阿里云
容器服务Kubernetes版

最佳实践

文档版本：20190814

为了无法计算的价值 | 阿里云

法律声明

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

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

通用约定

格式	说明	样例
	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

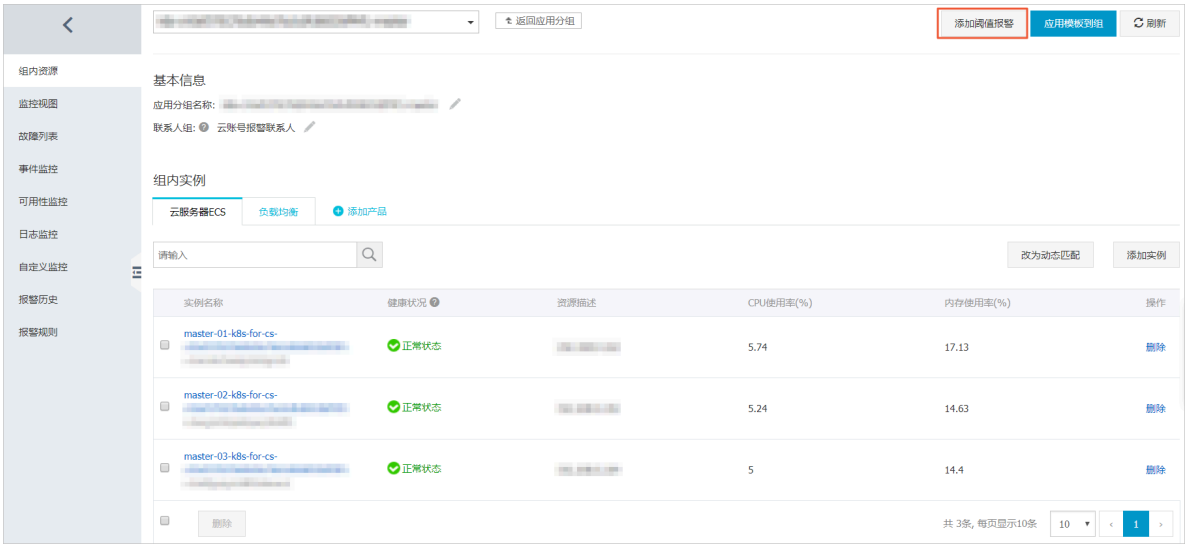
阿里云容器服务与云监控进行集成，通过配置节点监控，提供实时的监控服务。通过添加监控告警规则，节点上的资源使用量很高的时候，可快速定位问题。

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#unique_11。
- 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@[redacted] ~]# kubectl get nodes
NAME                                STATUS    ROLES     AGE       VERSION
cn-hangzhou.[redacted]              Ready     <none>    23d       v1.11.2
cn-hangzhou.[redacted]              Ready     <none>    23d       v1.11.2
cn-hangzhou.[redacted]              Ready     master    47d       v1.11.2
cn-hangzhou.[redacted]              Ready     master    47d       v1.11.2
cn-hangzhou.[redacted]              Ready     master    47d       v1.11.2
cn-hangzhou.[redacted]              Ready     <none>    47d       v1.11.2
cn-hangzhou.[redacted]              Ready     <none>    47d       v1.11.2
[root@[redacted] ~]#
```

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@[redacted] ~]# 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
```

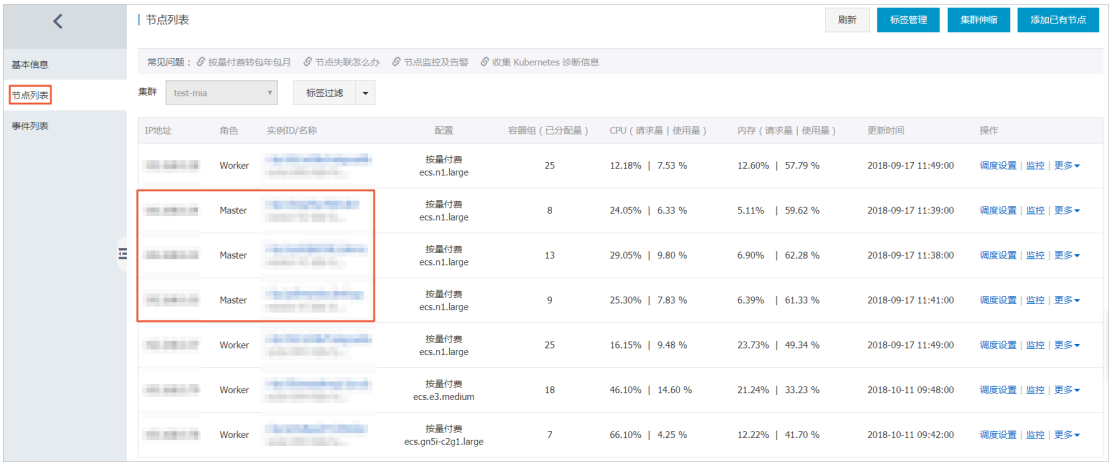
```
[root@                    ~]# kubectl get nodes
NAME                                STATUS    ROLES     AGE       VERSION
cn-hangzhou.i                       Ready     <none>    22d       v1.11.2
cn-hangzhou.i                       Ready     <none>    22d       v1.11.2
cn-hangzhou.i                       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 > 集群，进入集群列表页面。

  c. 选择目标集群，单击集群名称，查看集群详细信息，单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@[redacted] ~]# 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@[redacted] ~]#
```

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

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

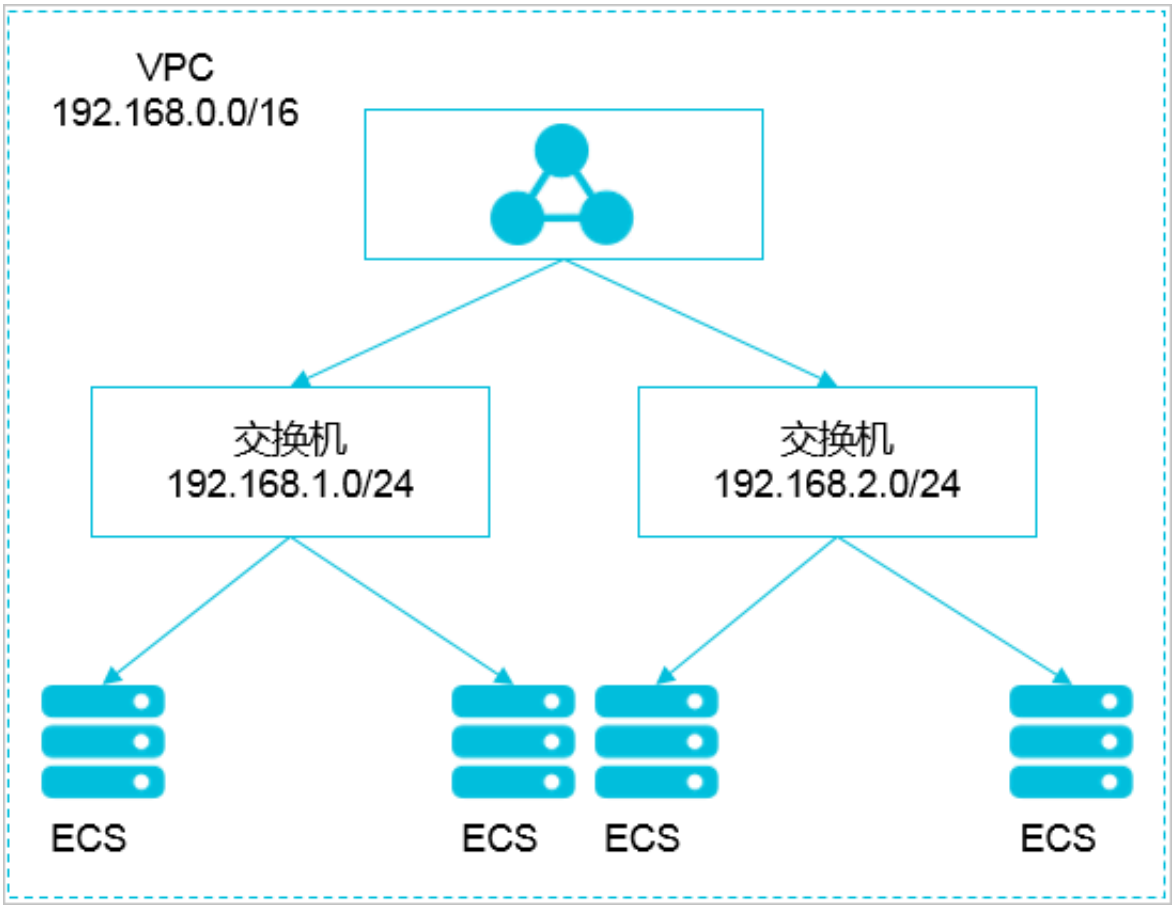
VPC 网段

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。



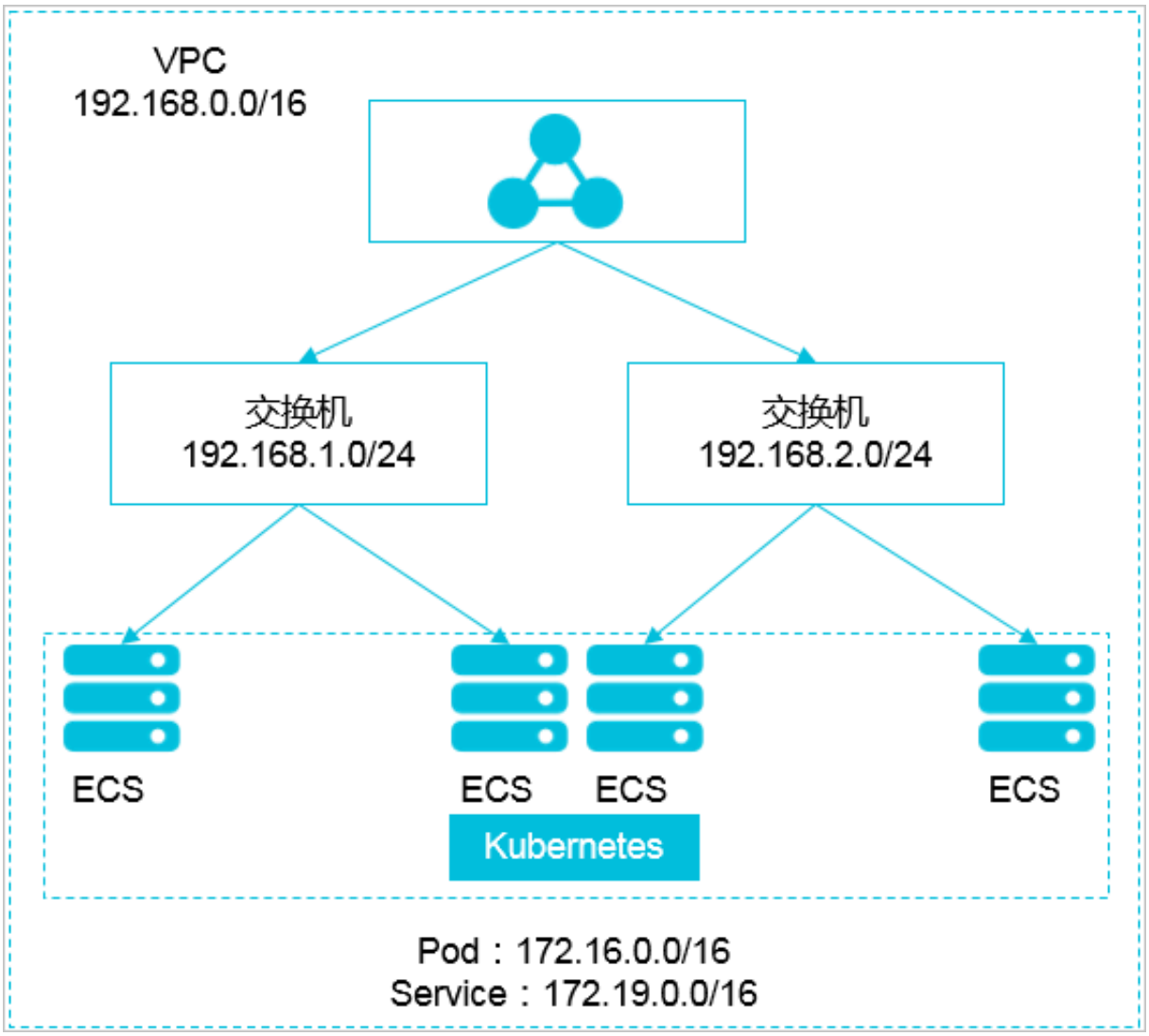
Pod 地址段

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

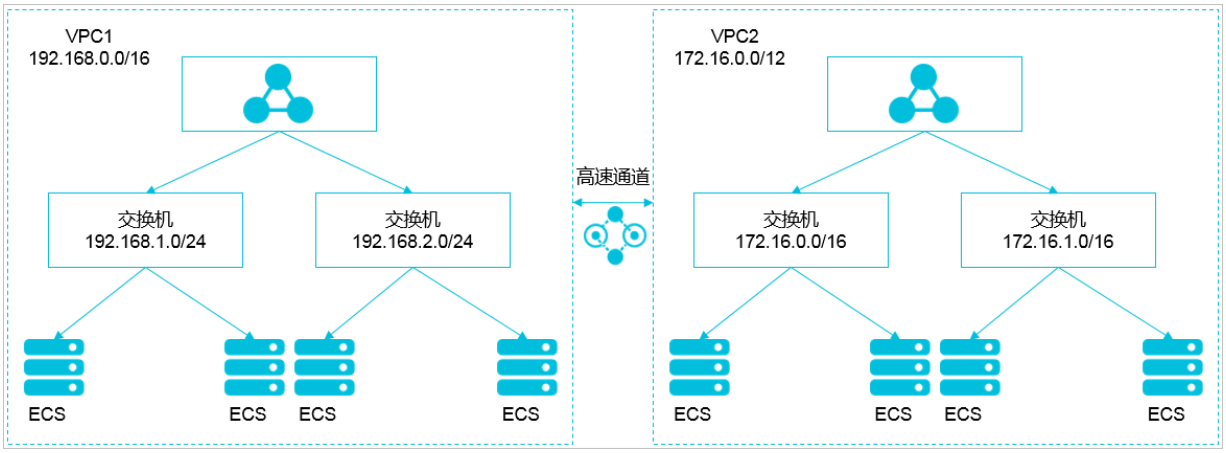
一个 VPC 下创建多个 Kubernetes 集群。在默认的网络模式下（Flannel），Pod 的报文需要通过 VPC 路由转发，容器服务会自动在 VPC 路由上配置到每个 Pod 地址段的路由表。所有 Kubernetes 集群的 Pod 地址段不能重叠，但 Service 地址段可以重叠。

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。

1.5 使用自定义镜像创建Kubernetes集群

随着云原生时代的到来，用户应用、业务上云的需求也越来越多，不同的业务场景对容器平台的需求也不尽相同，其中一个非常重要的需求就是使用自定义镜像创建Kubernetes集群。

