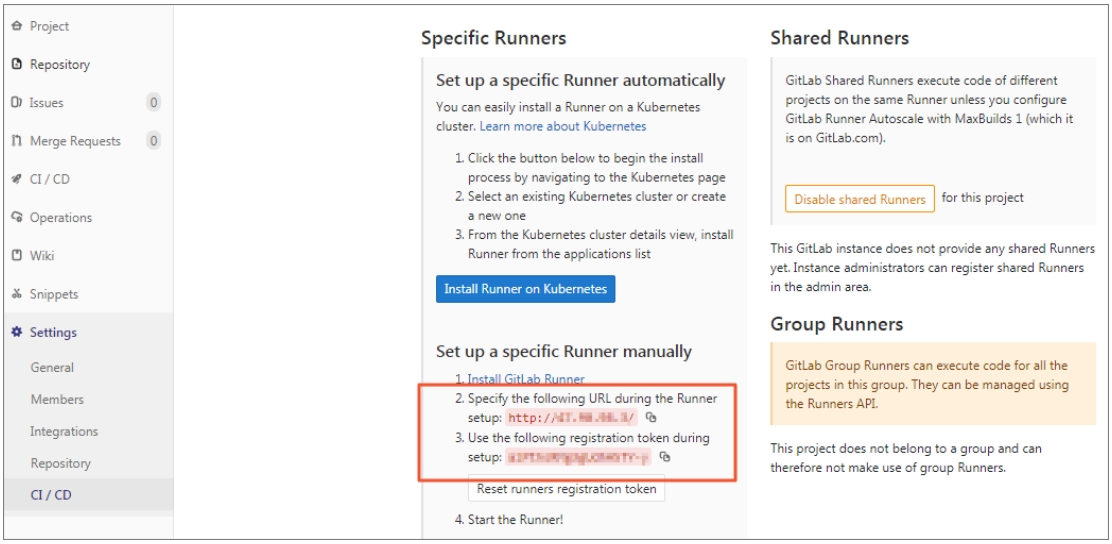
      C. 在Your projects页签下，选择相应的project。

      D. 在左侧导航栏中，选择Settings > CI / CD。

      E. 单击Runners右侧的Expand。



      F. 获取URL和registration token信息。



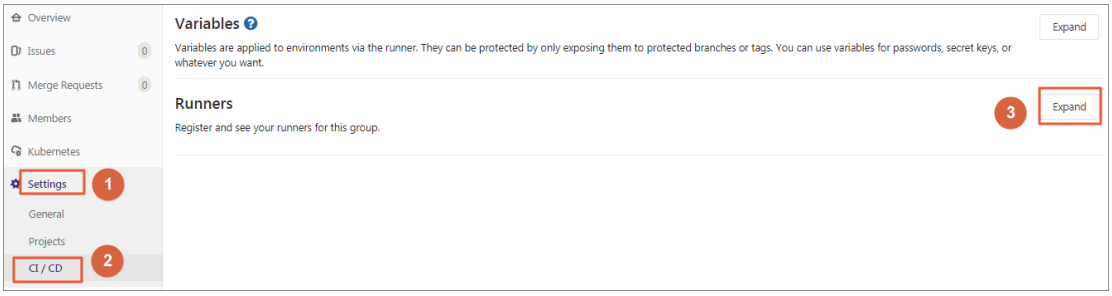
   b. 获取Group Runners的注册信息。

      A. 在顶部导航栏中，选择Groups > Your groups。

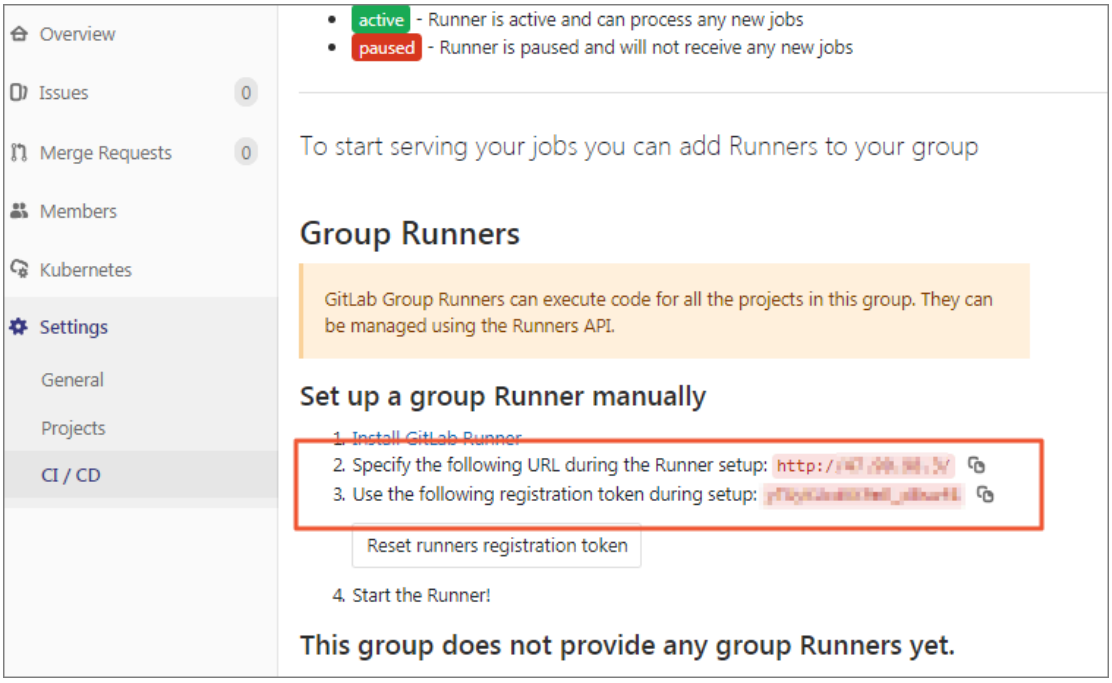
      B. 在Your groups页签下，选择相应的group。

C. 在左侧导航栏中，选择Settings > CI / CD。

D. 单击Runners右侧的Expand。



E. 获取URL和registration token信息。



c. 获取Shared Runners的注册信息。

> **说明:**
> 只有管理员有权限执行此步操作。

A. 在顶部导航栏中，单击  进入Admin Area页面。

B. 在左侧导航栏中，选择Overview > Runners。

C. 获取URL和registration token信息。

2. **执行以下命令获取并修改GitLab Runner的Helm Chart。**

```
$ git clone https://code.aliyun.com/CodePipeline/gitlab-runner.git
```

修改*values.yaml*文件, 示例如下:

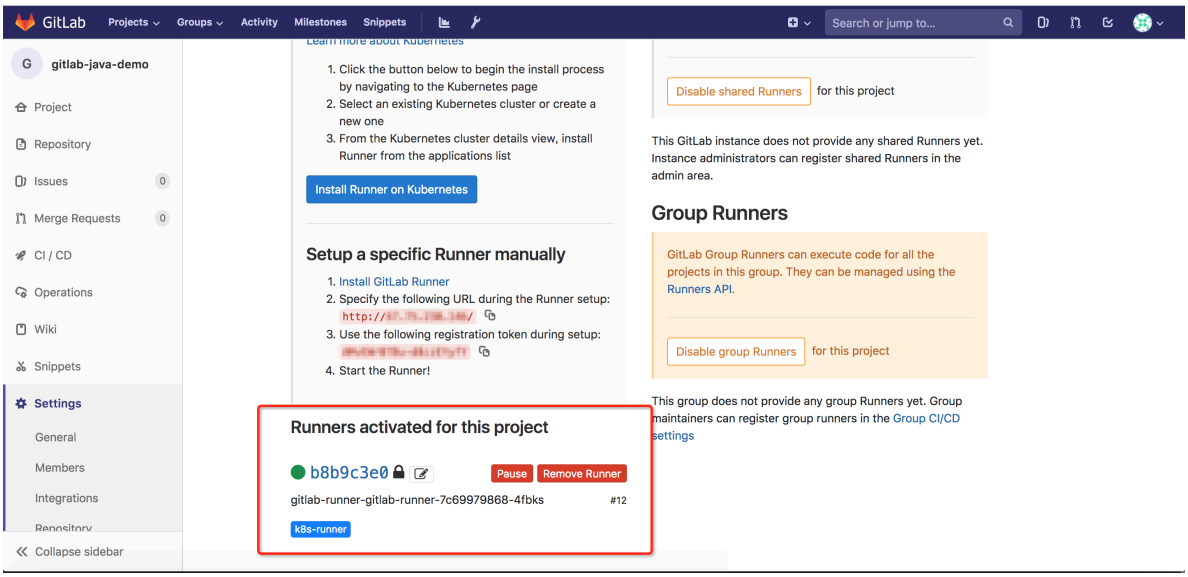
```
## GitLab Runner Image
##
image: gitlab/gitlab-runner:alpine-v11.4.0

## Specify a imagePullPolicy
##
imagePullPolicy: IfNotPresent

## Default container image to use for initcontainer
init:
  image: busybox
  tag: latest

## The GitLab Server URL (with protocol) that want to register the
runner against
##
gitlabUrl: http://xx.xx.xx.xx/

## The Registration Token for adding new Runners to the GitLab
Server. This must
## be retreived from your GitLab Instance.
##
runnerRegistrationToken: "AMvEWrBTBu-d8czEYyfY"
## Unregister all runners before termination
##
unregisterRunners: true

## Configure the maximum number of concurrent jobs
##
concurrent: 10

## Defines in seconds how often to check GitLab for a new builds
##
checkInterval: 30

## For RBAC support:
##
rbac:
  create: true
  clusterWideAccess: false

## Configure integrated Prometheus metrics exporter
##
metrics:
  enabled: true

## Configuration for the Pods that that the runner launches for each
 new job
##
runners:
  ## Default container image to use for builds when none is
specified
  ##
  image: ubuntu:16.04
```

```
  ## Specify the tags associated with the runner. Comma-separated
list of tags.
  ##
  tags: "k8s-runner"

  ## Run all containers with the privileged flag enabled
  ## This will allow the docker:dind image to run if you need to run
 Docker
  ## commands. Please read the docs before turning this on:
  ##
  privileged: true

  ## Namespace to run Kubernetes jobs in (defaults to the same
namespace of this release)
  ##
  namespace: gitlab

  cachePath: "/opt/cache"

  cache: {}
  builds: {}
  services: {}
  helpers: {}

resources: {}
```

3. **执行以下命令安装GitLab Runner。**

```
$ helm package .
Successfully packaged chart and saved it to: /root/gitlab/gitlab-
runner/gitlab-runner-0.1.37.tgz

$ helm install --namespace gitlab --name gitlab-runner *.tgz
```

查看相关的deployment/pod是否成功启动。若成功启动，则可在GitLab上看到注册成功的GitLab Runner，如下图所示：



缓存配置

GitLab Runner对缓存方案的支持有限，所以您需要使用挂载volume的方式做缓存。在上面的示例中，安装GitLab Runner时默认使用*/opt/cache*目录作为缓存空间。您也可以通过修改*values.yaml*文件中的`runners.cachePath`字段修改缓存目录。

例如，如需建立maven缓存，您可以在`variables`下添加`MAVEN_OPTS`变量并指定本地缓存目录：

```
variables:
  KUBECONFIG: /etc/deploy/config
  MAVEN_OPTS: "-Dmaven.repo.local=/opt/cache/.m2/repository"
```

如需挂载新的volume，您可以修改*templates/configmap.yaml*文件中的如下字段：

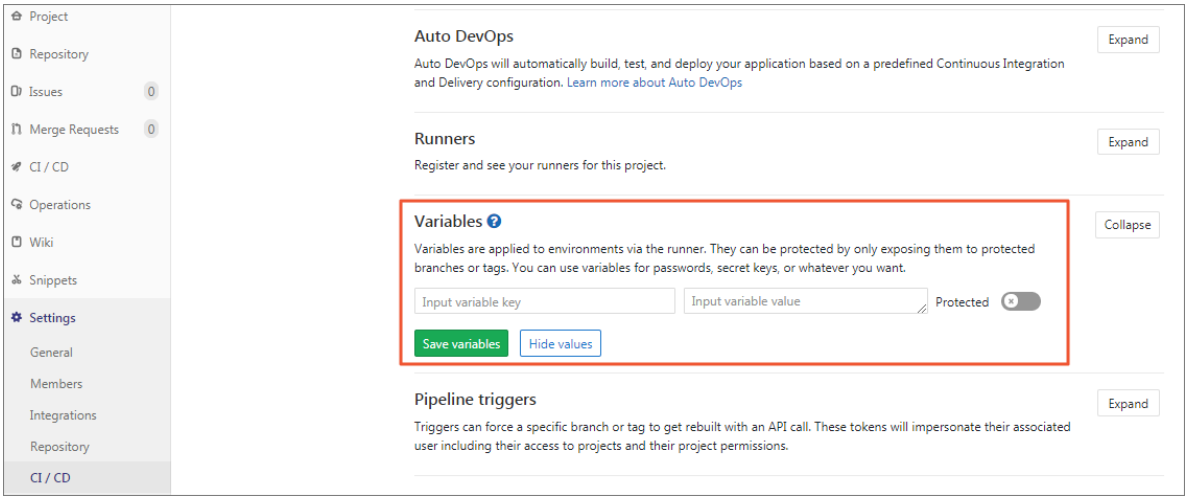
```
cat >>/home/gitlab-runner/.gitlab-runner/config.toml <<EOF
            [[runners.kubernetes.volumes.pvc]]
              name = "gitlab-runner-cache"
              mount_path = "{{ .Values.runners.cachePath }}"
EOF
```

即在GitLab Runner进行register后，run之前，修改*config.toml*的配置。

设置全局变量

1. 在GitLab的顶部导航栏中，选择Projects > Your projects。
2. 在Your projects页签下，选择相应的project。
3. 在左侧导航栏中，选择Settings > CI / CD。

4. 单击Variables右侧的Expand。添加GitLab Runner可用的环境变量。本示例中，添加以下三个变量：



- REGISTRY_USERNAME：镜像仓库用户名。
- REGISTRY_PASSWORD：镜像仓库密码。
- kube_config：KubeConfig的编码字符串。

执行以下命令生成KubeConfig的编码字符串：

```
echo $(cat ~/.kube/config | base64) | tr -d " "
```

编写.gitlab-ci.yml

编写*.gitlab-ci.yml*文件，完成java demo源码项目的编译构建、镜像推送和应用部署（可参考gitlabci-java-demo源码项目中的.gitlab-ci.yml.example）。

*.gitlab-ci.yml*示例如下：

```
image: docker:stable
stages:
  - package
  - docker_build
  - deploy_k8s
variables:
  KUBECONFIG: /etc/deploy/config
mvn_build_job:
  image: registry.cn-beijing.aliyuncs.com/codepipeline/public-
blueocean-codepipeline-slave-java:0.1-63b99a20
  stage: package
  tags:
    - k8s-test
  script:
    - mvn package -B -DskipTests
    - cp target/demo.war /opt/cache
docker_build_job:
  image: registry.cn-beijing.aliyuncs.com/codepipeline/public-
blueocean-codepipeline-slave-java:0.1-63b99a20
  stage: docker_build
  services:
```

```
    - docker:dind
  variables:
    DOCKER_DRIVER: overlay
    DOCKER_HOST: tcp://localhost:2375
  tags:
    - k8s-test
  script:
    - docker login -u $REGISTRY_USERNAME -p $REGISTRY_PASSWORD
registry.cn-beijing.aliyuncs.com
    - mkdir target
    - cp /opt/cache/demo.war target/demo.war
    - docker build -t registry.cn-beijing.aliyuncs.com/gitlab-demo/
java-demo:$CI_PIPELINE_ID .
    - docker push registry.cn-beijing.aliyuncs.com/gitlab-demo/java-
demo:$CI_PIPELINE_ID
deploy_k8s_job:
  image: registry.cn-beijing.aliyuncs.com/codepipeline/public-
blueocean-codepipeline-slave-java:0.1-63b99a20
  stage: deploy_k8s
  tags:
    - k8s-test
  script:
    - mkdir -p /etc/deploy
    - echo $kube_config |base64 -d > $KUBECONFIG
    - sed -i "s/IMAGE_TAG/$CI_PIPELINE_ID/g" deployment.yaml
    - cat deployment.yaml
    - kubectl apply -f deployment.yaml
```

*.gitlab-ci.yml*定义了一个Pipeline，分三个阶段步骤执行：

```
image: docker:stable  # Pipeline中各个步骤阶段的构建镜像没有指定时，默认使用
docker:stable镜像
stages:
  - package                 # 源码打包阶段
  - docker_build          # 镜像构建和打包推送阶段
  - deploy_k8s            # 应用部署阶段
variables:
  KUBECONFIG: /etc/deploy/config   # 定义全局变量KUBECONFIG
```

- maven源码打包阶段：

```
mvn_build_job:     # job名称
  image: registry.cn-beijing.aliyuncs.com/codepipeline/public-
blueocean-codepipeline-slave-java:0.1-63b99a20  # 本阶段构建使用的构建镜
像
  stage: package      # 关联的阶段名称
  tags:                     # GitLab Runner的tag
    - k8s-test
  script:
    - mvn package -B -DskipTests  # 执行构建脚本
    - cp target/demo.war /opt/cache   # 构建物保存至缓存区
```

- 镜像构建和打包推送阶段：

```
docker_build_job:  # job名称
  image: registry.cn-beijing.aliyuncs.com/codepipeline/public-
blueocean-codepipeline-slave-java:0.1-63b99a20 # 本阶段构建使用的构建镜
像
  stage: docker_build  # 关联的阶段名称
  services:                  # 使用docker:dind 服务，要求GitLab
Runner Helm Chart中runners.privileged为true
```

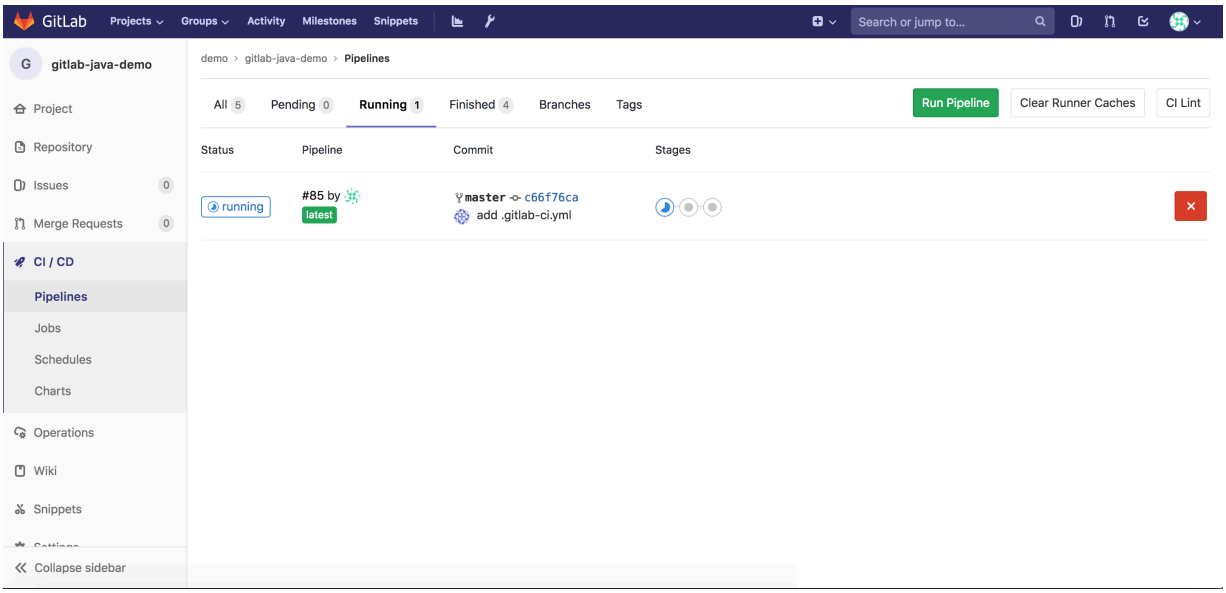
```
    - docker:dind
  variables:
    DOCKER_DRIVER: overlay
    DOCKER_HOST: tcp://localhost:2375   # 连接Docker Daemon
  tags:                         # GitLab Runner的tag
    - k8s-test
  script:
    - docker login -u REGISTRY_USERNAME -p $REGISTRY_PASSWORD
registry.cn-beijing.aliyuncs.com   # 登录镜像仓库
    - mkdir target
    - cp /opt/cache/demo.war target/demo.war
    - docker build -t registry.cn-beijing.aliyuncs.com/gitlab-demo
/java-demo:$CI_PIPELINE_ID .     # 打包Docker镜像，使用的tag为本次
Pipeline的ID
    - docker push registry.cn-beijing.aliyuncs.com/gitlab-demo/java-
demo:$CI_PIPELINE_ID      # 推送Docker镜像
```