背景信息

容器服务支持用户使用自定义镜像创建Kubernetes集群，但用户在制作打包自定义镜像时，往往会遇到以下情况：

- 人工操作步骤，效率低
- 镜像变更历史记录缺失，不便于故障定位
- 无法对自定义镜像进行校验并判断是否符合Kubernetes集群节点要求

基于以上痛点，我们开源了ack-image-builder项目帮助用户快速制作符合ACK集群节点要求的自定义镜像。

ack-image-builder 项目基于开源工具HashiCorp Packer，提供默认配置模板和校验脚本。

使用ACK-image-builder项目创建kubernetes集群自定义节点镜像的步骤如下。

操作步骤

1. 安装Packer。

   从官方下载页面选择操作系统对应的软件版本，并按照安装说明文档安装和验证packer。

   执行如下命令，显示结果如下时，说明packer已安装成功。

```
$ packer version
Packer v1.4.1
```

2. 定义Packer模板。

   使用Packer创建自定义镜像时，需要创建一个JSON格式的模板文件。在该模板文件中，您需要指定创建自定义镜像的Alicloud Image Builder（生成器）和Provisioners（配置器）。

```
{
  "variables": {
    "region": "cn-hangzhou",
    "image_name": "test_image{{timestamp}}",
    "source_image": "centos_7_06_64_20G_alibase_20190711.vhd",
    "instance_type": "ecs.n1.large",
    "access_key": "{{env `ALICLOUD_ACCESS_KEY`}}",
    "secret_key": "{{env `ALICLOUD_SECRET_KEY`}}"
  },
```

```
  "builders": [
    {
      "type": "alicloud-ecs",
      "access_key": "{{user `access_key`}}",
      "secret_key": "{{user `secret_key`}}",
      "region": "{{user `region`}}",
      "image_name": "{{user `image_name`}}",
      "source_image": "{{user `source_image`}}",
      "ssh_username": "root",
      "instance_type": "{{user `instance_type`}}",
      "io_optimized": "true"
    }
  ],
  "provisioners": [
    {
      "type": "shell",
      "scripts": [
        "scripts/updateKernel.sh",
        "scripts/reboot.sh",
        "scripts/cleanUpKerneles.sh",
        "config/default.sh",
        "scripts/updateDNS.sh",
        "scripts/verify.sh"
      ],
      "expect_disconnect": true
    }
  ]
}
```

参数	描述
access_key	您的AccessKeyID
secret_key	您的AccessKeySecret
region	创建自定义镜像时使用临时资源的地域
image_name	自定义镜像的名称
source_image	基础镜像的名称，可以从阿里云公共镜像列表获得
instance_type	创建自定义镜像时生成的临时实例的类型
provisioners	创建自定义镜像时使用的 Packer 配置器类型

3. 创建子账号并生成AccessKey。

   制作自定义镜像的权限要求较大，一般建议用户创建子账户并授权Packer需要的对应RAM Policy并创建AccessKey。

4. 导入AccessKey信息并制作自定义镜像。

   a) 执行如下命令，导入AccessKey信息。

```
export ALICLOUD_ACCESS_KEY=XXXXXX
export ALICLOUD_SECRET_KEY=XXXXXX
```

   b) 执行如下命令，制作自定义镜像。

```
$ packer build alicloud.json
```

```
alicloud-ecs output will be in this color.

==> alicloud-ecs: Prevalidating source region and copied regions
...
==> alicloud-ecs: Prevalidating image name...
    alicloud-ecs: Found image ID: centos_7_06_64_20G_alibase_201
90711.vhd
==> alicloud-ecs: Creating temporary keypair: xxxxxx
==> alicloud-ecs: Creating vpc...
    alicloud-ecs: Created vpc: xxxxxx
==> alicloud-ecs: Creating vswitch...
    alicloud-ecs: Created vswitch: xxxxxx
==> alicloud-ecs: Creating security group...
    alicloud-ecs: Created security group: xxxxxx
==> alicloud-ecs: Creating instance...
    alicloud-ecs: Created instance: xxxxxx
==> alicloud-ecs: Allocating eip...
    alicloud-ecs: Allocated eip: xxxxxx
    alicloud-ecs: Attach keypair xxxxxx to instance: xxxxxx
==> alicloud-ecs: Starting instance: xxxxxx
==> alicloud-ecs: Using ssh communicator to connect: 47.111.127.54
==> alicloud-ecs: Waiting for SSH to become available...
==> alicloud-ecs: Connected to SSH!
==> alicloud-ecs: Provisioning with shell script: scripts/verify.
sh
    alicloud-ecs: [20190726 11:04:10]: Check if kernel version >=
3.10.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if systemd version
 >= 219.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if sshd is running
and listen on port 22.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if cloud-init is
installed.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if wget is installed.
  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if curl is installed.
  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubeadm is cleaned
 up.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubelet is cleaned
 up.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubectl is cleaned
 up.  Verify Passed!
    alicloud-ecs: [20190726 11:04:10]: Check if kubernetes-cni is
cleaned up.  Verify Passed!
==> alicloud-ecs: Stopping instance: xxxxxx
==> alicloud-ecs: Waiting instance stopped: xxxxxx
==> alicloud-ecs: Creating image: test_image1564110199
    alicloud-ecs: Detach keypair xxxxxx from instance: xxxxxxx
==> alicloud-ecs: Cleaning up 'EIP'
==> alicloud-ecs: Cleaning up 'instance'
==> alicloud-ecs: Cleaning up 'security group'
==> alicloud-ecs: Cleaning up 'vSwitch'
==> alicloud-ecs: Cleaning up 'VPC'
==> alicloud-ecs: Deleting temporary keypair...
Build 'alicloud-ecs' finished.

==> Builds finished. The artifacts of successful builds are:
--> alicloud-ecs: Alicloud images were created:

cn-hangzhou: m-bp1aifbnupnaktj00q7s
```

其中scripts/verify.sh为对检查项的校验部分。

5. 使用自定义镜像创建Kubernetes集群。

   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 > 集群，进入集群列表页面。

   c) 单击页面右上角的创建 Kubernetes 集群，在弹出的选择集群模板中，选择标准专有集群页面，单击创建。请参见创建Kubernetes 集群。

   d) 配置集群创建需要的相关基础信息后，单击显示高级选项，自定义镜像中单击选择，选择自定义镜像进行集群创建。

   e) 配置完成后，单击创建集群。

2 网络

2.1 部署高可靠 Ingress Controller

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

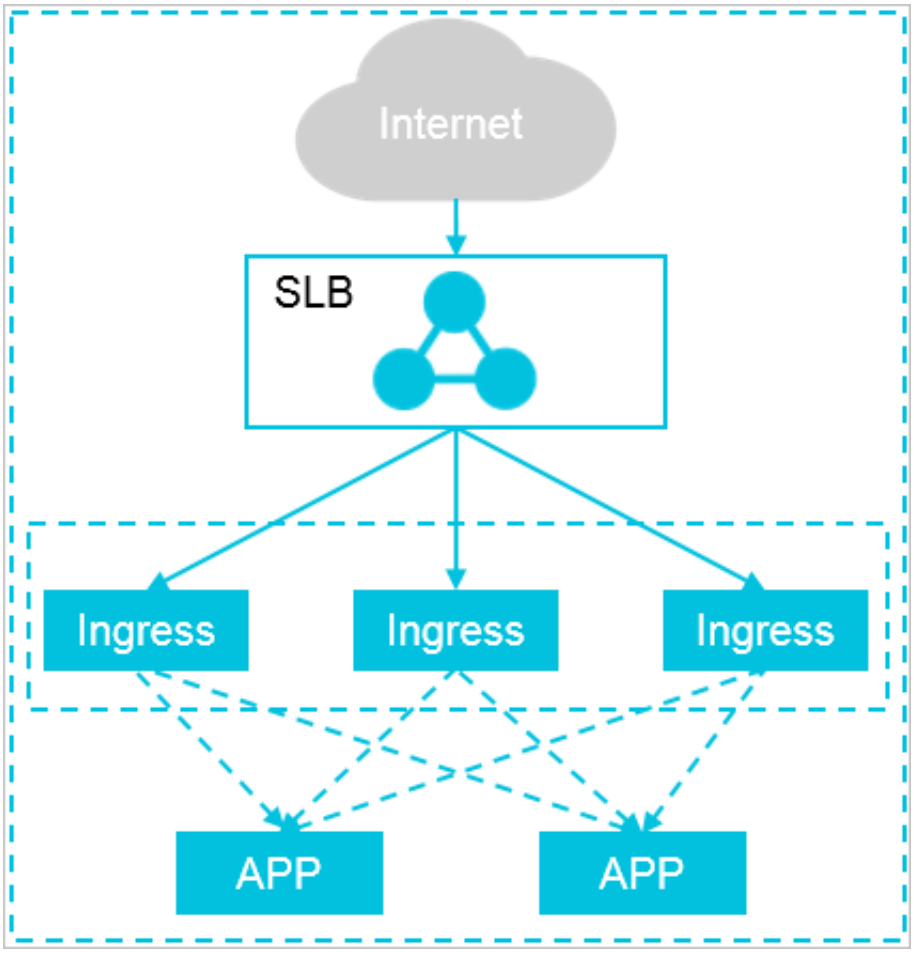
- 您已成功创建一个 Kubernetes 集群，参见#unique_11。
- SSH 连接到 Master 节点，参见SSH访问Kubernetes集群。

背景信息

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。

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，通常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副本及公网地址

当您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。

1. 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
```

2. 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 增加Replica数量

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

1. 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
```

2. 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

如果你需要Nginx Ingress Controller仅运行在指定的一些高配置节点上，可以通过给节点添加标签来完成。

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-bp11bcmsna8d4bp****   Ready    master   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24l****   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24l****   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24l****   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp181pofzyyksie****   Ready    master   21d   v1.11.5
cn-hangzhou.i-bp1cbsg6rf3580z****   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。

  查询结果中，ROLES的值显示为none，表示为Worker节点

```
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#unique_11。
- 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:
```

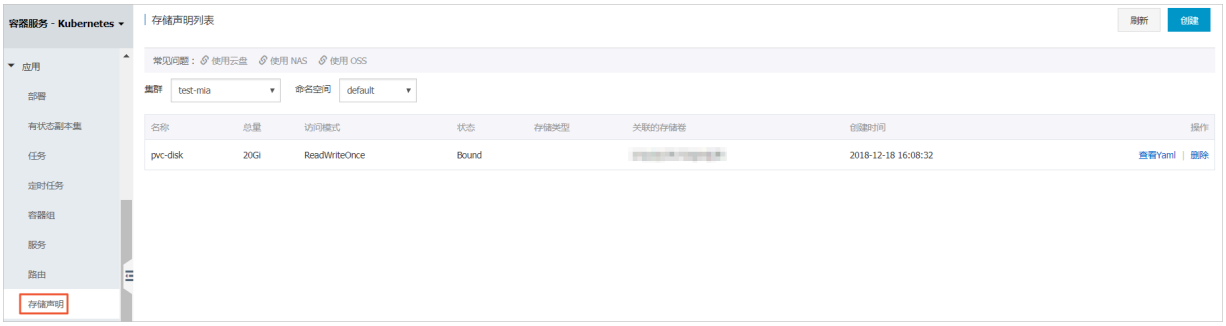
```
      storage: 20Gi
  selector:
    matchLabels:
      alicloud-pvname: <your-disk-id>
```

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#unique_11。
- 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。

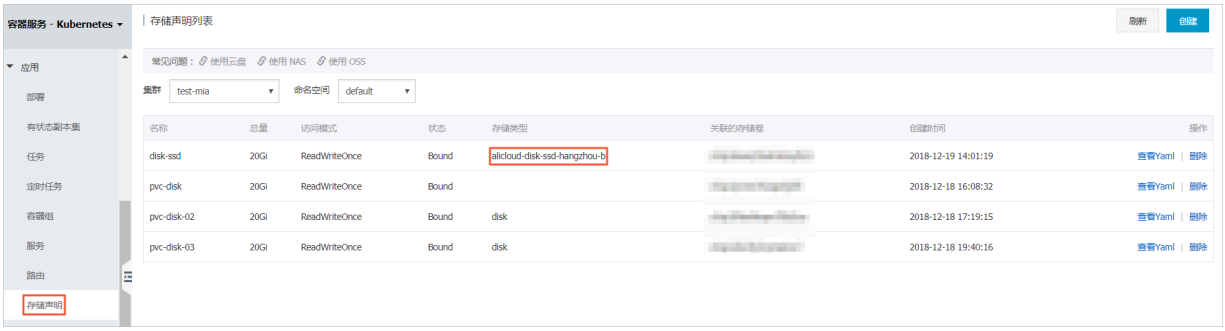
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: disk-ssd
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: alicloud-disk-ssd-hangzhou-b
  resources:
    requests:
      storage: 20Gi
```

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。

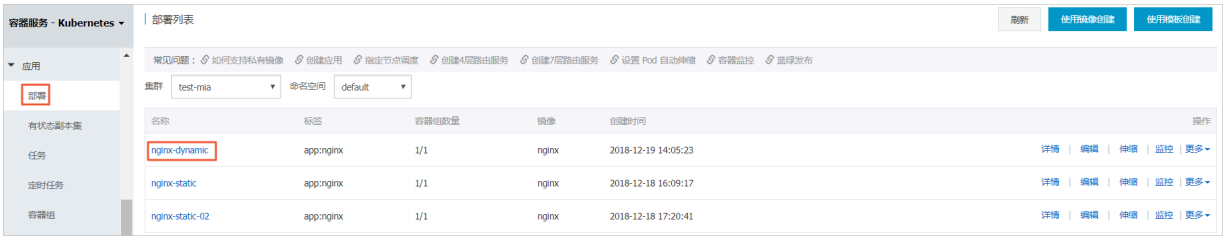
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-dynamic
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - 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#unique_11。
- 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#unique_11。
- 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Linux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#unique_29。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。

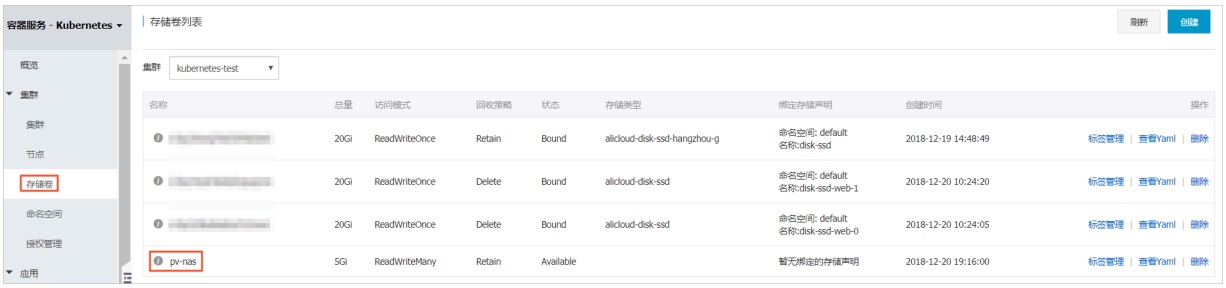
> **说明:**
>
> - 挂载 NAS的根目录不能配置访问权限。
> - 当 NAS中数据量很大时，配置`mode`会导致挂载非常慢，甚至挂载失败，不建议配置。

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。

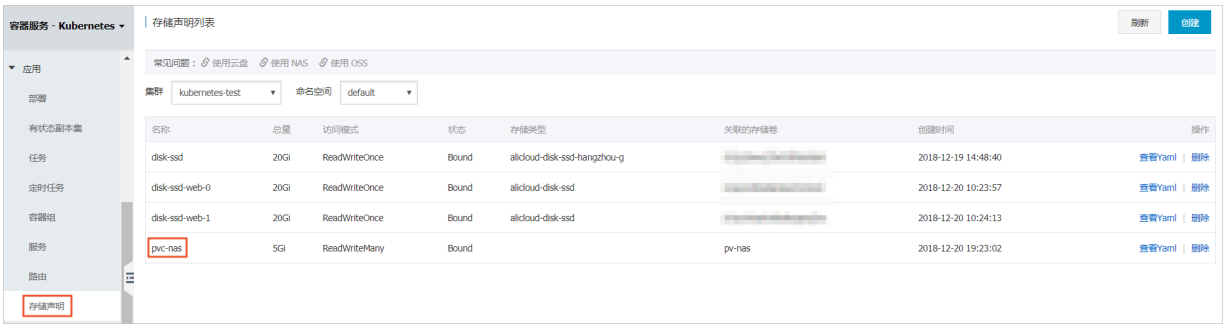
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-nas
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-nas
```