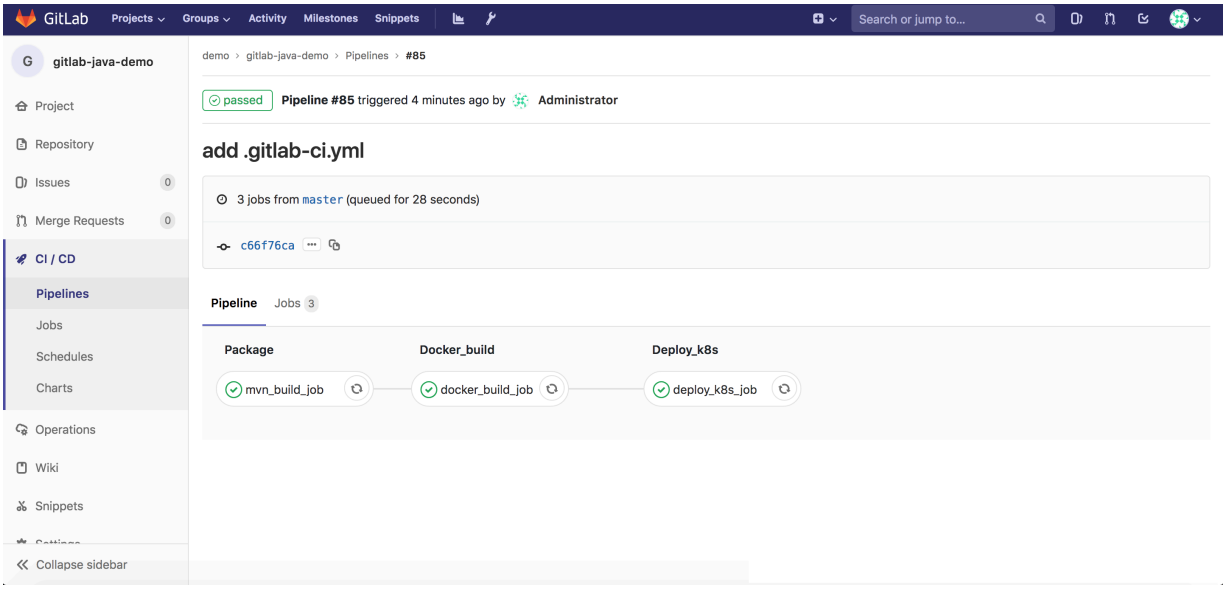
- 应用部署阶段：

```
deploy_k8s_job:   # job名称
  image: registry.cn-beijing.aliyuncs.com/codepipeline/public-
blueocean-codepipeline-slave-java:0.1-63b99a20   # 本阶段构建使用的构建
镜像
  stage: deploy_k8s   # 关联的阶段名称
  tags:                         # GitLab Runner的tag
    - k8s-test
  script:
    - mkdir -p /etc/deploy
    - echo $kube_config |base64 -d > $KUBECONFIG   # 配置连接
Kubernetes集群的config文件
    - sed -i "s/IMAGE_TAG/$CI_PIPELINE_ID/g" deployment.yaml  # 动态
替换部署文件中的镜像tag
    - kubectl apply -f deployment.yaml
```

执行Pipeline

提交*.gitlab-ci.yml*文件后，Project gitlab-java-demo会自动检测到这个文件并自行Pipeline，如下图所示：





访问服务

如果部署文件中没有指定namespace，则默认会部署到GitLab命名空间下：

```
$ kubectl -n gitlab get svc
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)
    AGE
java-demo    LoadBalancer   172.19.9.252   xx.xx.xx.xx   80:32349/TCP
  1m
```

浏览器访问xx.xx.xx.xx/demo进行验证。

了解更多容器服务的相关内容，请参见容器服务。

了解更多GitLab CI的相关内容，请参见*GitLab CI*。

7.4 在Serverless Kubernetes服务上快速搭建Jenkins环境及执行流水线构建

本文主要演示如何在阿里云Serverless Kubernetes服务上快速搭建Jenkins持续集成环境，并基于提供的示例应用快速完成应用源码编译、镜像构建和推送以及应用部署的流水线。

前提条件

您已经成功创建了一个Serverless Kubernetes集群。有关详细信息，请参见创建 *Serverless Kubernetes* 集群。

部署Jenkins

1. 执行以下命令下载部署文件：

```
$ git clone https://github.com/AliyunContainerService/jenkins-on-
serverless.git
```

```
$ cd jenkins-on-serverless
```

2. **完成jenkins_home持久化配置。**

   Serverless Kubernetes目前不支持云盘，如需持久化jenkins_home，您可以挂载nfs volume, 修改*serverless-k8s-jenkins-deploy.yaml*文件，注释掉以下字段并配置您的nfs信息：

```
#volumeMounts:
        #  - mountPath: /var/jenkins_home
        #    name: jenkins-home
      .....
      #volumes:
      #  - name: jenkins-home
      #    nfs:
      #      path: /
```

```
        #       server:
```

3. 执行以下命令部署Jenkins。

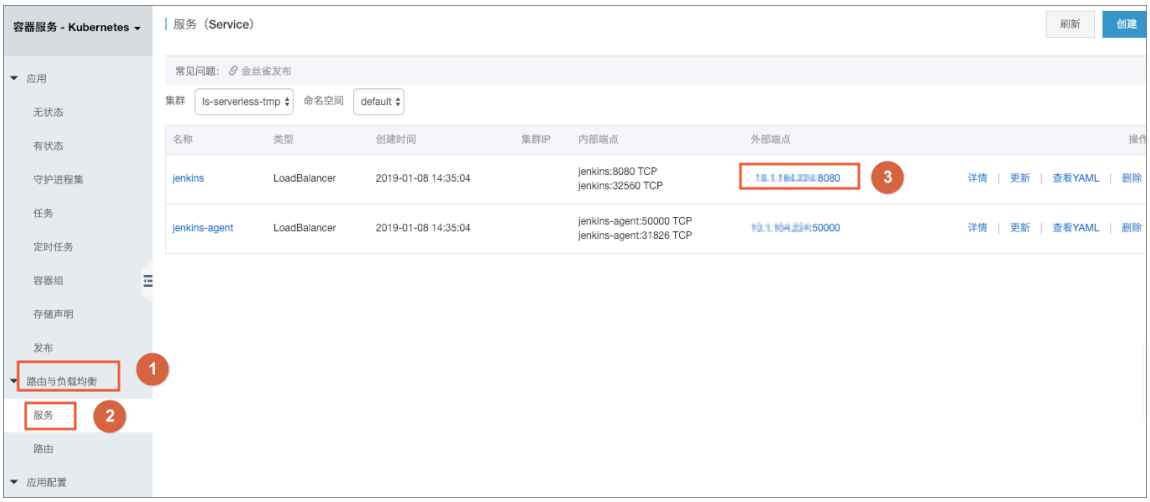
```
$ kubectl apply -f serverless-k8s-jenkins-deploy.yaml
```

4. 登录Jenkins。

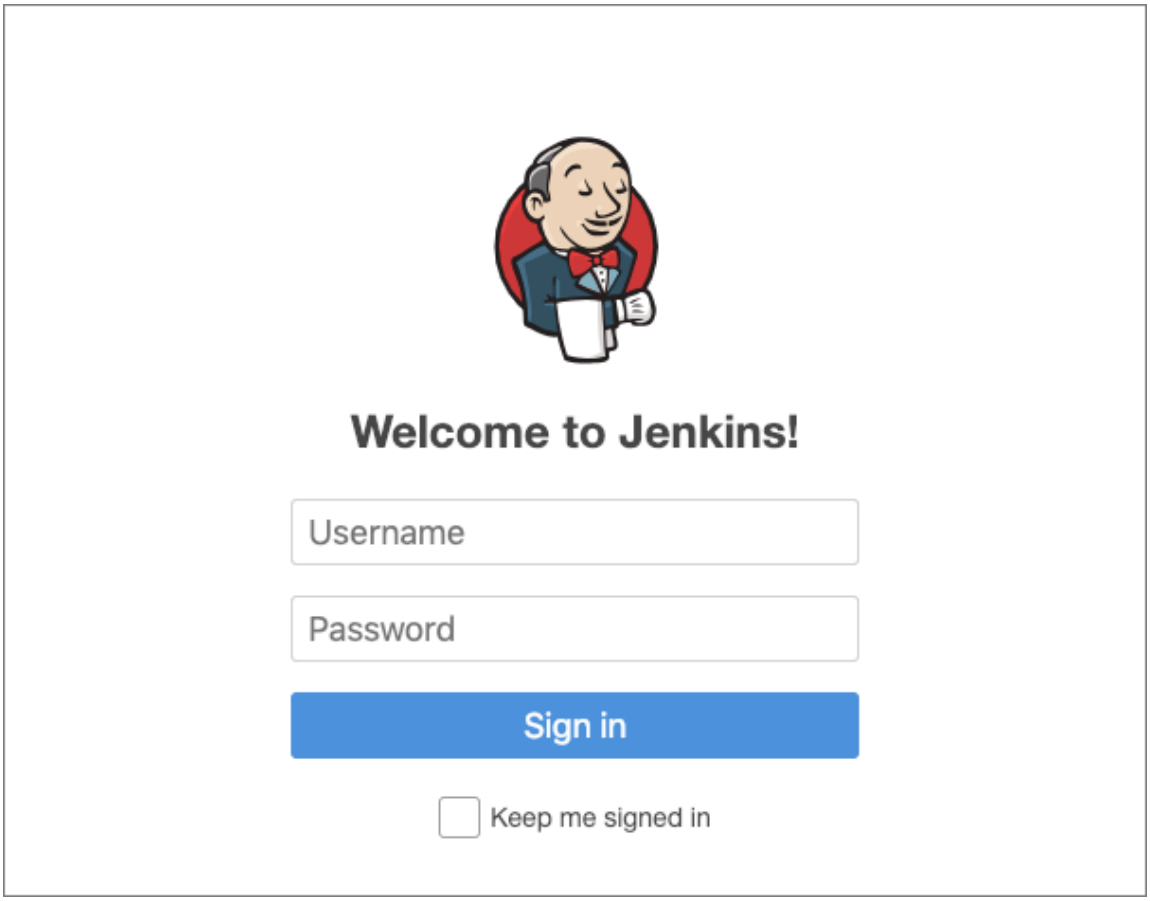
   a. 登录容器服务管理控制台。

   b. 在左侧导航栏中，选择路由与负载均衡 > 服务。

   c. 单击Jenkins服务的外部端点登录Jenkins。



   d. 在Jenkins登录页面，输入用户名和密码。默认用户名和密码均为admin，请于登录后进行修改。


Welcome to Jenkins!
Username
Password
Sign in
Keep me signed in

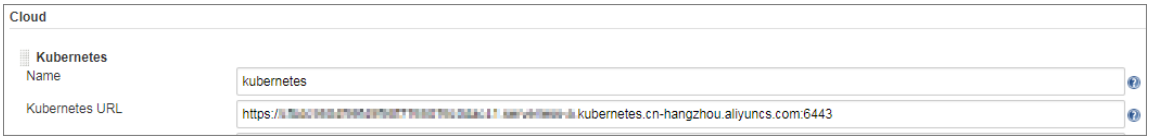
创建集群证书和镜像仓库证书，构建和部署示例应用

1. 配置Kubernetes Cloud，动态创建slave pod。

   a. 在左侧导航栏中，选择系统管理。

   b. 在右侧的Manage Jenkins页面，单击系统设置。

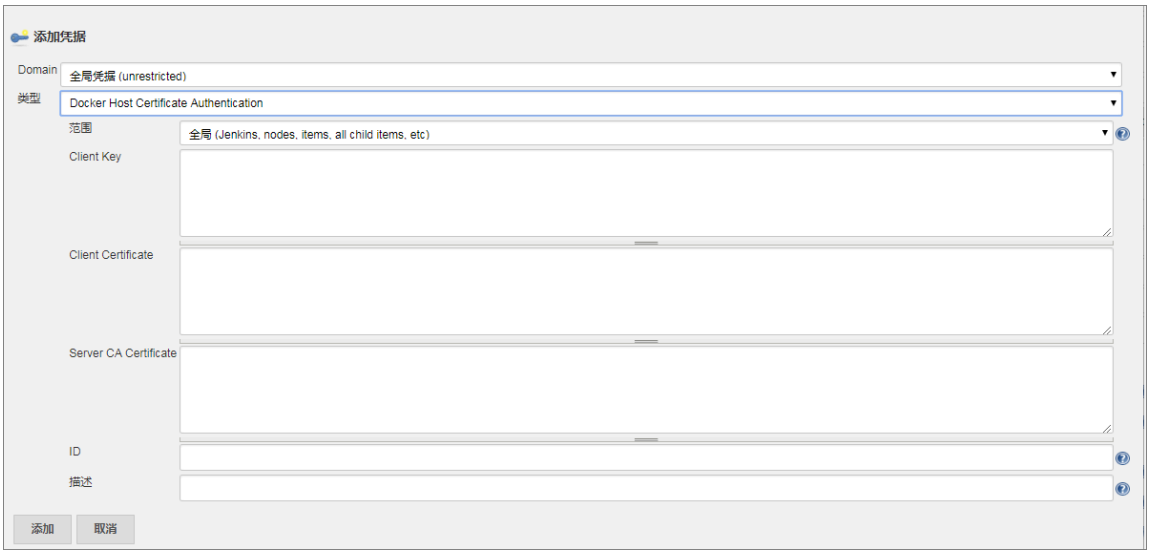
   c. 在Cloud区域中，填写KubeConfig中的API server URL作为Kubernetes URL。



   d. 单击Credentials右侧的Add。

   在添加凭据前，先在Serverless Kubernetes集群的基本信息页面，找到配置集群凭据中提供的KubeConfig。

   在弹出的添加凭据对话框中，完成以下配置：



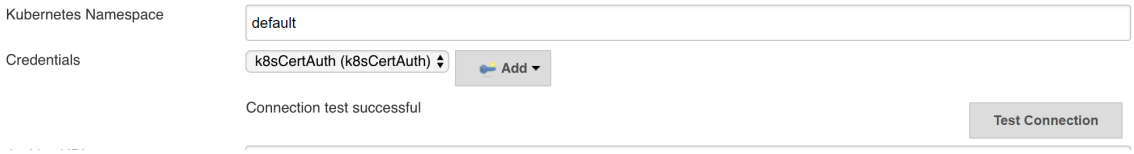
   · 类型：选择Docker Host Certificate Authentication。

- Client Key: 填写KubeConfig中client-key-data对应的内容。
- Client Certificate: 填写KubeConfig中client-certificate-data对应的内容。
- ID：填写证书ID。本示例中填写k8sCertAuth。
- 描述：填写描述。

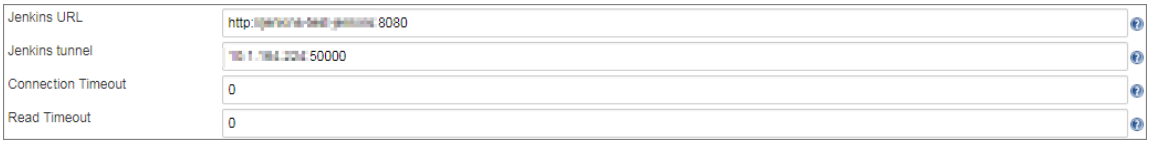
e. 单击添加。

f. 测试连通性。

从Credentials下拉框中选择上一步添加的凭据，单击Test Connection。



g. 填写jenkins服务的外部端点作为Jenkins URL，jenkins-agent的外部端点作为Jenkins tunnel。



h. 单击Save保存配置。

2. 通过kubectl在Serverless Kubernetes集群中创建jenkins-docker-cfg secret，用于设置镜像仓库权限。

本示例中使用阿里云镜像服务提供的北京区域镜像仓库：

```
$ docker login -u xxx -p xxx registry.cn-beijing.aliyuncs.com
Login Succeeded
```

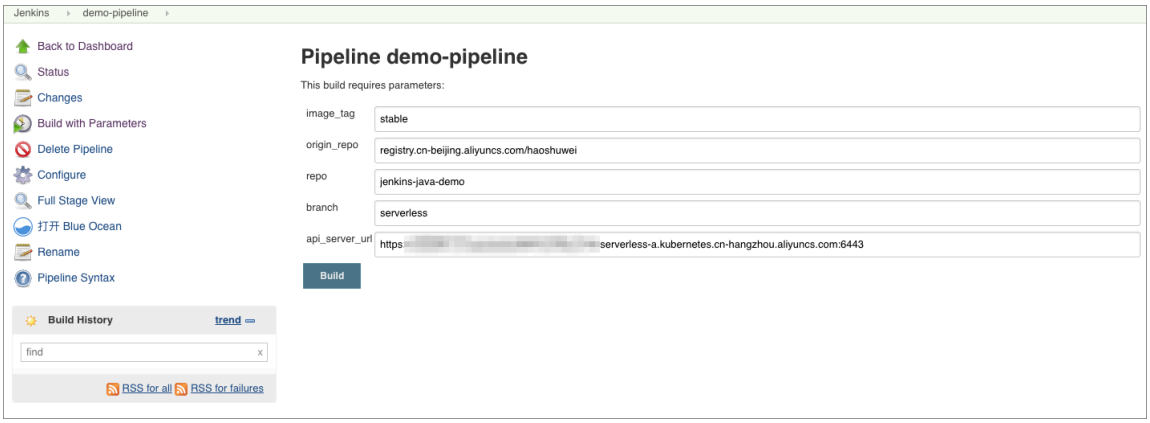
```
$ kubectl create secret generic jenkins-docker-cfg  --from-file=/
root/.docker/config.json
```

3. 构建demo-pipeline并访问应用服务。

   a. 在Jenkins首页，单击demo-pipeline。

   b. 在左侧导航栏中，选择Build with Parameters。

   c. 根据自己的镜像仓库信息修改构建参数，并填写KubeConfig中的API server URL作为api_server_url。本示例中源码仓库分支为serverless，镜像为registry.cn-beijing.aliyuncs.com/haoshuwei:stable。



   d. 单击Build。

   e. 查看Build History。下图表示构建成功。

   f. 构建成功后，登录容器服务管理控制台，查看应用的服务地址。

点击这里获取示例项目中使用的源码仓库。

7.5 通过CodePipeline构建项目并部署到Kubernetes集群

您可以通过CodePipeline，构建代码工作流模板、编译应用、构建与推送容器镜像、发布Kubernetes应用，从而实现利用代码发布应用的全过程自动化。本文以构建一个Java软件项目并将其部署到Kubernetes集群为例，说明如何使用CodePipeline。