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:
```

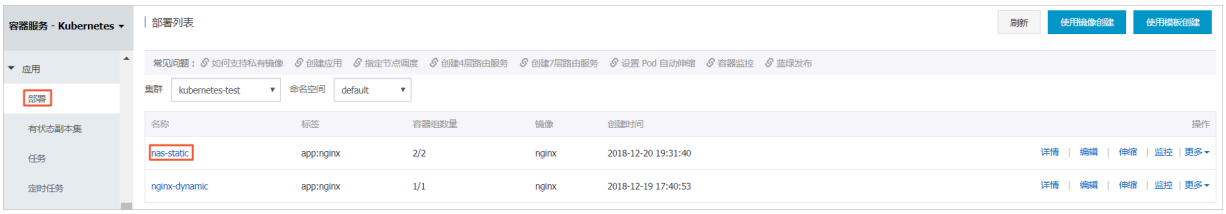
```
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-nas
            mountPath: "/data"
      volumes:
        - name: pvc-nas
          persistentVolumeClaim:
            claimName: 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#unique_11。
- 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，参见创建存储空间。

注意事项

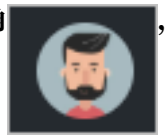
- 容器服务Kubernetes集群升级会重启kubelet，OSSFS驱动跟随一起重启，导致OSS目录不可用。这时使用OSS的Pod需要重建，可在yaml文件中增加健康检查的配置，在容器内OSS目录不可用时自动重启Pod，重新挂载OSS。
- 通过最新版本挂载的OSS已经解决了此问题。

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

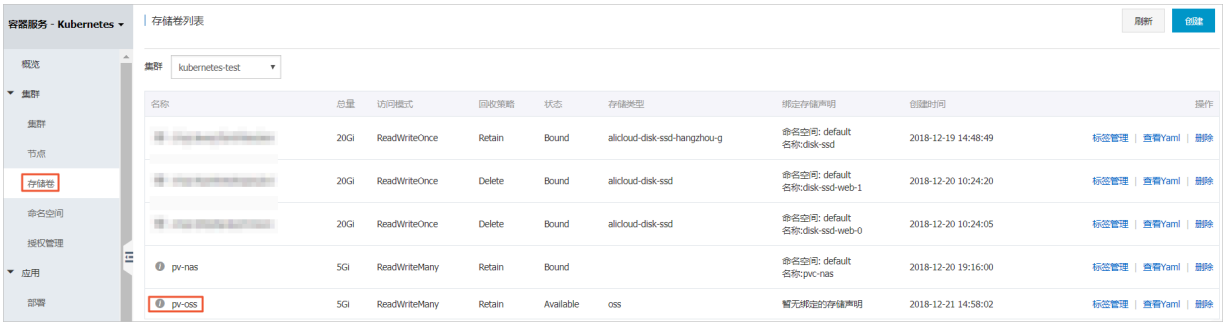
- `alicloud-pvname`：PV的名称，与PVC中的`selector`配合使用。
- `bucket`：Bucket名称，目前只支持挂载 Bucket，不支持挂载 Bucket 下面的子目录或文件。
- `url`：OSS Bucket的访问域名（Endpoint），可参见访问域名和数据中心。也可以在OSS管理控制台查询：登录OSS 管理控制台，在左侧导航栏选择目标Bucket，在访问域名区域，查看Endpoint（地域节点）。
- `akId`：用户的AccessKey ID。在容器服务管理控制台，单击右上角  ，主账号选择accesskeys，子账号选择AccessKey管理，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

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。

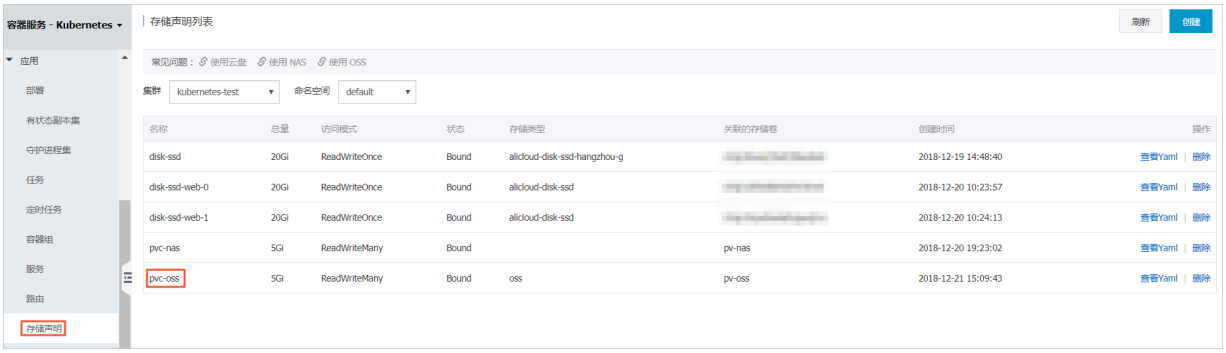
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-oss
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: oss
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-oss
```

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:

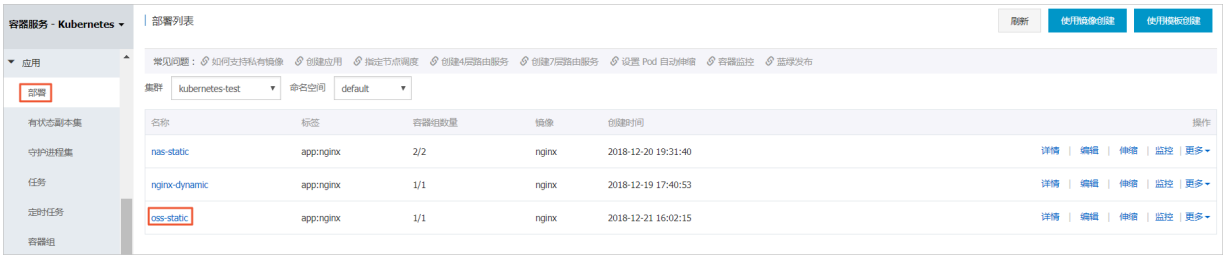
健康检查livenessProbe的详细内容及解释，请参见使用阿里云 OSS。

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。

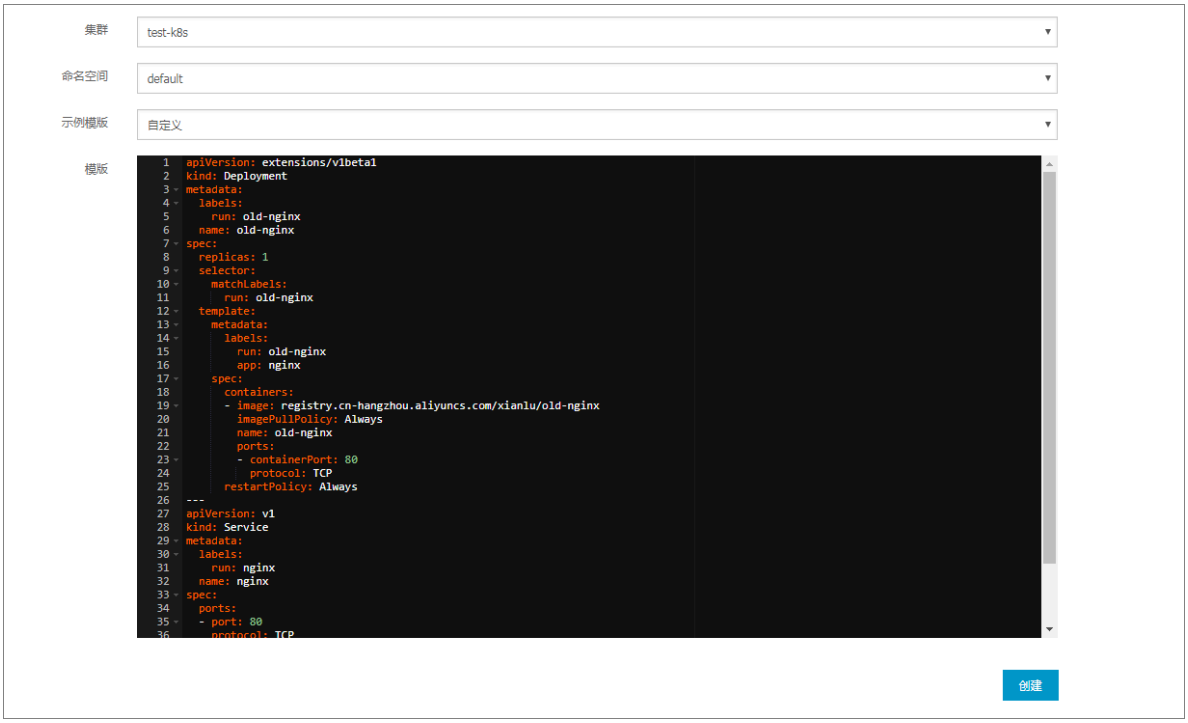
前提条件

- 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。

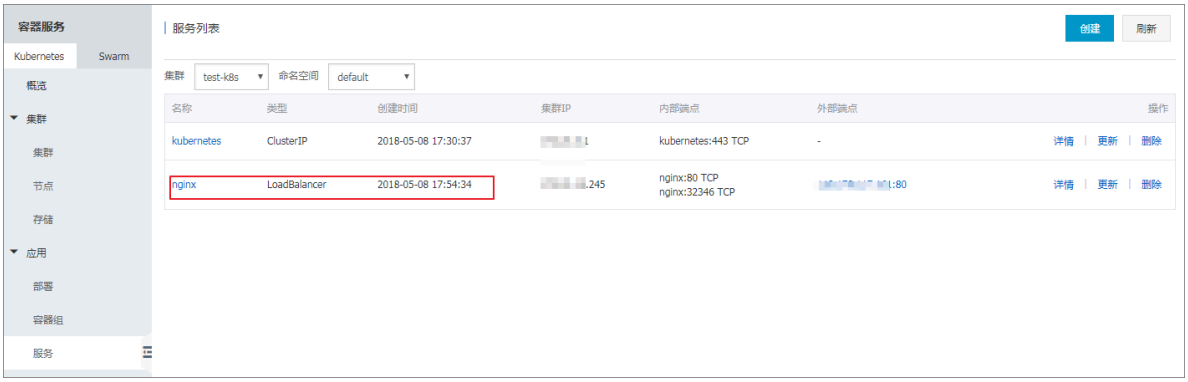


本例是一个 nginx 的编排，通过 SLB 对外暴露的服务。

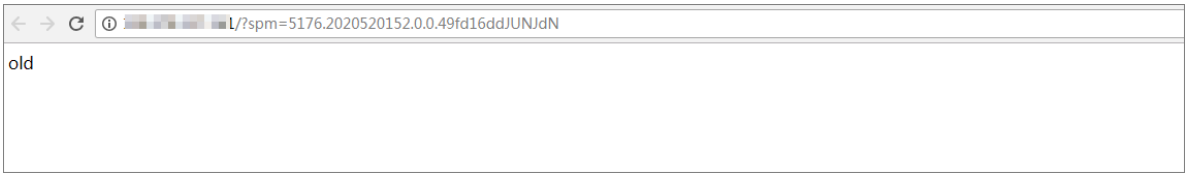
```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   labels:
     run: old-nginx
   name: old-nginx
 spec:
   replicas: 1
   selector:
     matchLabels:
       run: old-nginx
   template:
     metadata:
       labels:
         run: old-nginx
         app: nginx
     spec:
       containers:
       - 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。

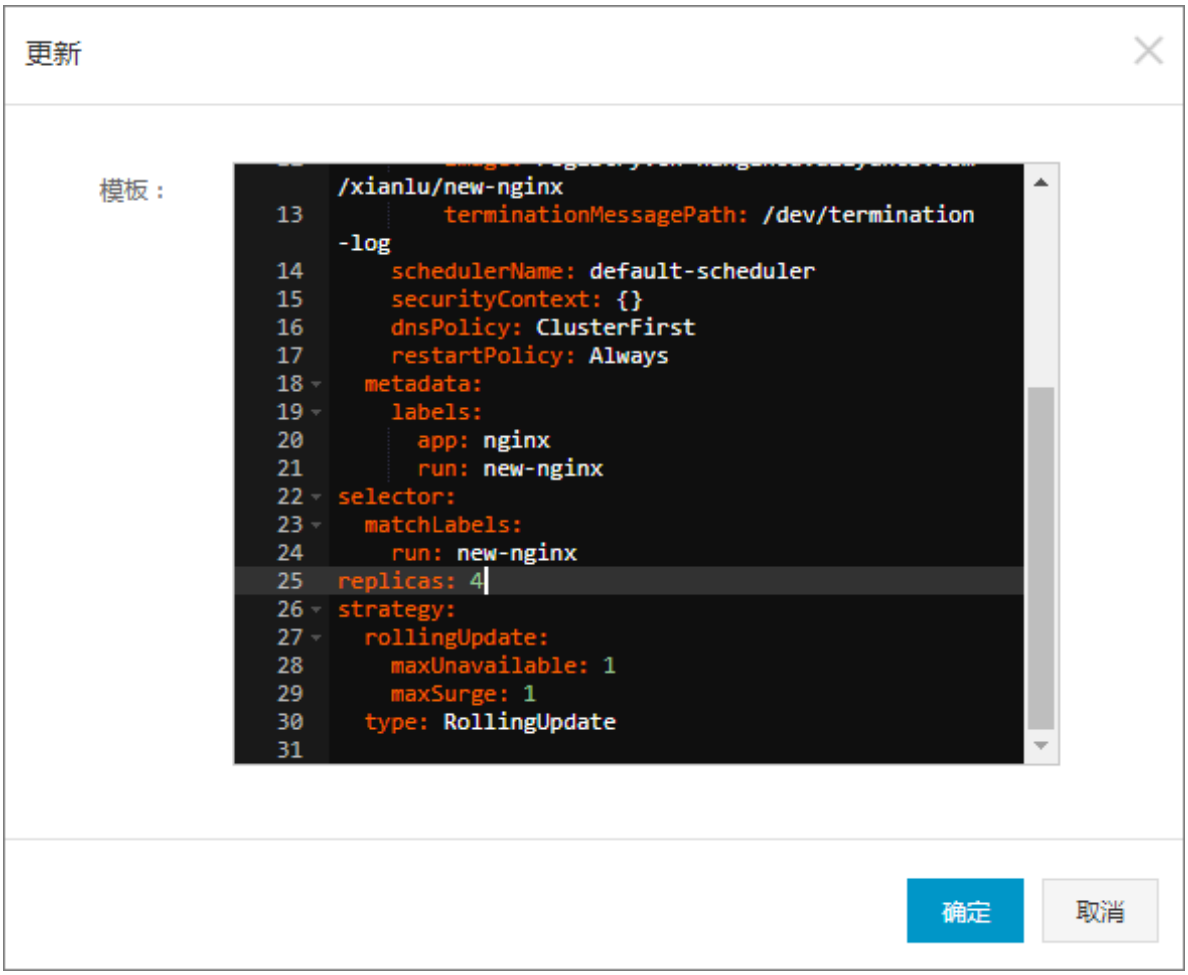
> **说明:**
> 新旧版本应用同时存在时，某个样本的 curl 命令返回的结果并非严格按照设置的权重，本例中执行 10 次 curl 命令，您可通过观察更大的样本获取接近的效果。

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 Kubernetes集群中通过 Ingress 实现灰度发布和蓝绿发布