CodePipeline简介

阿里云CodePipeline提供持续集成/持续交付的能力，并完全兼容Jenkins的能力与使用习惯，是一款SAAS化的产品。

其优点在于：

- 提供构建资源，无需用户运维，开箱即用；
- 与阿里云产品生态无缝集成；
- 兼容开源Jenkins使用习惯、轻量；
- 免费。

了解更多关于CodePipeline的内容，请参见*CodePipeline*产品帮助文档。

前提条件

使用CodePipeline之前，您需要先开通服务：

1. 登录*CodePipeline*管理控制台并开通服务。
2. 同意RAM的CodePipeline角色授权。

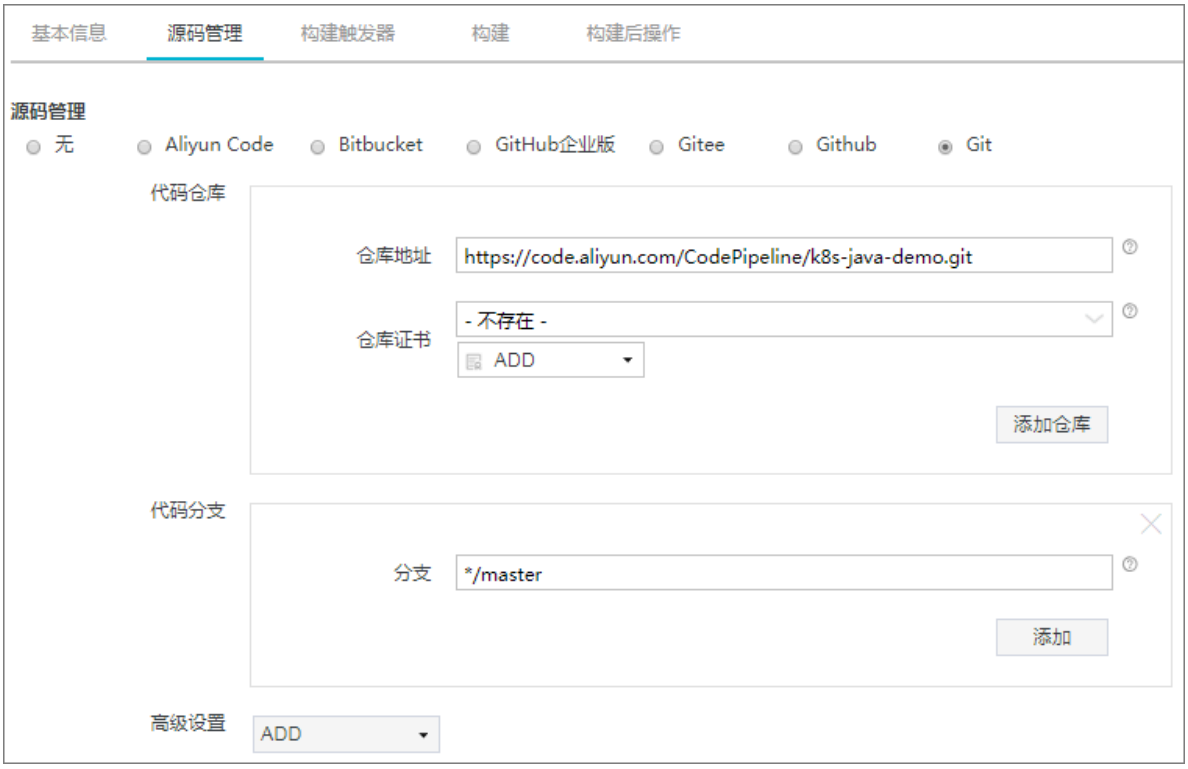
实施步骤

构建java-demo项目并部署到Kubernetes集群。

1. 登录*CodePipeline*管理控制台。
2. 单击右上角的新建。
3. 在新建项目页面，输入项目名称，选择构建一个自由风格的软件项目，并单击下一步。

4. 在基本信息区域，选择构建节点。本示例中选择国内节点和java构建环境（缓存）。

5. 在源码管理区域，配置源码仓库URL以及分支信息。本示例选择Git，并且使用源码项目`https://code.aliyun.com/CodePipeline/k8s-java-demo.git`。



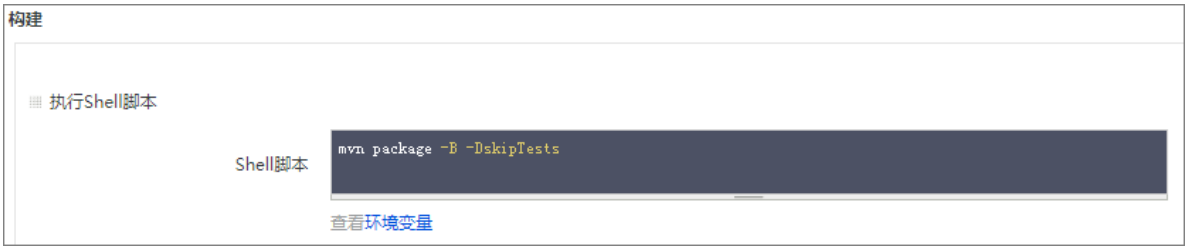
说明:

如果是私有仓库，您需要添加用户名密码类型证书。

6. 配置java-demo项目源码编译命令。

   在构建区域的增加构建步骤下拉框中，选择执行Shell脚本，并输入以下构建命令：

```
mvn package -B -DskipTests
```



7. 注入环境变量IMAGE_TAG。

   a. 在增加构建步骤下拉框中选择执行Shell脚本，并配置以下命令。通过该命令，您可以生成变量文件并写入变量的键值对。

```
TIME=`date +%Y%m%d%H%M%S`
```

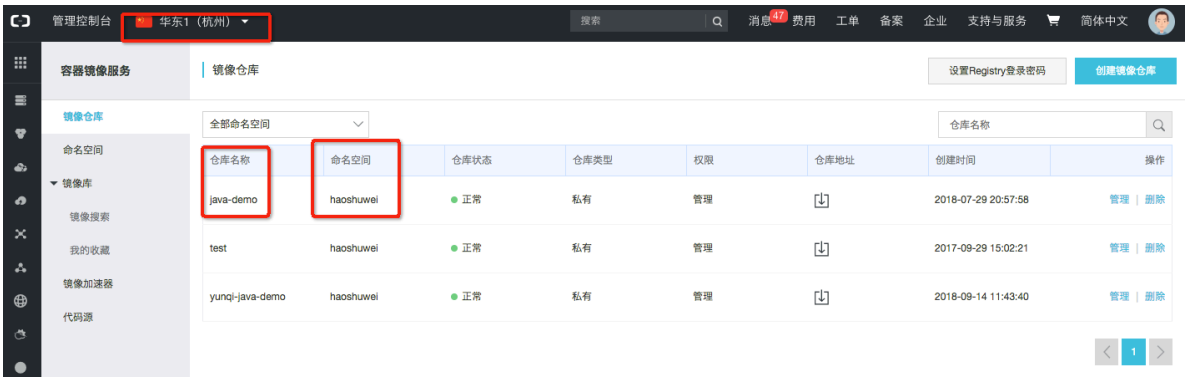
```
echo IMAGE_TAG=$TIME >> env.properties
```

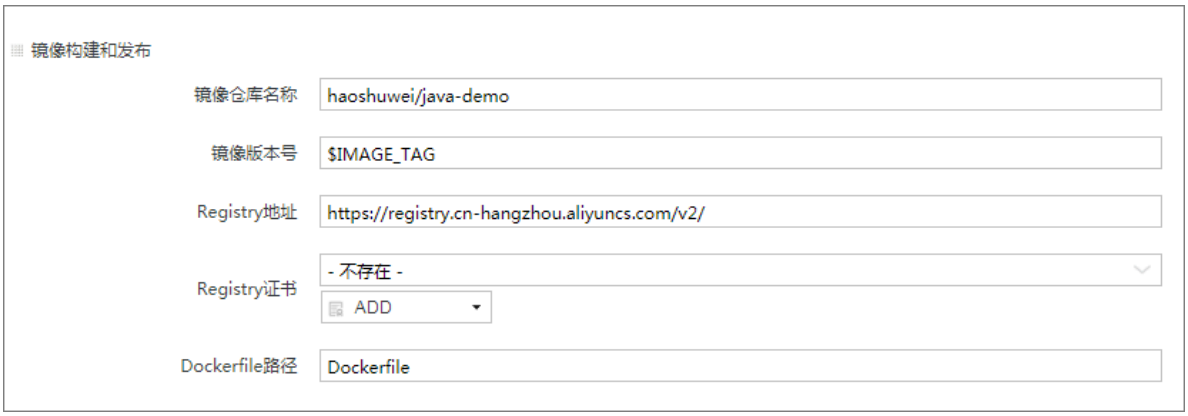
b. 在增加构建步骤下拉框中选择注入环境变量，并填写变量文件路径。

8. 配置Docker镜像构建并推送至私有镜像仓库。

在增加构建步骤下拉框中选择镜像构建和发布，并填写相关信息。

如果您需要把新构建的镜像推送至下图所示的私有镜像仓库，请完成以下参数配置。





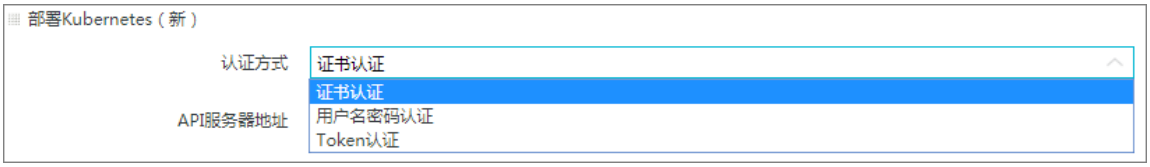
- 镜像仓库名称：填写镜像仓库名称，格式为命名空间/仓库名称。本示例中，填写haoshuwei/java-demo。
- 镜像版本号：填写镜像版本号，例如$IMAGE_TAG。
- Registry地址：填写Registry地址，例如杭州区域的Registry地址`https://registry.cn-hangzhou.aliyuncs.com/v2/`。
- Registry证书：创建Registry类型的证书。
- Dockerfile路径：填写Dockerfile路径。该路径必须位于源码项目的根目录下。

9. 配置部署Kubernetes应用。

在增加构建步骤下拉框中选择部署Kubernetes（新），并填写相关信息。

a. 选择认证方式。

CodePipeline目前支持证书认证、用户名密码认证和Token认证三种认证方式。Kubernetes集群默认选择证书认证。

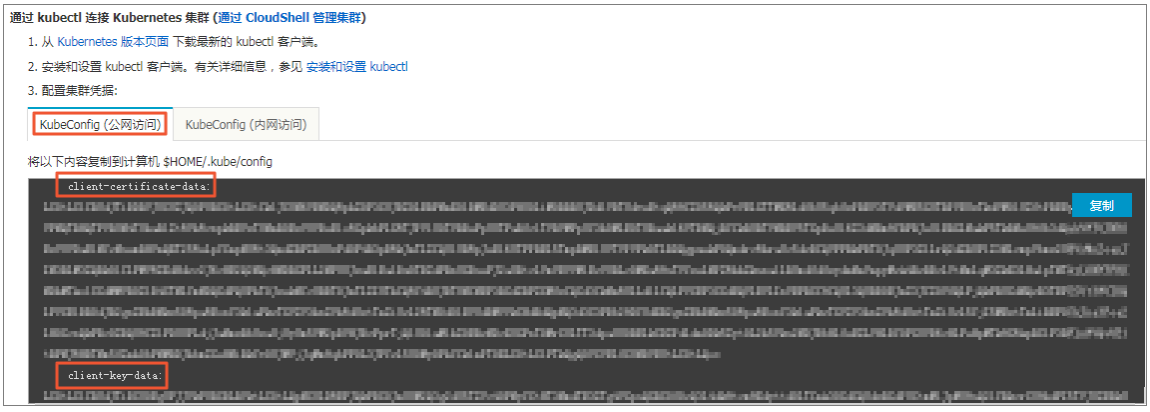


b. 填写API服务器地址。

填写Kubernetes集群的API服务器地址。您可以在容器服务管理控制台查看Kubernetes集群的API Server公网连接端点，例如`https://1.12.123.134:6443`。

c. 添加证书。

在添加证书之前，先在Kubernetes集群的基本信息页面，找到配置集群凭据中提供的KubeConfig。



在弹出的添加证书对话框中，完成以下配置：

- 类型：选择Docker授权。
- 客户端Key(key.pem)：填写KubeConfig中client-key-data对应的内容。
- 客户端证书(cert.pem)：填写KubeConfig中client-certificate-data对应的内容。

单击添加添加该证书。

d. 添加部署配置文件。

添加yaml格式的Kubernetes部署配置文件。如果该文件位于当前项目的workspace下，请直接输入文件名。如果位于当前项目workspace的子目录中，请输入 `../###/###`。不支持添加位于当前项目workspace之外的文件。

e. 添加状态检查配置。

支持检验的Kubernetes资源对象种类有：pods、daemonsets、deployments、replicasets、replicationcontrollers、statefulsets。如果检验的不是default命名空间

下的资源，请在首行填写命名空间名称。请用“:”分隔资源对象种类与资源对象名称。如果一种资源对象有多个名称，请用“,”分隔。每一行描述一种资源对象。

说明:

请严格按照说明填写单词，不要删减字母。请勿填写多余的空格或者换行。

f. 添加变量申明配置，声明需要使用的变量。

您可以在上文的部署配置文件*deployment.yaml*中使用变量。请以${IMAGE_TAG}的格式严格填写，其他格式插件会被忽略。

10.单击提交保存配置。

11.在左侧导航栏中，选择立即构建 > 开始构建，并查看构建日志。


```
d683471ab65a: Layer already exists
0f25831f224d: Layer already exists
923bc9a5d019: Layer already exists
185c62a48a71: Layer already exists
f715ed19c28b: Layer already exists
08a01612ffca: Layer already exists
59cd9510530c: Pushed

8bb25f9cdc41: Layer already exists
20181023032856: digest: sha256:6f6765e1199d7532e653611d13b12cbb7d4478d8546dd16e013413348af72feb size: 3046
----------------------------------------------------------------

Start variable Substition.

[YAD-PLUGIN] Injecting DOCKER_CONTAINER_ID variable.
[YAD-PLUGIN] Injecting JENKINS_CLOUD_ID variable.
[YAD-PLUGIN] DOCKER_HOST variable.
----------------------------------------------------------------

Apply config file.

kubectl apply -f /home/jenkins/workspace/java-demo/deployment.yaml --kubeconfig=/home/jenkins/workspace/java-demo/.kubeconfig
deployment "k8s-java-demo" created
service "k8s-java-demo" created
----------------------------------------------------------------

Start status check.

#Change namespace to: default
##Exec status check for: k8s-java-demo
kubectl get deployments --kubeconfig=/home/jenkins/workspace/java-demo/.kubeconfig --namespace=default
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
k8s-java-demo   2         2         2            0           1s
##Exec status check for: k8s-java-demo
kubectl get deployments --kubeconfig=/home/jenkins/workspace/java-demo/.kubeconfig --namespace=default
NAME           DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
k8s-java-demo   2         2         2            2           32s
----------------------------------------------------------------

Start list nodes.

    addresses:
    - address: 192.168.0.33
    addresses:
    - address: 192.168.0.31
    addresses:
    - address: 192.168.0.32
    addresses:
    - address: 192.168.0.34
Finished: SUCCESS
```

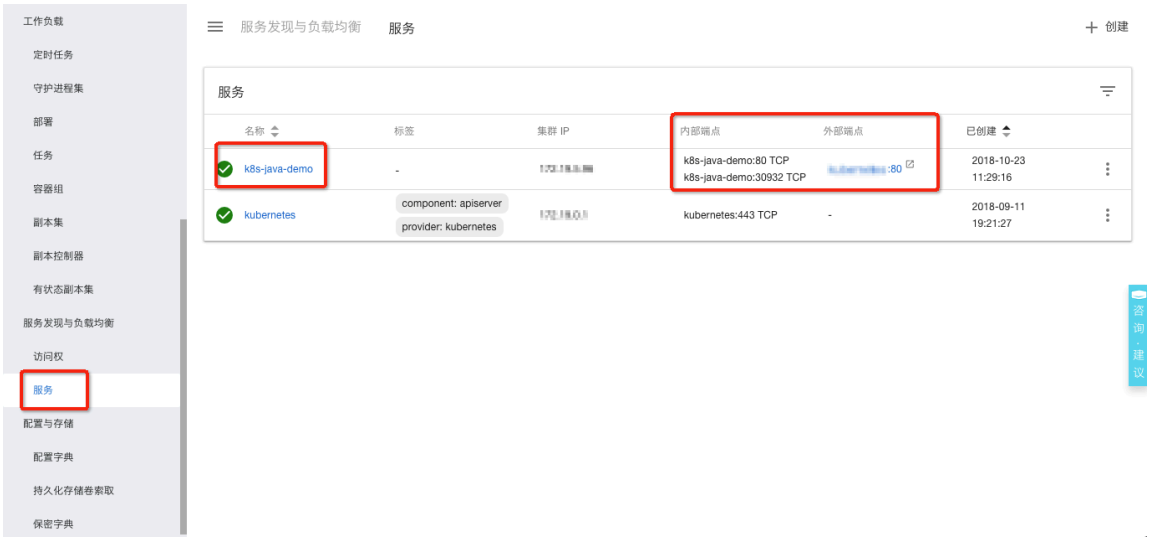
12.在Kubernetes集群控制台查看并访问服务。

a. 登录容器服务管理控制台。

b. 单击集群右侧的控制台，进入 Kubernetes集群控制台页面。

c. 在左侧导航栏中，选择服务发现与负载均衡 > 服务。

d. 查看并访问部署了Java软件项目的服务。



了解更多CodePipeline的相关内容，请参见*CodePipeline*。

了解更多容器服务的相关内容，请参见容器服务。

7.6 Kubernetes基于云效的DevOps实践

7.6.1 概述

本文介绍容器服务Kubernetes如何基于云效实现Devops。

背景信息

云效，是一个面向阿里云开发者的持续交付解决方案，为云端开发者提供从：项目管理、代码托管、测试、发布的一站式研发云端开发协同解决方案。通过云效开发者可以快速的构建和发布镜像，并且通过持续交付流水线将应用发布到阿里云容器服务Kubernetes版。