4.2.1 概述

本文介绍如何利用阿里云容器服务的Ingress功能，实现灰度发布和蓝绿发布。

背景信息

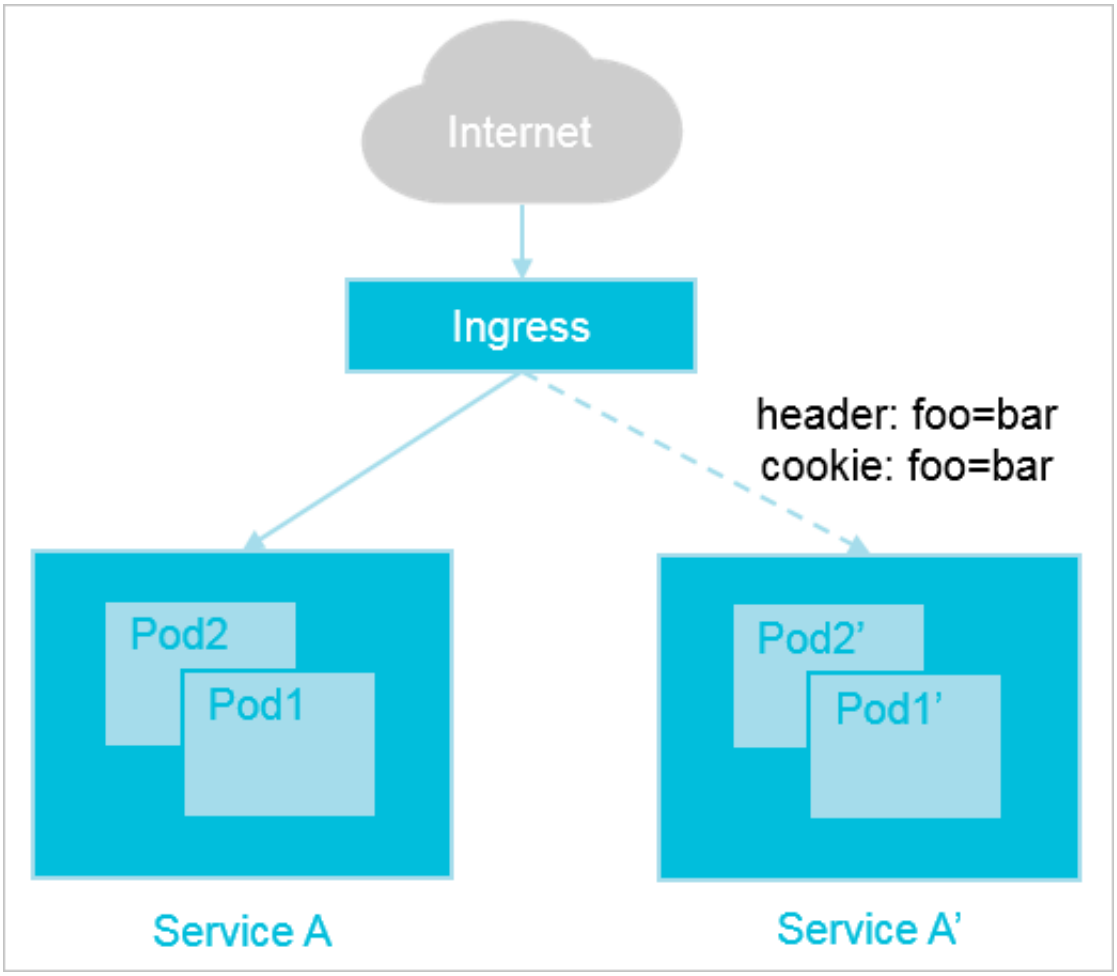
灰度及蓝绿发布是为新版本创建一个与老版本完全一致的生产环境，在不影响老版本的前提下，按照一定的规则把部分流量切换到新版本，当新版本试运行一段时间没有问题后，将用户的全量流量从老版本迁移至新版本。

其中A/B测试就是一种灰度发布方式，一部分用户继续使用老版本的服务，将一部分用户的流量切换到新版本，如果新版本运行稳定，则逐步将所有用户迁移到新版本。

应用场景

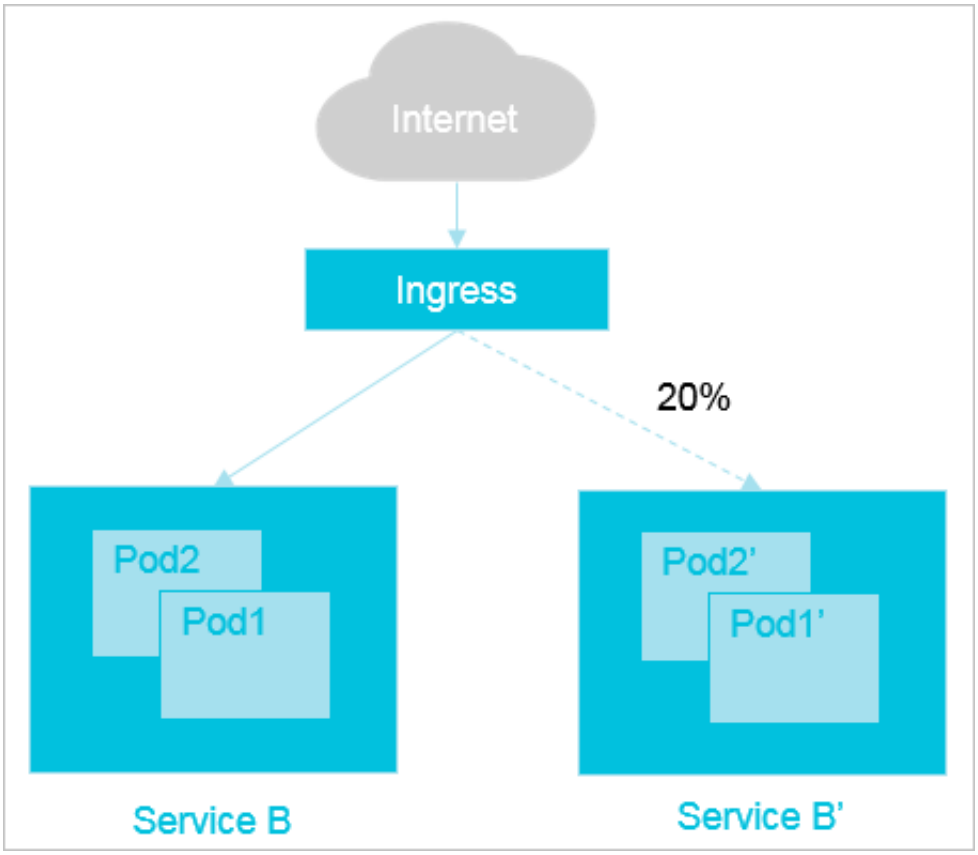
场景一

假设当前线上环境，您已经有一套服务Service A对外提供7层服务，此时上线了一些新的特性，需要发布上线一个新的版本Service A'，但又不想简单地直接替换掉Service A服务，而是希望将请求头中包含`foo=bar`或者cookie中包含`foo=bar`的客户端请求转发到Service A'服务中，待运行一段时间稳定后，可将所有的流量从Service A切换到Service A'服务中，再平滑地将Service A服务下线。



场景二

假设当前线上环境，您已经有一套服务Service B对外提供7层服务，此时修复了一些问题，需要发布上线一个新的版本Service B'，但又不想简单地将所有客户端流量切换到新版本Service B'中，而是希望将20%的流量切换到新版本Service B'中，待运行一段时间稳定后，可将所有的流量从Service B切换到Service B'服务中，再平滑地将Service B服务下线。



针对以上多种不同的应用发布需求，阿里云容器服务Kubernetes的 Ingress 功能提供的4种流量切分方式：

灰度发布中的A/B 测试：

- 基于Request Header的流量切分
- 基于Cookie的流量切分
- 基于Query Param的流量切分

蓝绿发布：

- 基于服务权重的流量切分

4.2.2 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
```

4.2.3 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。

4.2.4 步骤1：部署服务

介绍如何部署服务。

前提条件

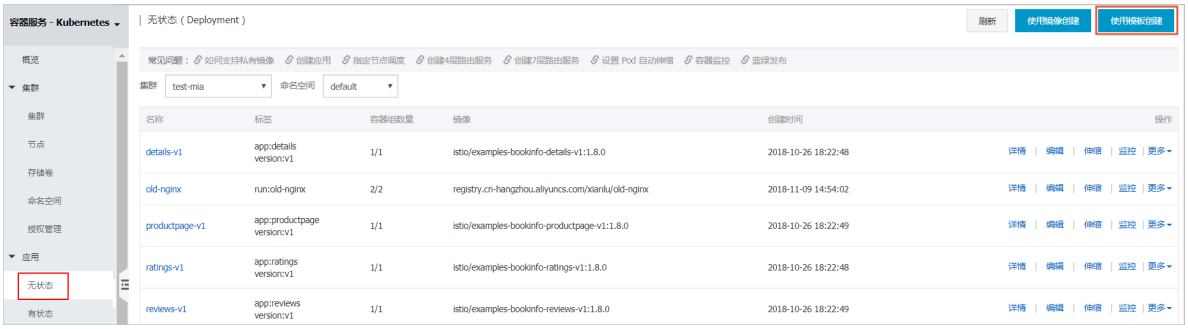
- 您已成功部署一个Kubernetes集群，参见#unique_11。
- 您可以通过Kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。

背景信息

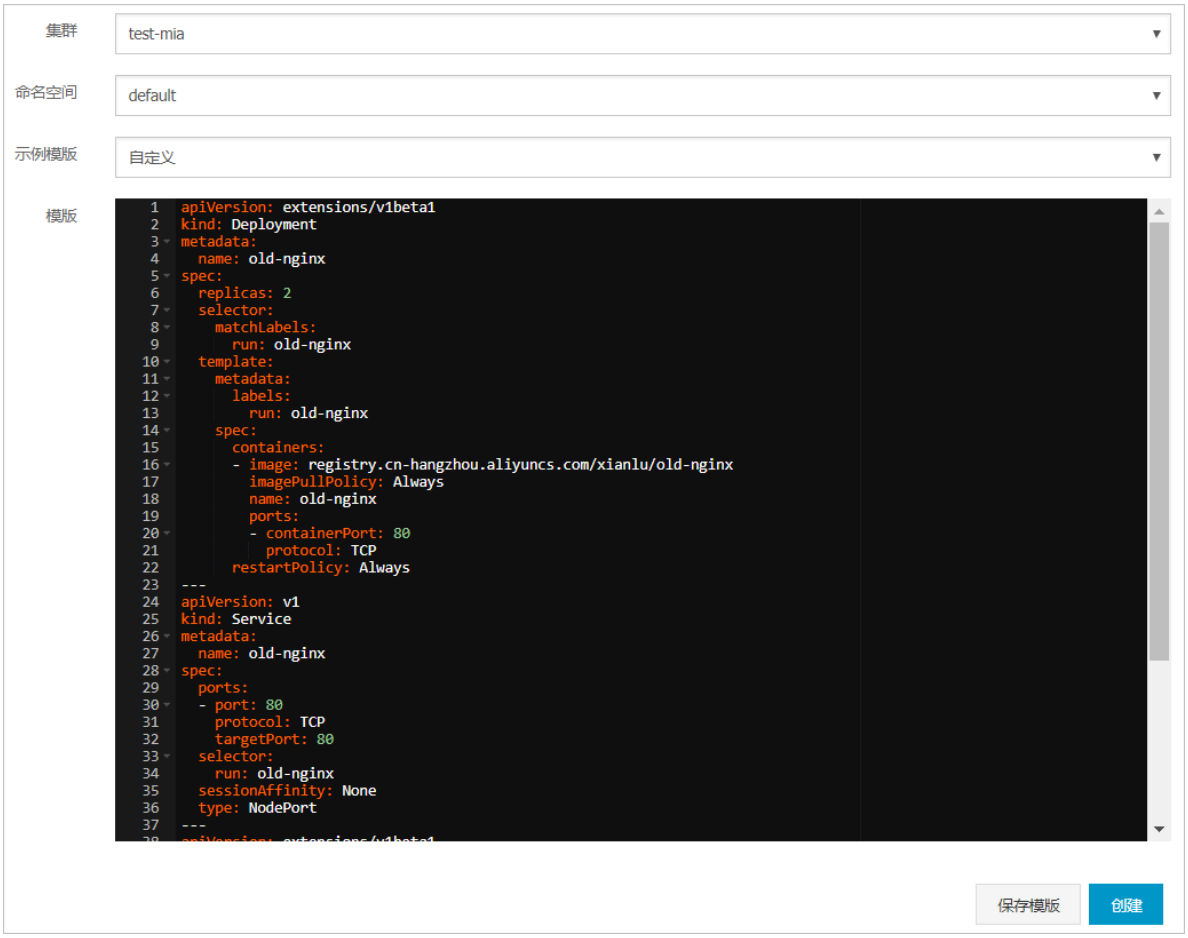
灰度及蓝绿发布是为新版本创建一个与老版本完全一致的生产环境，在不影响老版本的前提下，按照一定的规则把部分流量切换到新版本，当新版本试运行一段时间没有问题后，将用户的全量流量从老版本迁移至新版本。因此，这里先在集群中部署一个旧版本的服务。

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 > 无状态，进入无状态（Deployment）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。



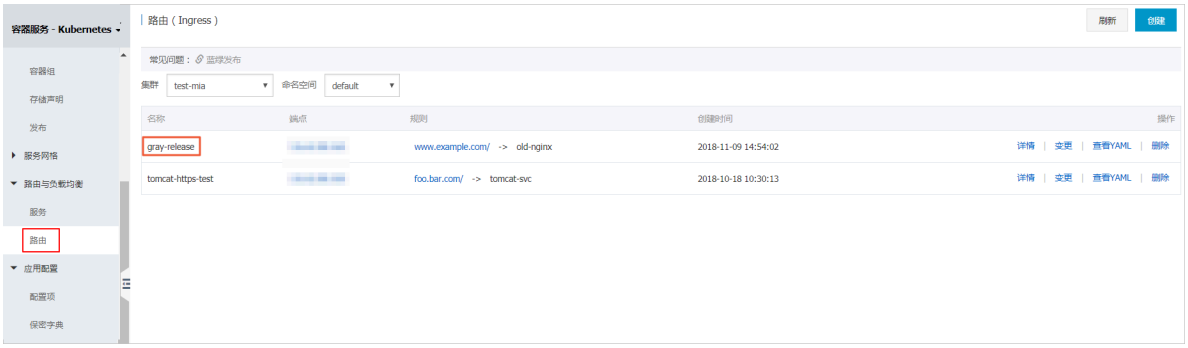
本例是一个 nginx 应用，包含一个 deployment、 service 以及 ingress。deployment 通过 NodePort 对外暴露端口，并且有一个 ingress 正在对外提供服务。编排模板如下。

```
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路由与负载均衡 > 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>
```

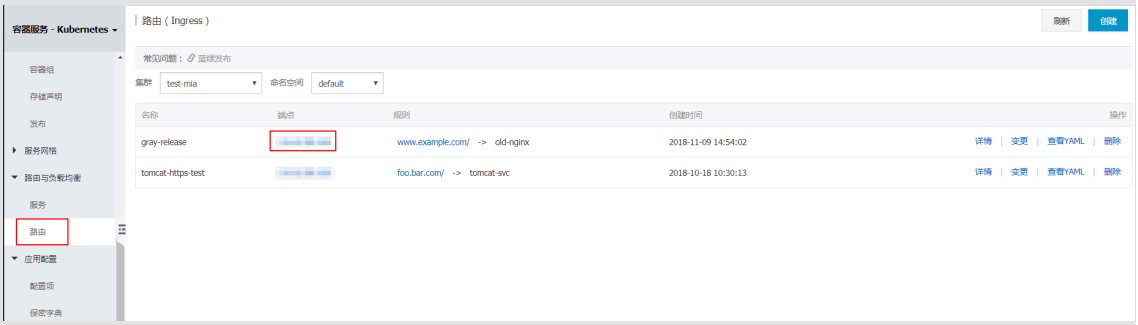
说明:

`<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。



4.2.5 步骤2：发布新版本服务

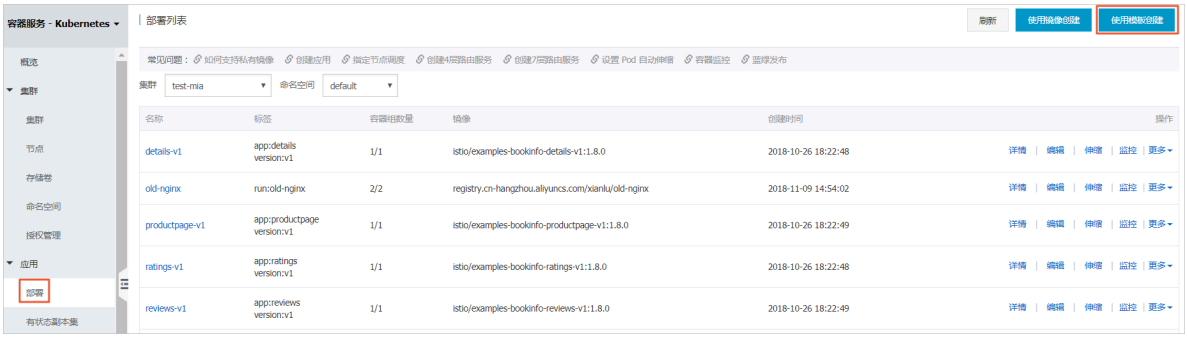
本文介绍如何利用阿里云容器服务的Ingress功能，发布新版本服务。

前提条件

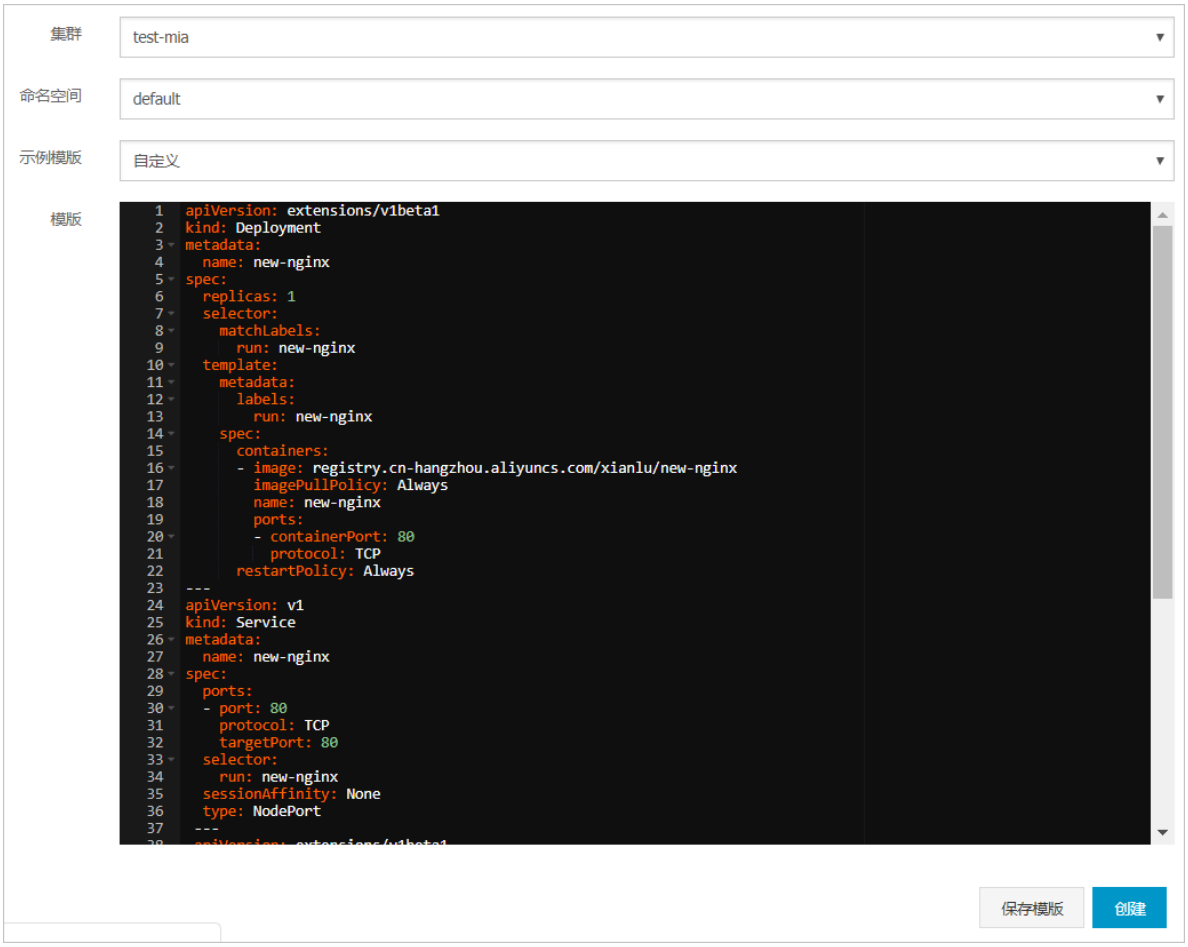
- 您已成功部署一个Kubernetes集群，参见#unique_11。
- 您可以通过Kubectl连接到Kubernetes集群，参见通过 kubectl 连接 Kubernetes 集群。

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
```

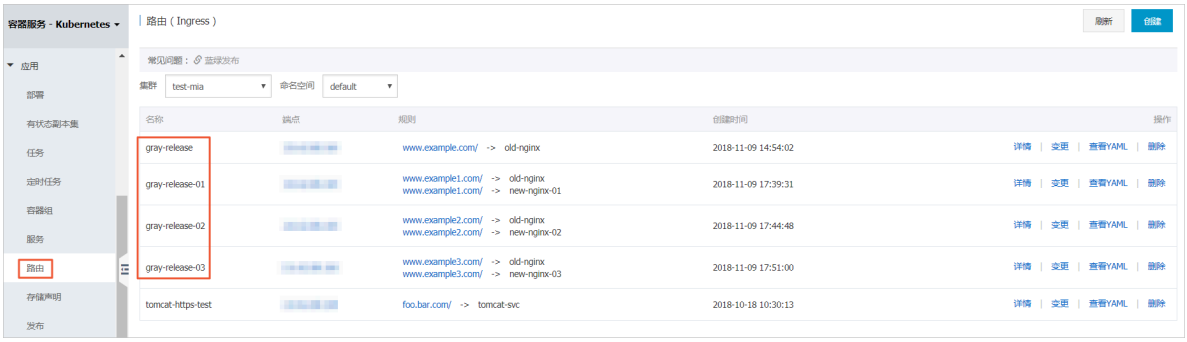
- 满足foo=bar的客户端请求仅允许50%的流量被路由到新版本服务

```
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_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://[redacted]
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://[redacted]
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://[redacted]
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example1.com" -H "foo: bar" http://[redacted]
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://[redacted]
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://[redacted]
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://[redacted]
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://[redacted]
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_IP>
```

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

4.2.6 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#unique_11。
- 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

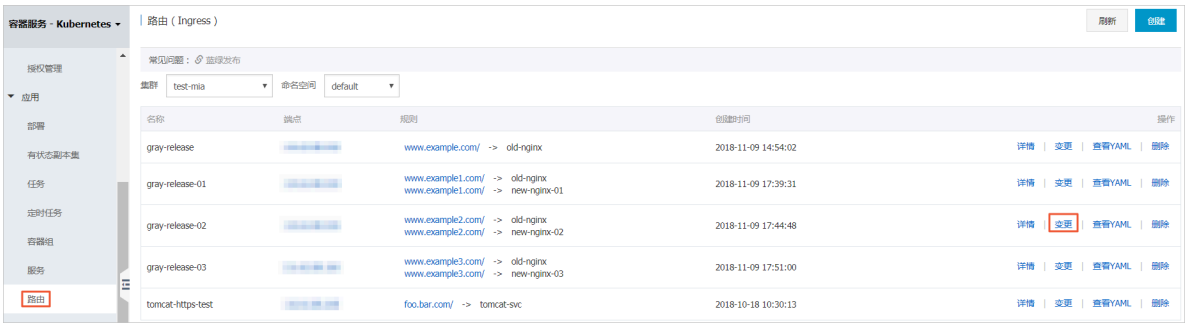
1. 执行以下命令，编辑步骤2：发布新版本服务已经部署的yaml文件，删除老版本的服务。