开通云效服务

1. 登录云效管理控制台。
2. 输入您的工作邮箱地址，绑定邮箱。

3. 输入企业名称，创建一个新的企业。

7.6.2 关联Kubernetes集群

本文介绍如何在云效管理控制台关联Kubernetes集群。

前提条件

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

操作步骤

1. 在顶部菜单栏，单击创建的企业名称，在下拉菜单中，单击企业设置。

2. 单击导航栏容器服务账号，选择kubernetes集群证书导入。

3. 导入集群。可选择手动导入证书或阿里云k8s集群导入。

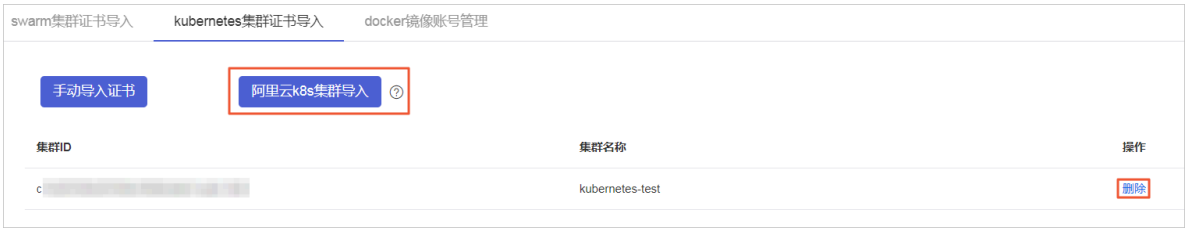
手动导入证书

- 集群名称：请填写待导入的集群名称。
- KubeConfig：请填写待导入集群的kubeconfig。

阿里云k8s集群导入

使用阿里云k8s集群导入功能，可自动导入k8s集群。

单击操作列的删除，可删除不需要展示的集群。



7.6.3 镜像仓库授权

本文介绍如何授权云效访问当前用户在阿里云镜像仓库中的镜像信息。

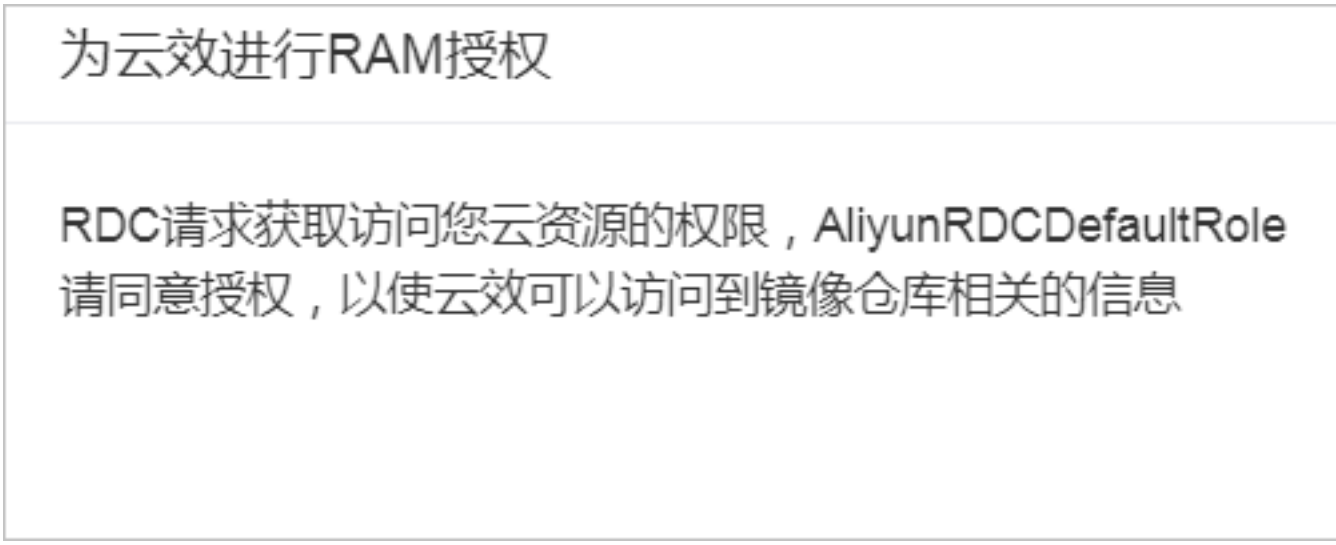
操作步骤

1. 单击导航栏容器服务账号，选择docker镜像账号管理页签。

2. 在RAM账号授权区域，单击右侧未授权，进入云资源访问授权页面。

3. 单击同意授权，进入为云效进行RAM授权页面。

4. 单击确认，授权云效可以访问当前用户在阿里云镜像仓库中的所有镜像信息。



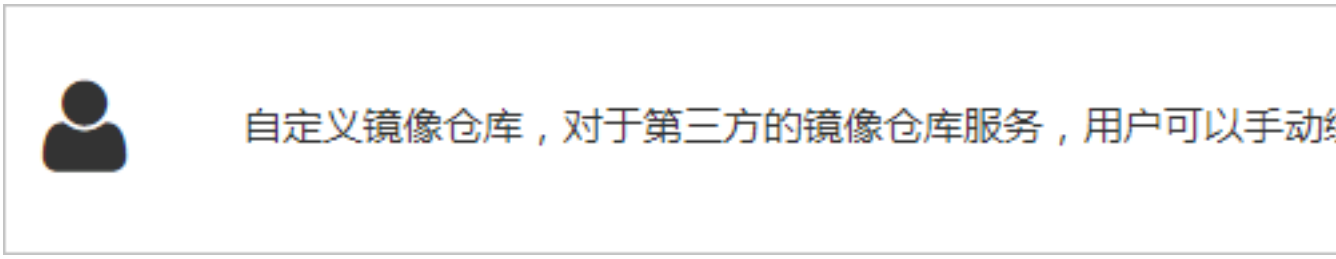
5. 单击RAM账号授权区域的已授权，点击完成绑定，进入绑定账号页面。

完成授权并绑定账号后，系统显示已绑定。

6. 若是获取第三方的镜像仓库服务，请在自定义镜像仓库区域，单击右侧未绑定。

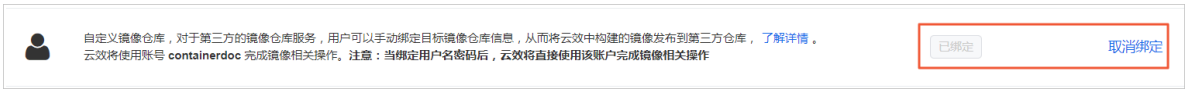
说明:

若您授权了RAM账号，同时绑定了第三方的镜像仓库，则绑定的第三方镜像仓库的优先级高。



7. 输入需要绑定镜像仓库的账号与密码。

绑定账号后，系统显示已绑定。



7.6.4 创建应用

本文介绍如何通过云效管理控制台创建应用。

操作步骤

1. 在顶部菜单栏选择首页，在一站式企业协同研发云区域，选择自定义配置。

说明:

也可通过快速开始创建，请参见云效快速入门。

2. 配置基本信息。

   · 应用名：请填写应用的名称。

   · 所属项目：请填写所属的项目名称。

   说明：

   - 若无项目，可单击所属项目文本框右侧点击新建项目。
   - 若已有项目，可单击所属项目文本框的下拉箭头，选择目标项目。

   · 开发模式：选择开发模式，具体模式的内容请参考自由模式和分支模式。

3. 单击下一步。

4. 配置代码库。

   · 代码源：自由模式支持两种代码库：Aliyun及码云。分支模式仅支持一种代码库Aliyun。

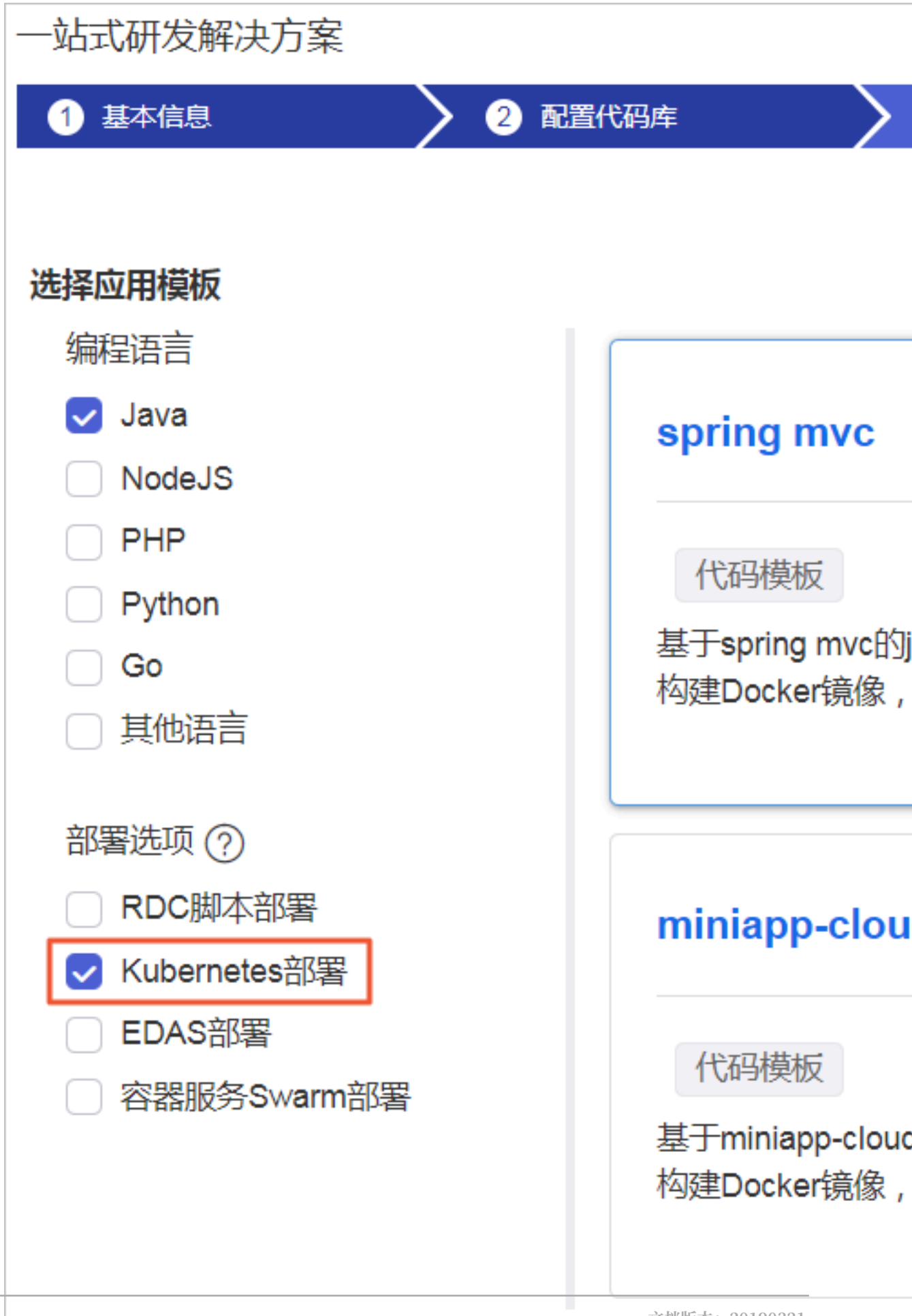
   说明：

在使用码云做为代码库时，需进行授权操作。授权后即可同步码云的数据。码云相关内容请参考码云 Gitee - 云端软件开发协作平台。

- 仓库：仓库可选择新建或关联已有。
- Git组/库：仓库选择新建时，需要填写此项。
  - Git库所在的组：可选择已有的组，也可点击新建Git组。
  - Git库名：请填写Git库的名称。
- Git库地址：仓库选择关联已有，需要填写此项。请填写已有代码库的URL，例如：`git@example.aliyun.com:ns/repo.git`。

5. 单击下一步。

6. 配置应用模板。单击左侧导航栏选择应用模板 > 部署选项 > Kubernetes部署，单击目标应用模板，本文以spring mvc为例，单击下一步。


一站式研发解决方案
1 基本信息
2 配置代码库
选择应用模板
编程语言
Java
NodeJS
PHP
Python
Go
其他语言
部署选项
RDC脚本部署
Kubernetes部署
EDAS部署
容器服务Swarm部署
spring mvc
代码模板
基于spring mvc的j
构建Docker镜像，
miniapp-clou
代码模板
基于miniapp-cloud
构建Docker镜像，

7. 配置构建信息。

   - Docker构建：根据源码库中的Dockerfile自动构建镜像并推送到容器镜像仓库。
     - 阿里云镜像仓库：若您关联了阿里云容器镜像服务，可以直接选择已有的镜像仓库。并填写镜像区域和镜像仓库。
     - 第三方仓库：库的镜像地址，例如：*registry.cn-hangzhou.aliyuncs.com/namespace/repo*。
   - 自定义构建镜像：您可以使用自定义镜像作为项目的构建时环境，请参考自定义构建镜像。
   - 构建配置：您可以预览已创建应用的构建配置信息，具体字段解释，请参考Web应用构建配置。

8. 单击下一步。

9. 应用信息预览，查看前面几个步骤的配置信息：

   · 若需要修改，请单击步骤名称后面的  ，进入相应页面修改。

   · 若不需要修改，单击确定创建应用。

10.在完成界面，显示创建代码库及创建应用的部署进度。

预期结果

在完成界面，显示查看应用。

7.6.5 流水线及构建

本文主要介绍流水线，以及如何进行构建。

流水线概述

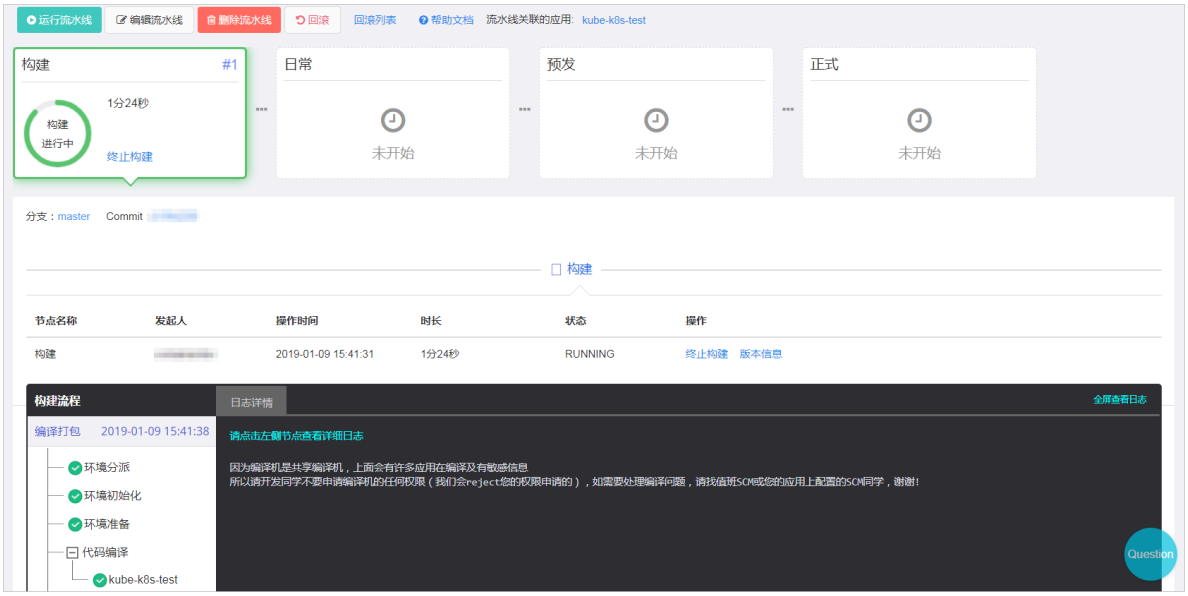
云效管理控制台会为应用创建一个持续交付的流水线。

默认情况下，一条流水线包含4个默认阶段：

- 构建：完成项目的编译、打包、镜像构建以及发布。
- 日常：应用的日常测试环境部署阶段。
- 预发：应用的预发环境部署阶段。
- 正式：应用的正式环境部署阶段。