> **说明:**
>
> 请将annotations一并删除。

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

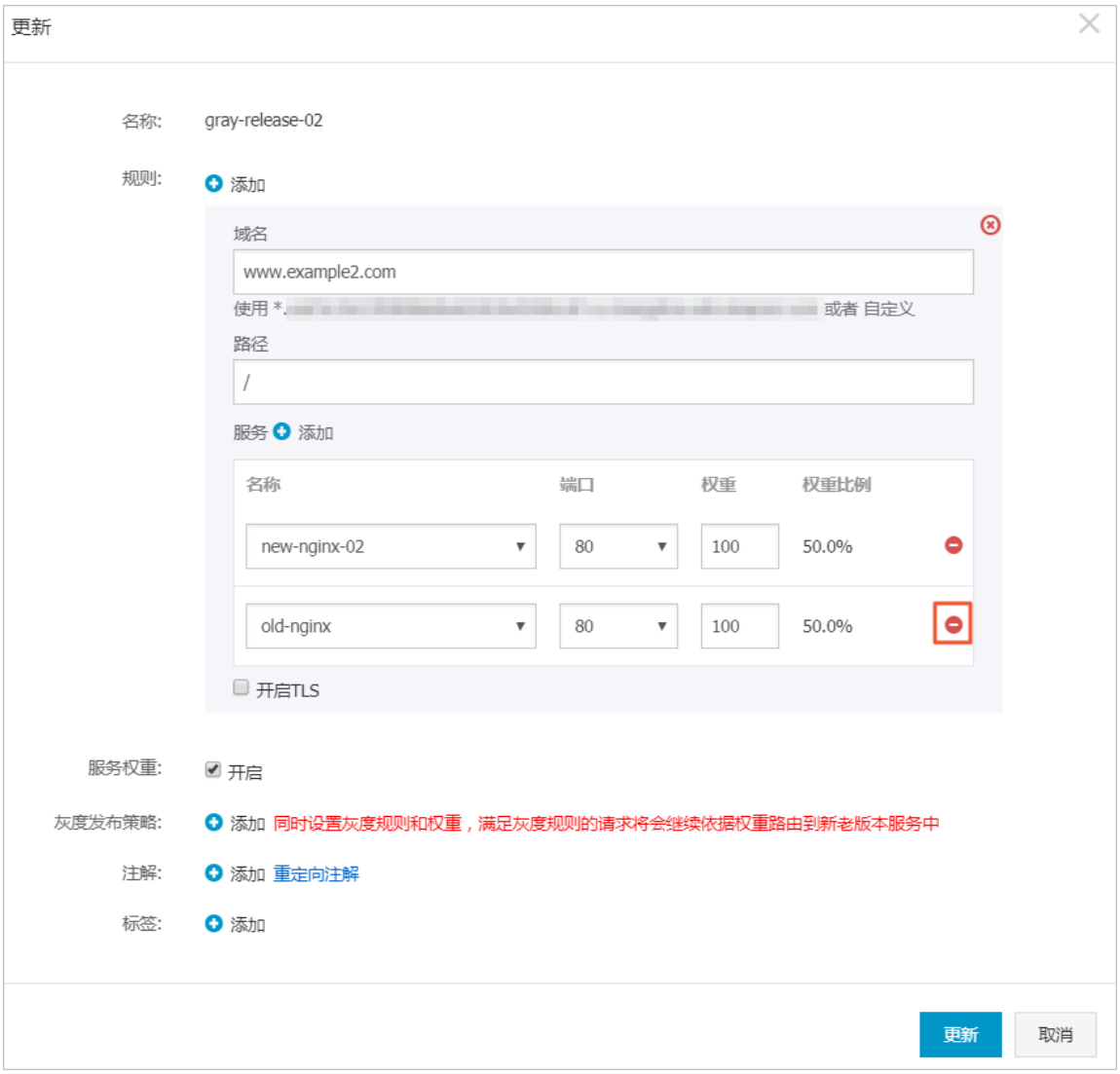
通过控制台删除

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：

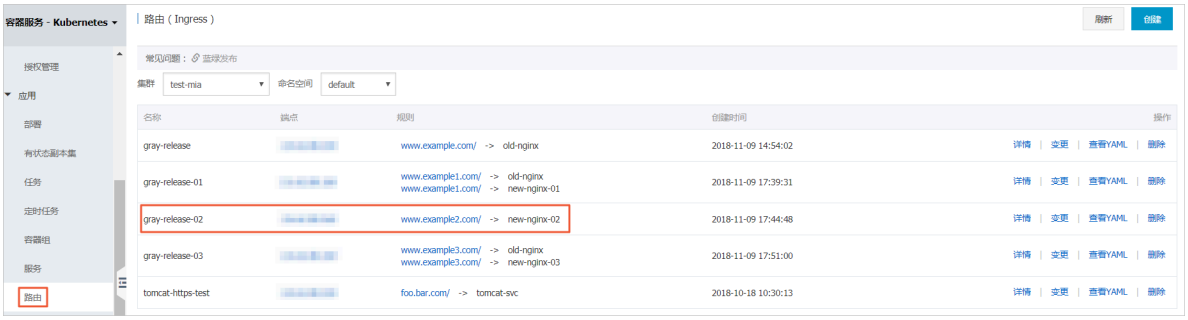
   a. 在规则 > 服务区域，删除老版本服务规则。



   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。

4.3 应用

4.3.1 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

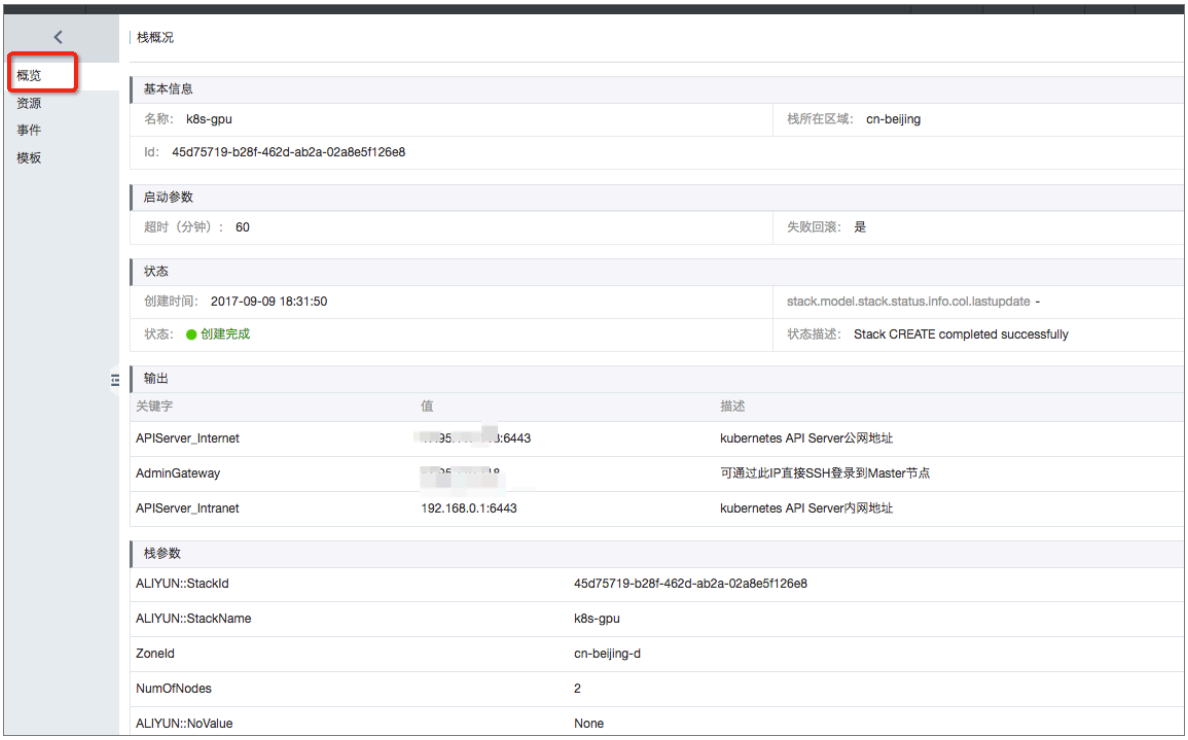
2. 填写参数并单击创建。

- 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
```

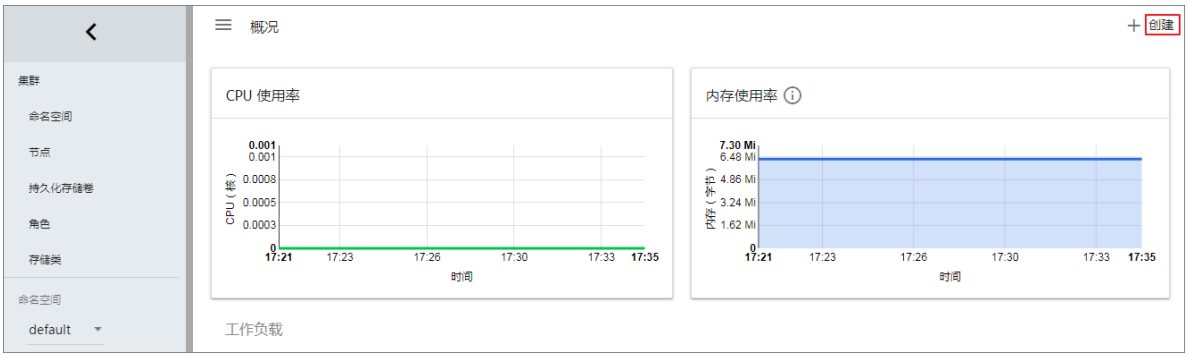
Deployment配置：

- `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。

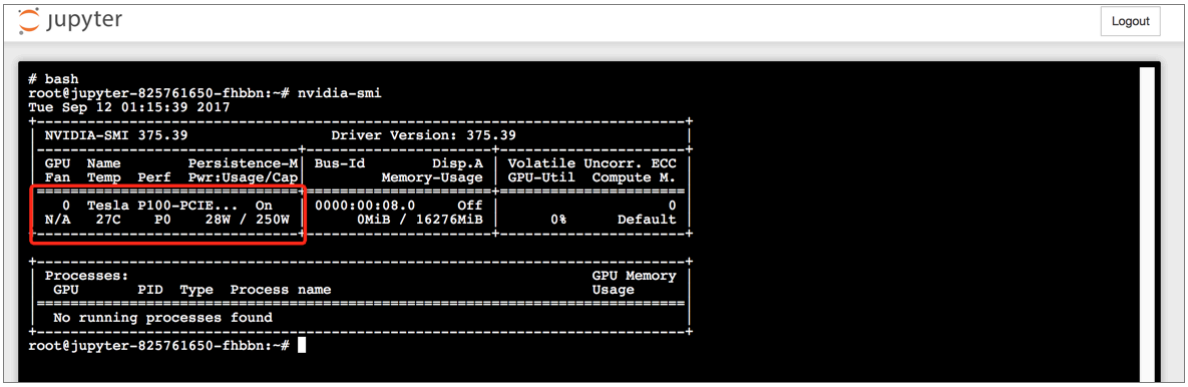


2. 单击使用文件创建。选择刚才创建的 jupyter.yml 文件

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。



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#unique_11。
- 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
```

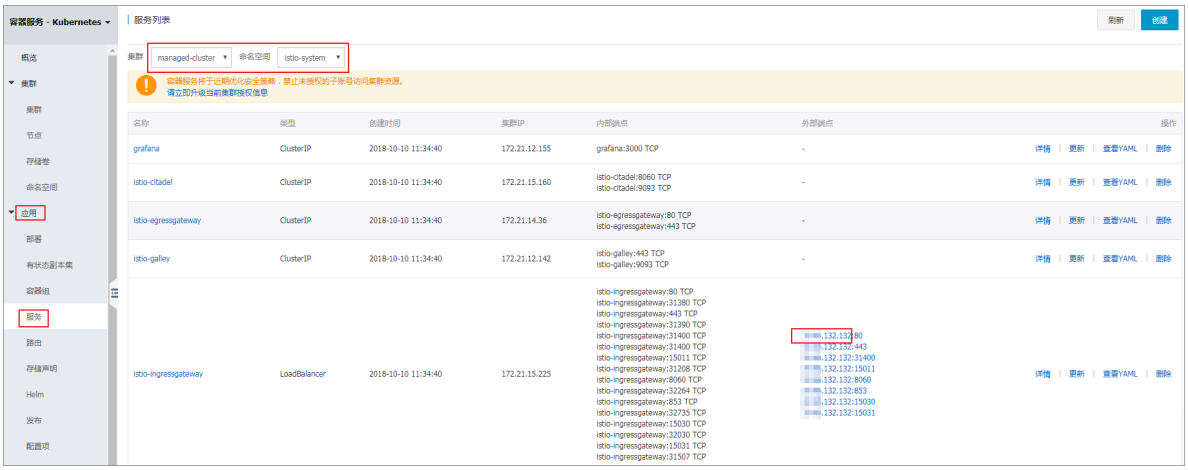
确认网关创建完成：

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

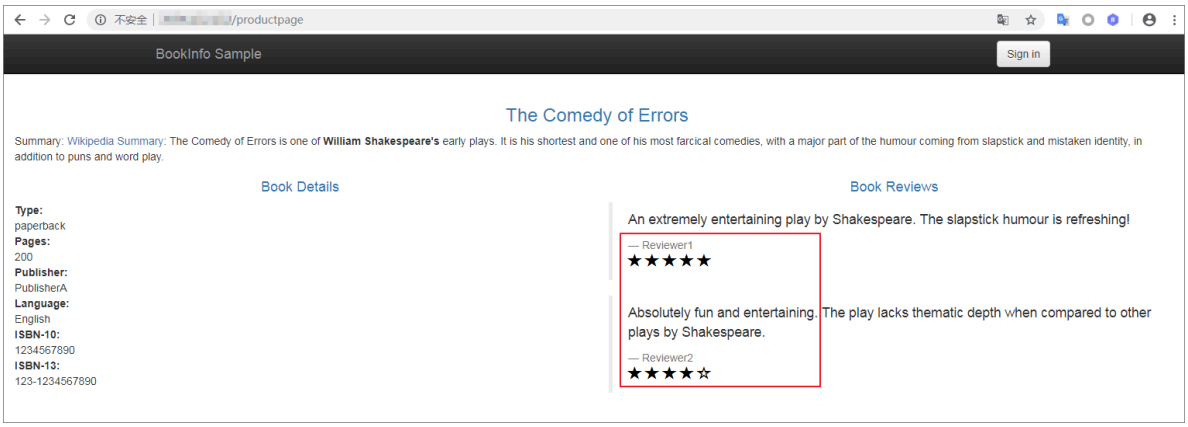
5. 然后确认`istio-ingressgateway`的访问地址。您可通过命令行快速查询。

```
$ 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
```

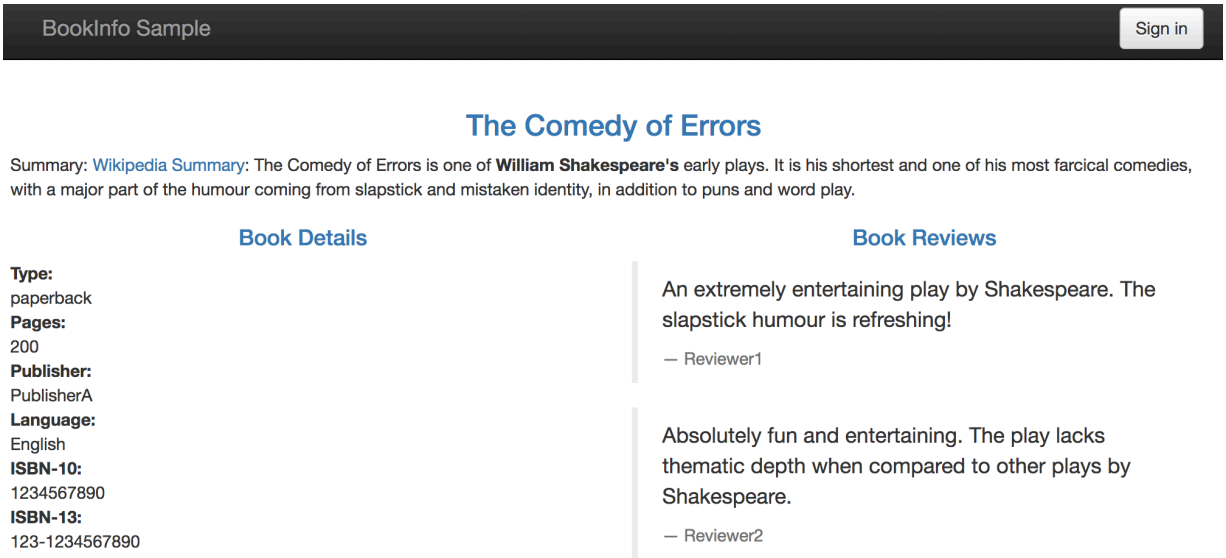
可以通过下面的命令来显示所有已创建的路由规则：

```
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：

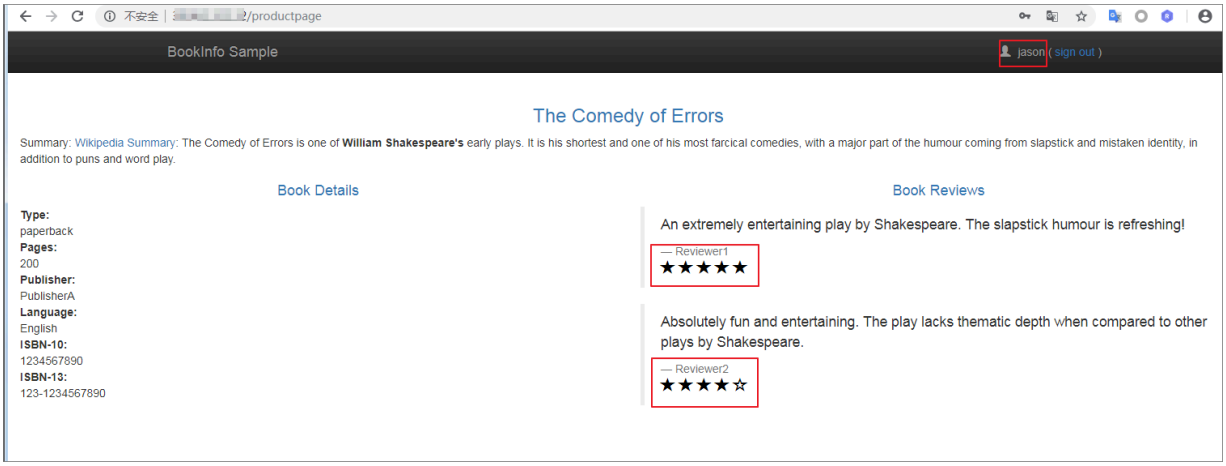
```
$ kubectl get virtualservice reviews -o yaml
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
  ...
spec:
  hosts:
  - reviews
  http:
  - match:
    - headers:
        end-user:
          exact: jason
    route:
    - destination:
        host: reviews
   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

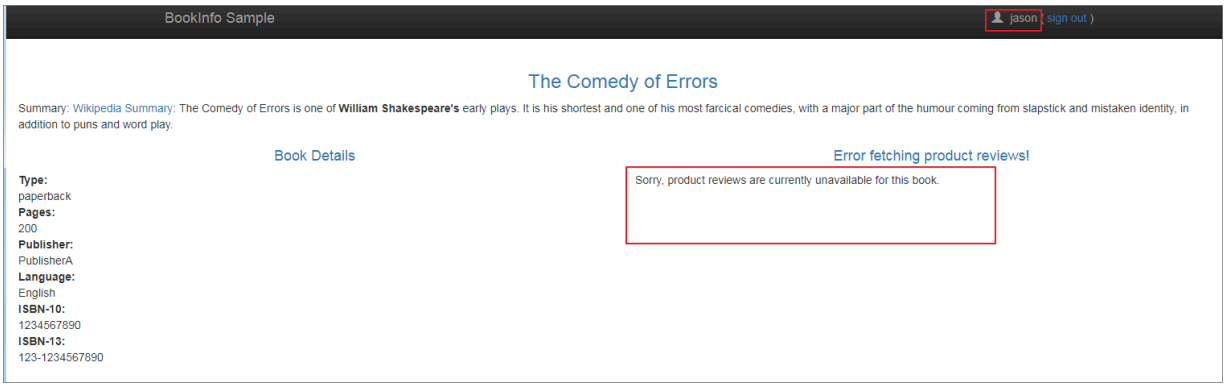
HTTP Delay

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。

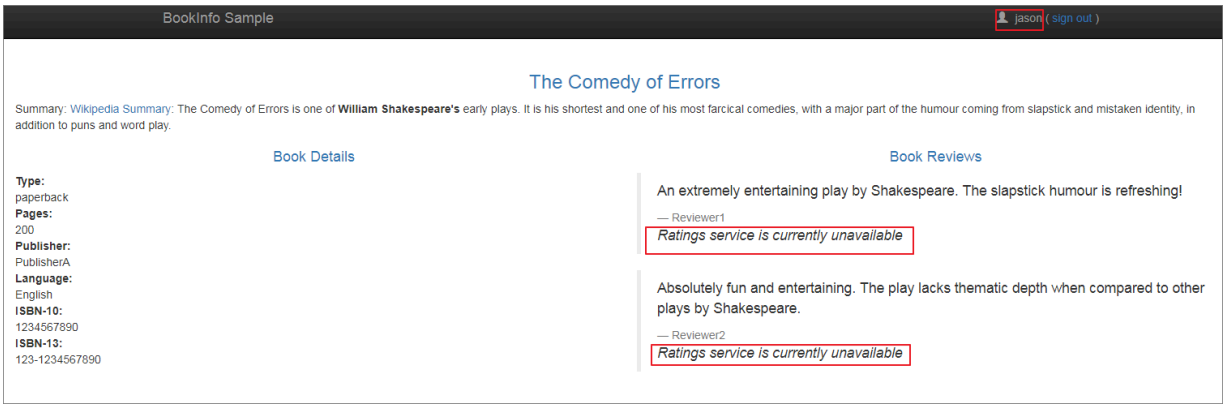
HTTP Abort

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#unique_11。
- 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1.1，更多关于部署Istio的信息，请参见部署Istio。

3. 在部署 Istio 页面中，选择启用分布式追踪 Jaeger。阿里云容器服务支持通过Web界面一键部署Jaeger追踪系统。
4. 部署完成后，在左侧菜单栏中单击路由与负载均衡 > 服务，选择所需集群和命名空间，找到tracing服务。
5. 如果tracing服务指定为LoadBalancer方式，单击tracing服务的外部地址，即可进入Jaeger UI页面。如果tracing服务指定为ClusterIP方式，可通过port-forward或路由方式访问。

   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
```

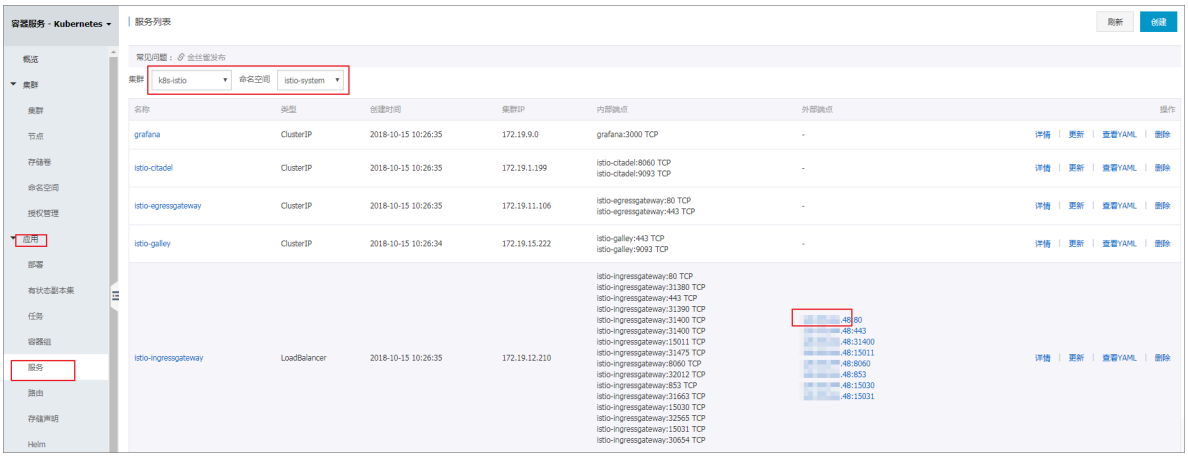
确认网关创建完成：

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

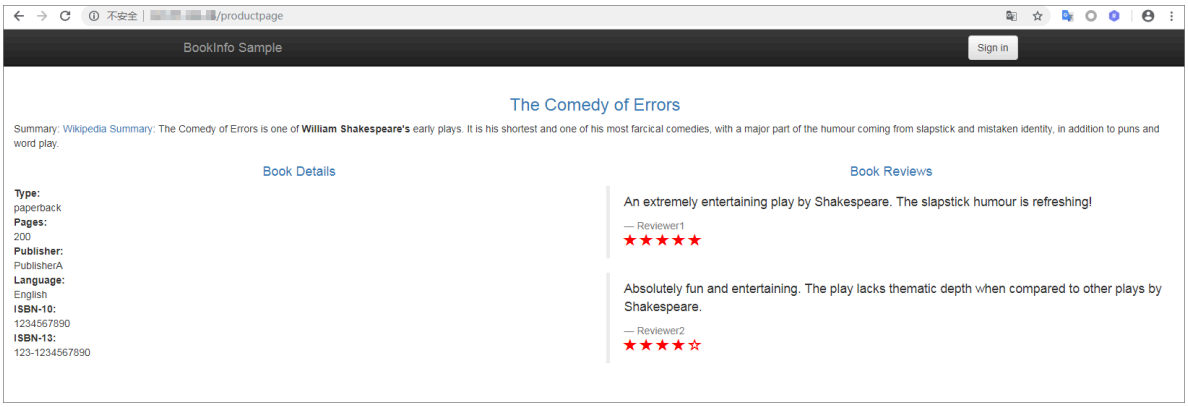
5. 然后确认`istio-ingressgateway`的访问地址。您可通过命令行快速查询。

```
$ 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`。



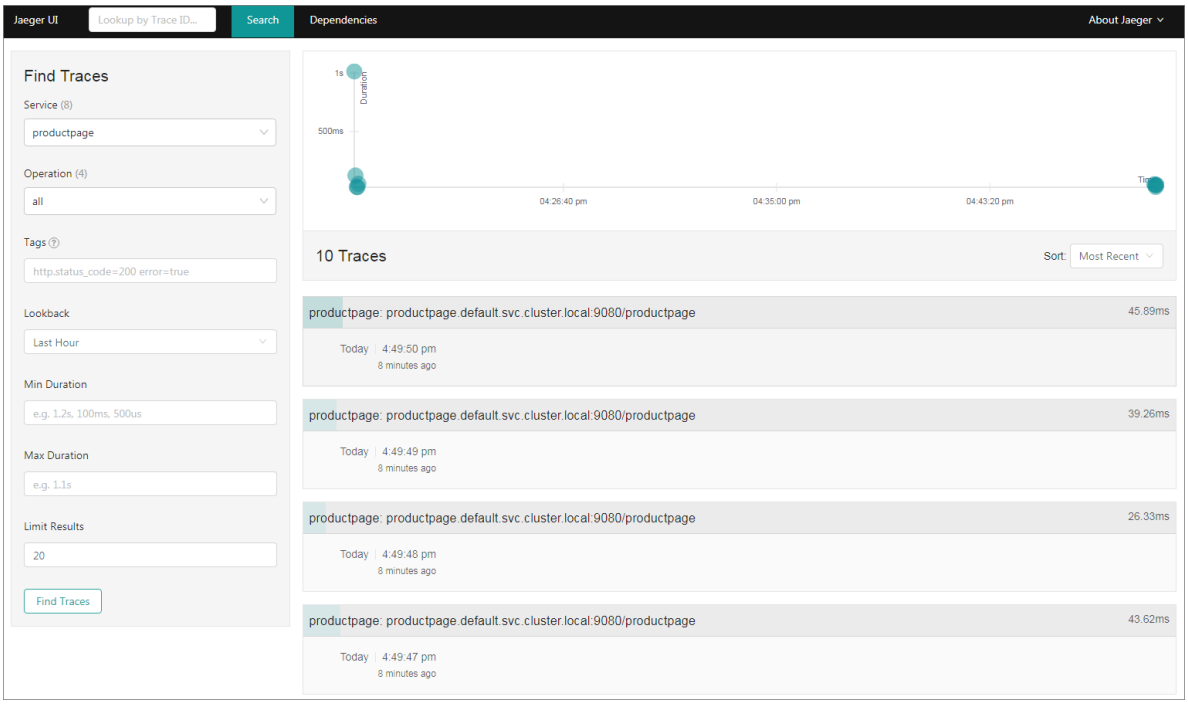
多次刷新浏览器，将在 productpage 中看到评论的不同的版本，它们会按照 round robin（红星、黑星、没有星星）的方式展现，因为目前为止还没有使用 Istio 来控制版本的路由。

步骤3 查看Jaeger追踪结果

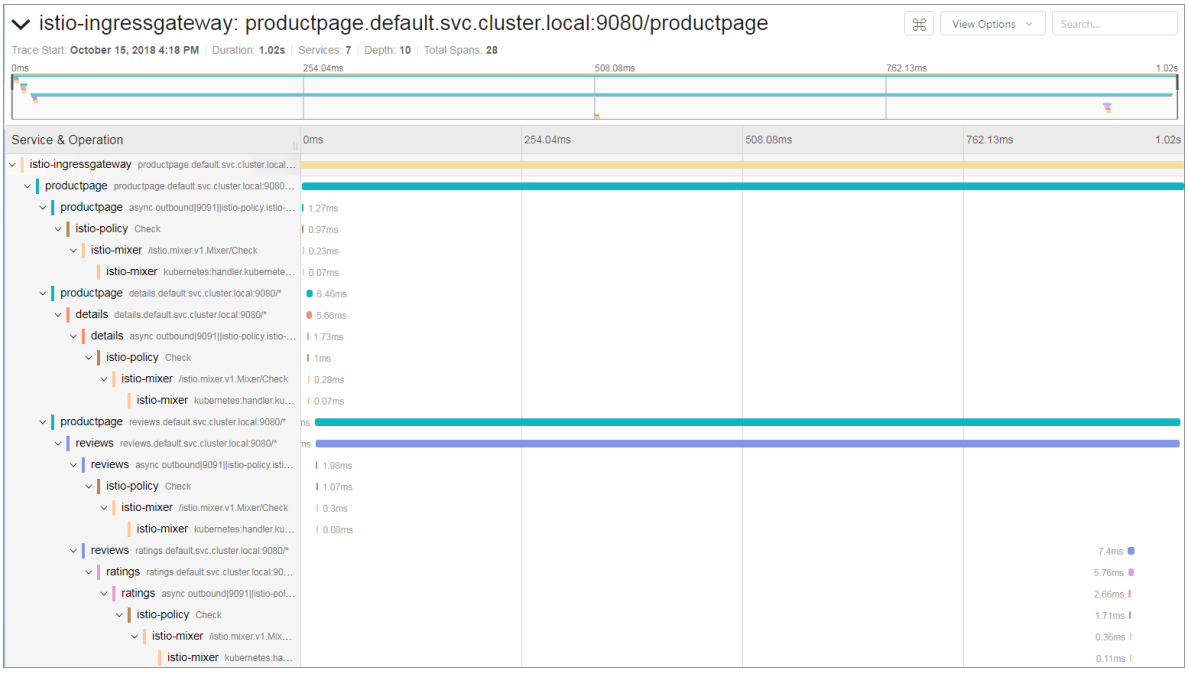
Jaeger UI显示了分布式服务追踪信息的结果。

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#unique_11。
- 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。

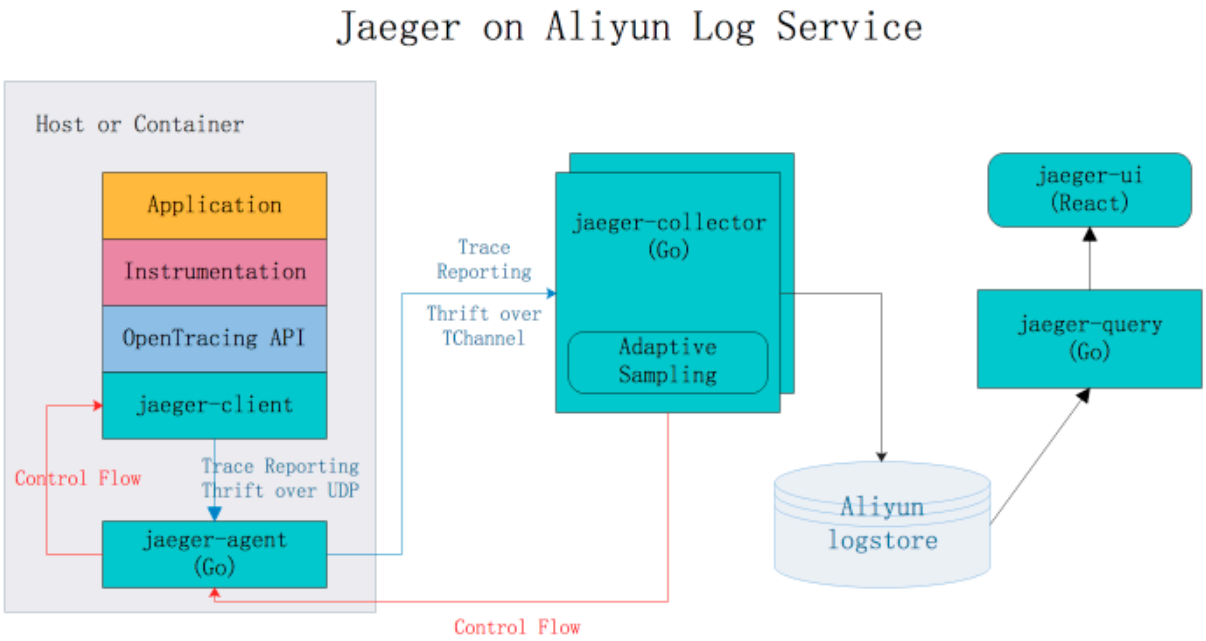
Jaeger 是CNCF下的一款开源分布式追踪系统，兼容 OpenTracing API。

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。

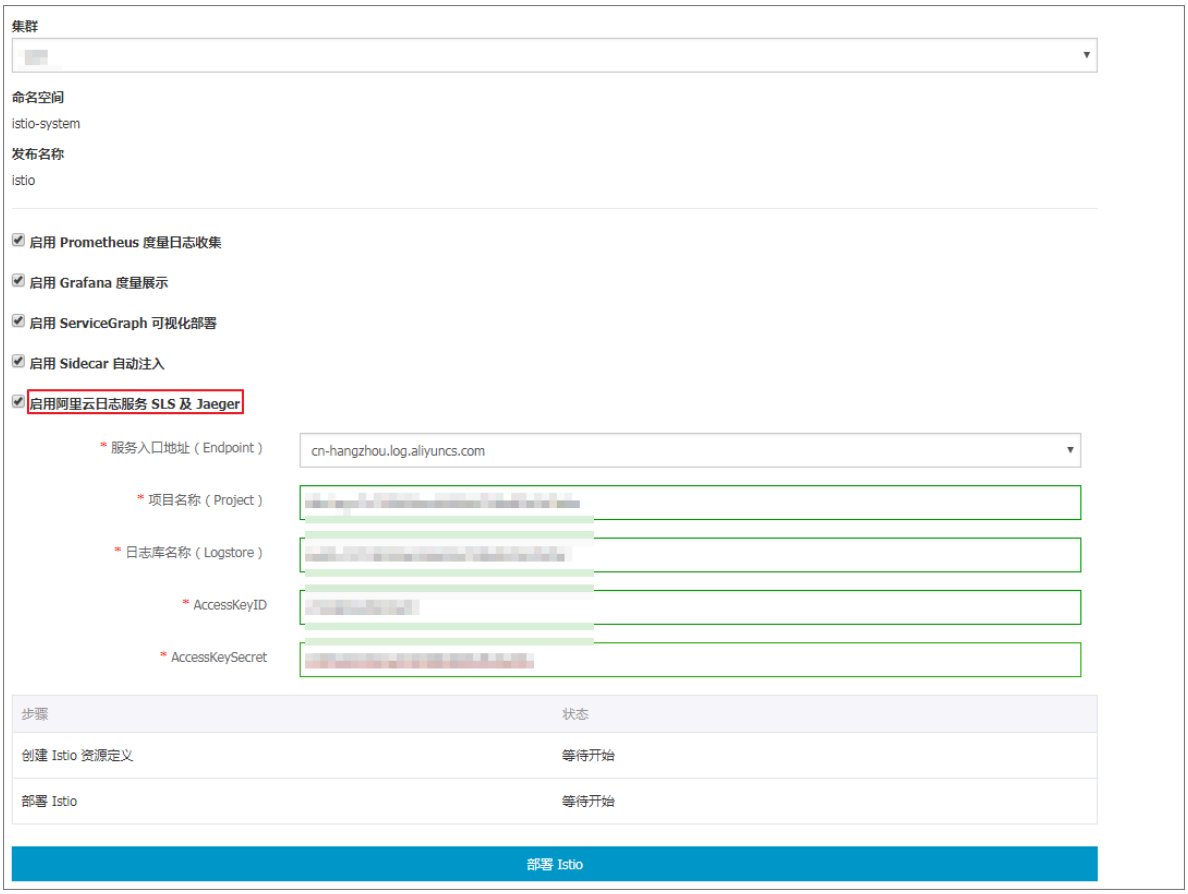
步骤1 通过Web界面一键部署Istio

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

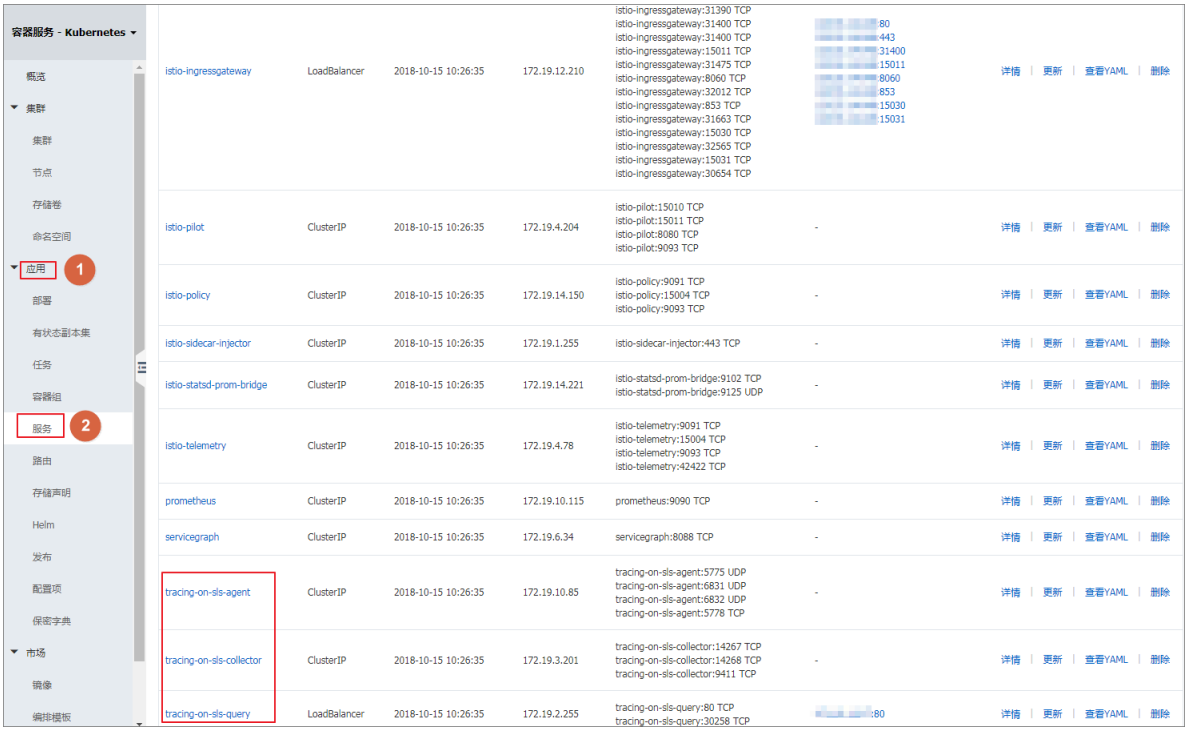
参数名	对应参数	描述
日志库名称（Logstore）	`storage.aliyun_sls.logstore`	指定用于存储 Span 的 Logstore，须由小写字母、数字、连字符（-）和下划线（_）组成，且以小写字母或者数字开头和结尾，长度为3-63字节。Logstore 名称在其所属项目内必须唯一
AccessKeyID	`storage.aliyun_sls.accesskey.id`	指定用户标识 Access Key ID
AccessKeySecret	`storage.aliyun_sls.accesskey.secret`	指定用户标识 Access Key Secret