构建

1. 在顶部菜单栏选择项目，单击左侧导航栏流水线。
2. 选择目标流水线，单击流水线名称。
3. 单击构建区域的开始，即开始进行构建。



执行结果

构建区域，显示构建已完成。

7.6.6 部署环境的发布策略

本文介绍如何部署环境以及发布策略。

背景信息

您已成功部署一条流水线，并完成构建，可参考流水线及构建。

操作步骤

1. 单击左侧导航栏应用，进入我的应用页面。
2. 选择目标应用，单击应用名称。
3. 单击顶部菜单栏环境。

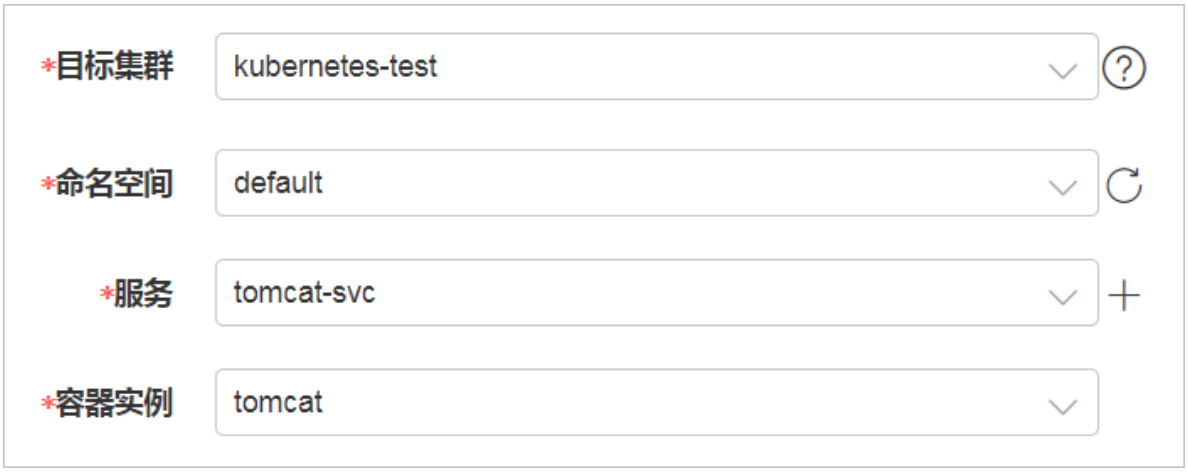
4. 在环境列表页面，单击日常环境的部署配置。

5. 在部署配置页面，修改部署方式选择为Kubernetes部署。

6. 选择目标集群集群、命名空间、服务及容器实例。

说明：

- 若没有集群，可参考关联Kubernetes集群，导入集群。
- 您可以创建新的服务，或选择已有的服务。若新建服务，部署时会自动创建Deployment资源。若选择已有服务，需要填写容器实例名称。



7. 配置发布策略，有3种方式：

- 分批发布，第一批暂停：对于预发环境和生产环境安全要求较高的场景，可以使用分批发布，第一批暂停。此时需要设置发布批次。具体内容请参考Kubernetes分批发布。

- 蓝绿部署：对于已经部署Istio功能的集群，云效会自动打开蓝绿部署选项，具体可参考基于Istio的蓝绿发布。
- Kubernetes滚动升级：滚动升级使用Kubernetes原生的RollingUpdate方式。

8. 参考以上步骤，部署预发环境及正式环境。

7.6.7 运行流水线

本文主要介绍如何运行流水线。

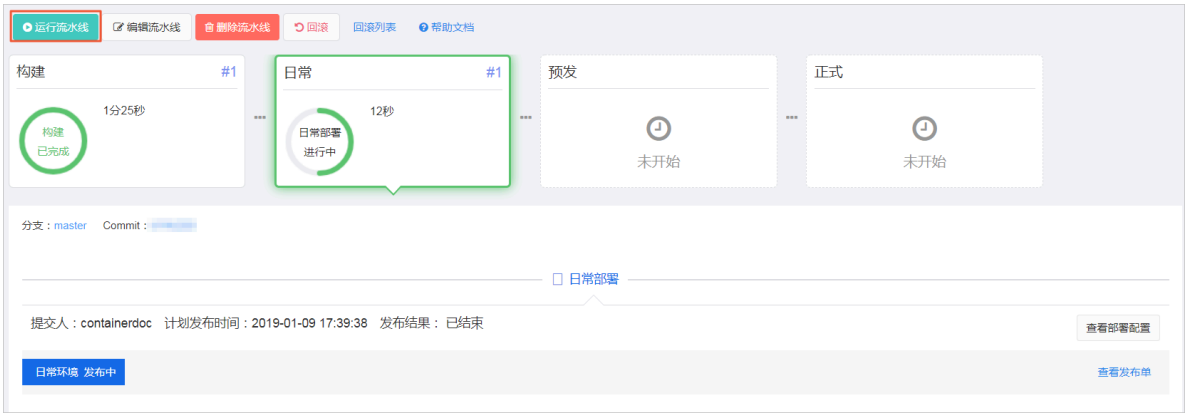
背景信息

日常环境、预发环境及正式环境已完成配置部署，可参考部署环境的发布策略。

操作步骤

1. 在顶部菜单栏选择项目，单击左侧导航栏流水线。

2. 单击运行流水线，启动对应用的构建及各个环境的自动化部署。



说明:

当环境关联的Service还未关联任何的Deployment资源时，云效会自动创建相应的Deployment资源。

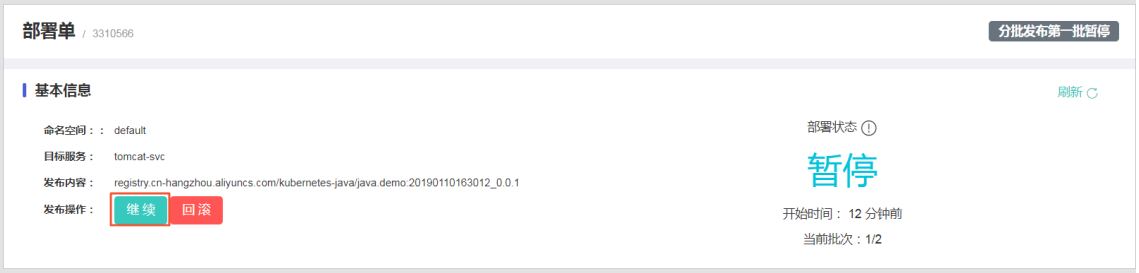
当环境关联的Service存在关联的Deployment资源时：

- 对于Kubernetes滚动升级：云效会自动替换镜像版本。

- 对于分批发布，第一批暂停：云效会为Service创建两个版本的Deployment实例，并且按照批次进行升级，并在最后一次的批次执行中自动清理老版本的Deployment资源。如果在发布过程中出现问题，则可以通过回滚操作回到发布前状态。

  在日常区域显示日常部署等待中时，请点击查看发布单。发布单的详细内容，请参考*Kubernetes*分批发布。

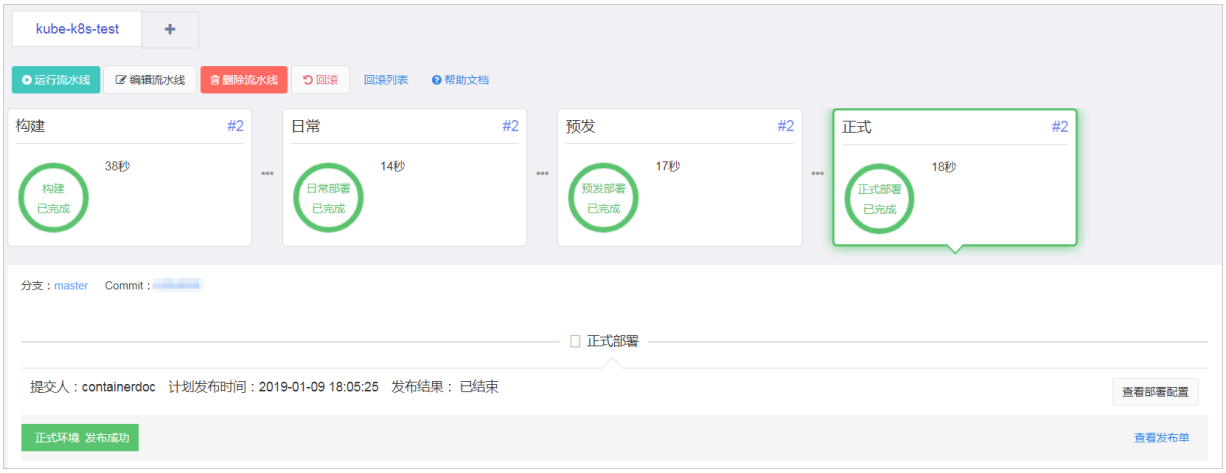
  在部署单页面，单击基本信息区域的继续。



- 对于蓝绿部署：请参考基于*Istio*的蓝绿发布。

预期结果

- 构建：显示构建已完成。
- 日常：显示日常部署已完成。
- 预发：显示预发部署已完成。
- 正式：显示正式部署已完成。


kube-k8s-test
运行流水线
编辑流水线
删除流水线
回滚
回滚列表
帮助文档
构建 #2
38秒
构建 已完成
日常 #2
14秒
日常部署 已完成
预发 #2
17秒
预发部署 已完成
正式 #2
18秒
正式部署 已完成
分支：master Commit：
正式部署
提交人：containerdoc 计划发布时间：2019-01-09 18:05:25 发布结果： 已结束
查看部署配置
正式环境 发布成功
查看发布单