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)。

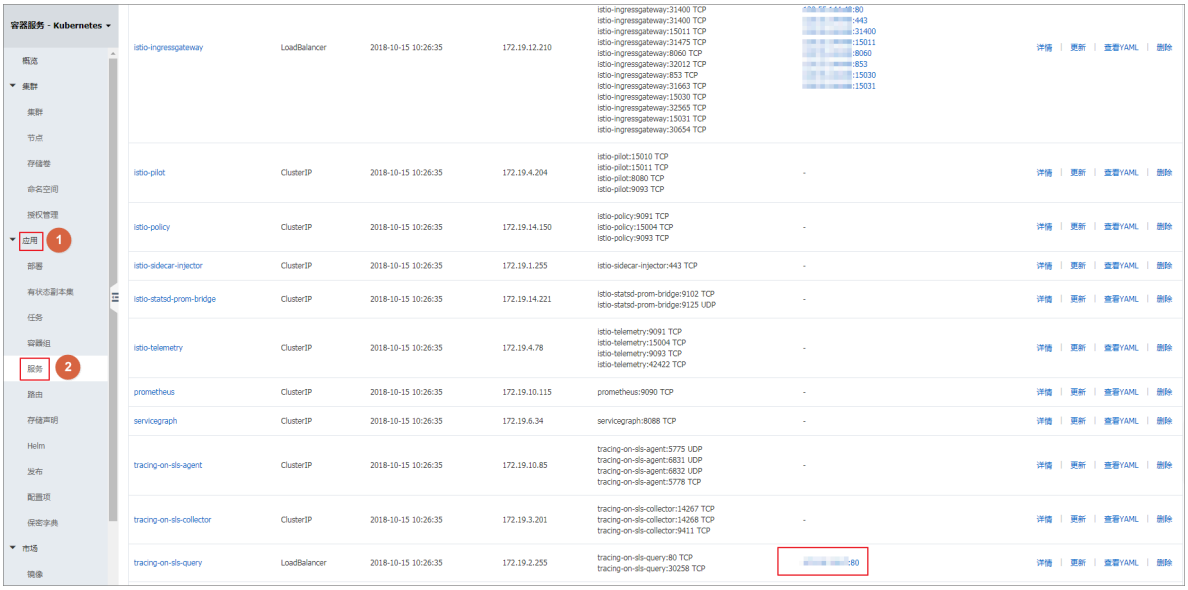
4. 最后单击部署 Istio。几分钟之后，一套集成阿里云日志服务的Istio实例就已创建完毕。

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。



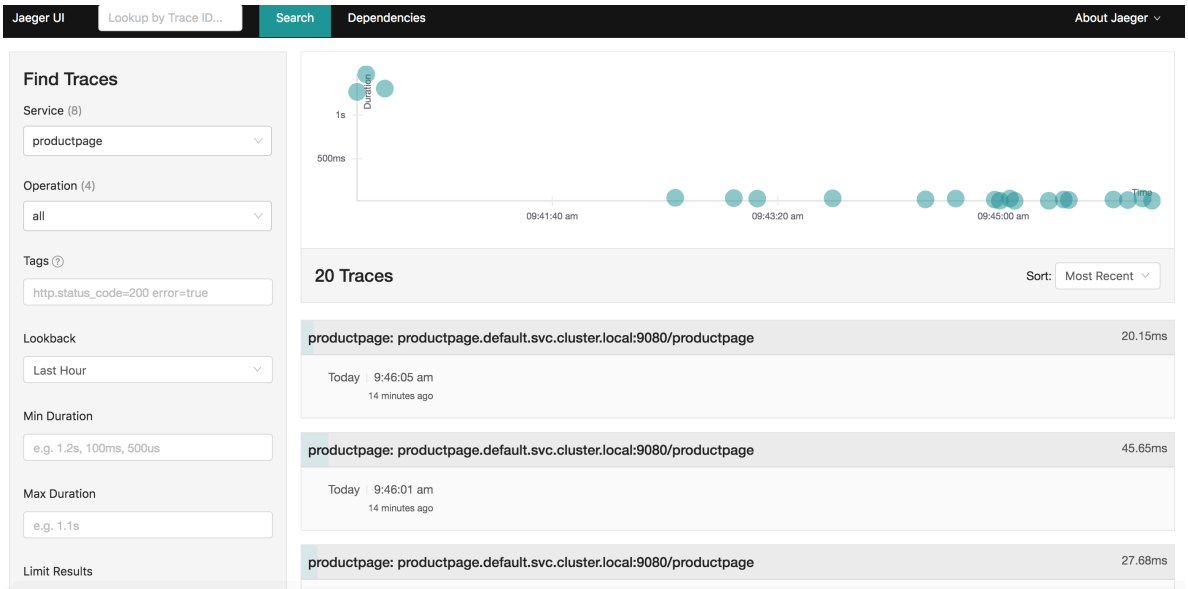
步骤2 查看分布式调用链跟踪Jaeger on Aliyun Log Service

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。

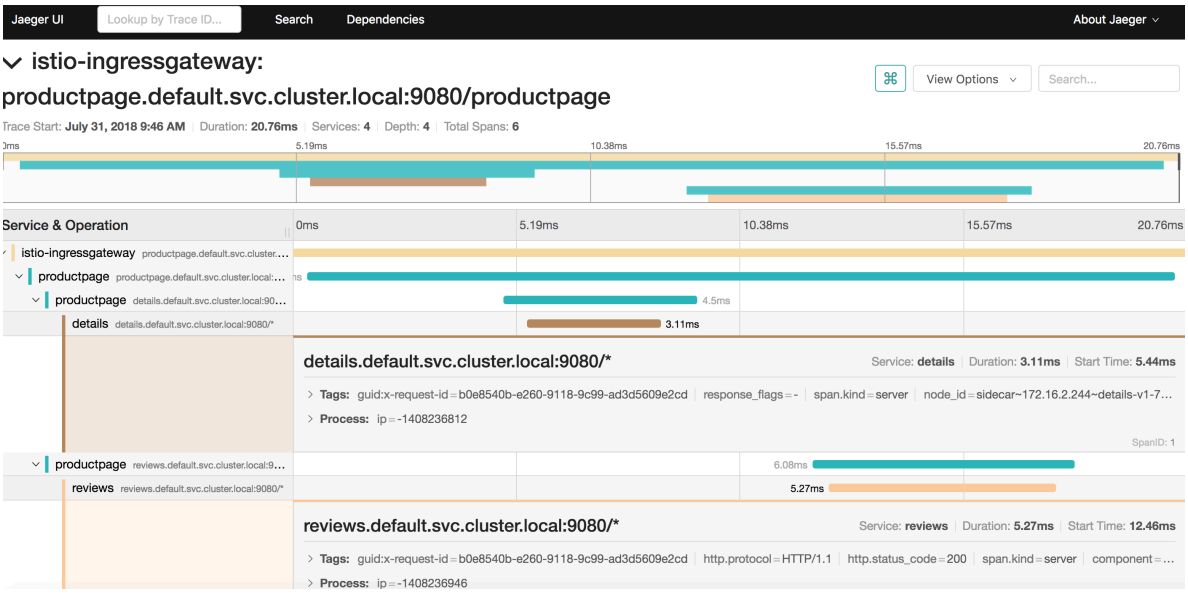


3. 在左侧选择Service，然后选择各项配置，最后单击Find Traces，右上角显示的时刻和持续时间散点图用可视化方式呈现了结果，并提供了向下挖掘能力。

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



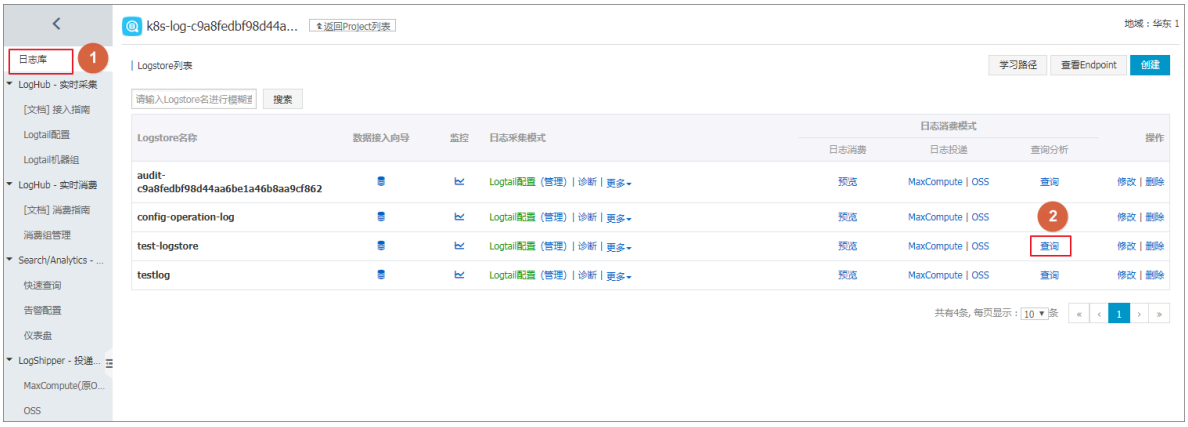
4. 用户可以选择用多种不同视图对追踪结果进行可视化，例如追踪时段内的直方图，或服务在追踪过程中的累积时间。



步骤3 查看阿里云日志服务Log Service

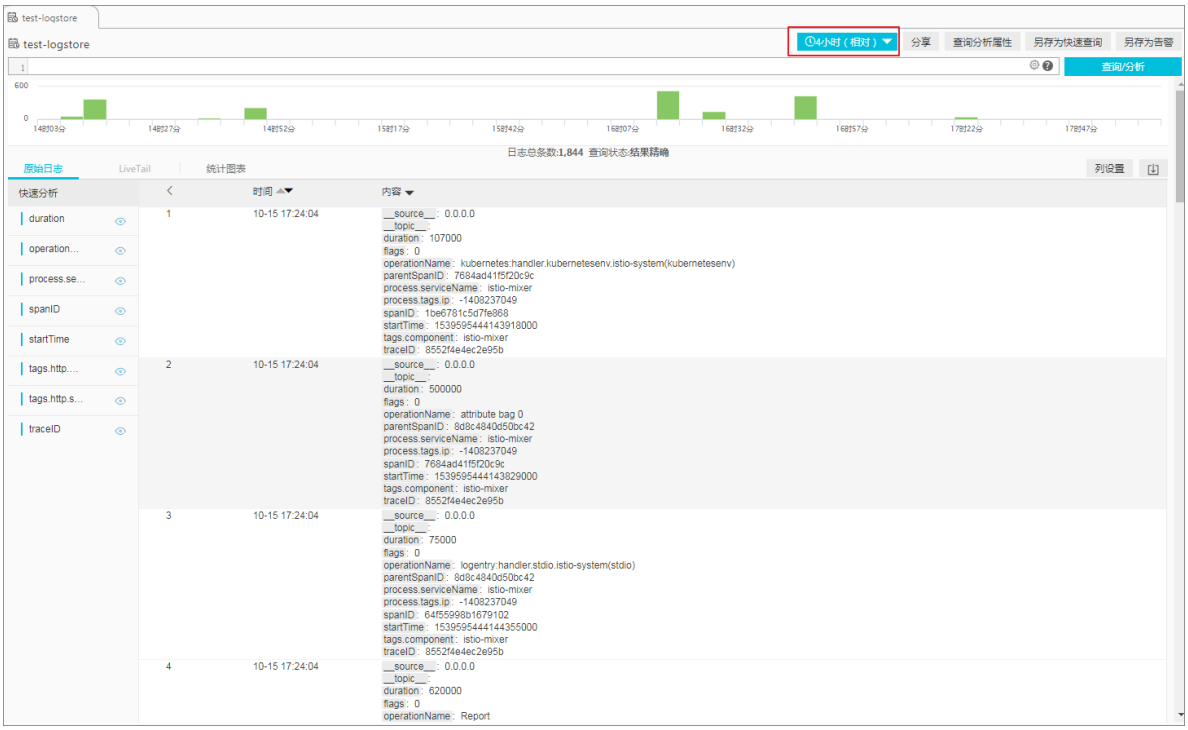
1. 登录 日志服务管理控制台。
2. 单击左侧导航栏中Project管理，选择所需的Project，并单击Project项目名称。

3. 进入Project详情页，单击查询。

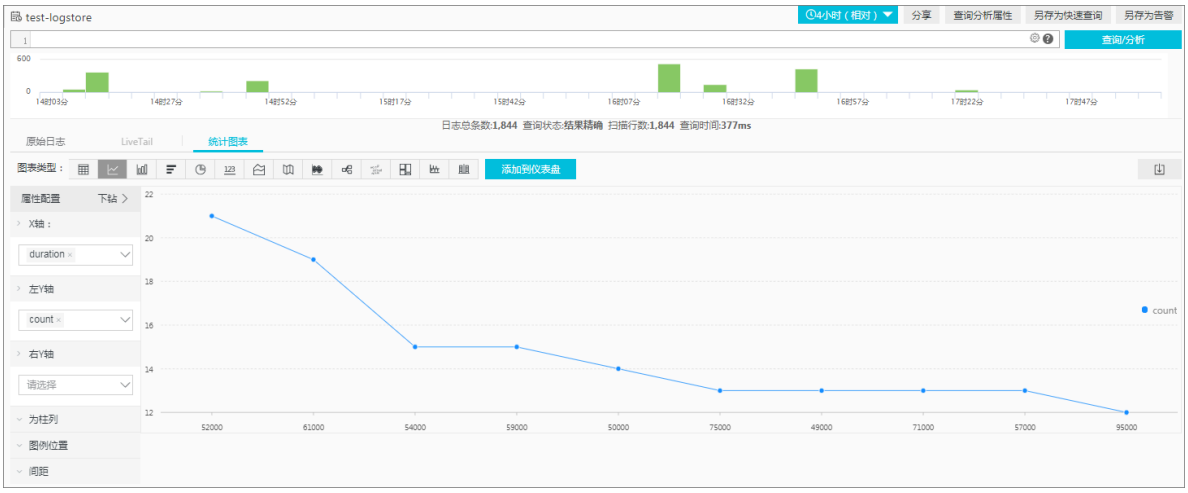


4. 选择日志查询时间，然后单击查询/分析。

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#unique_11。

> 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。

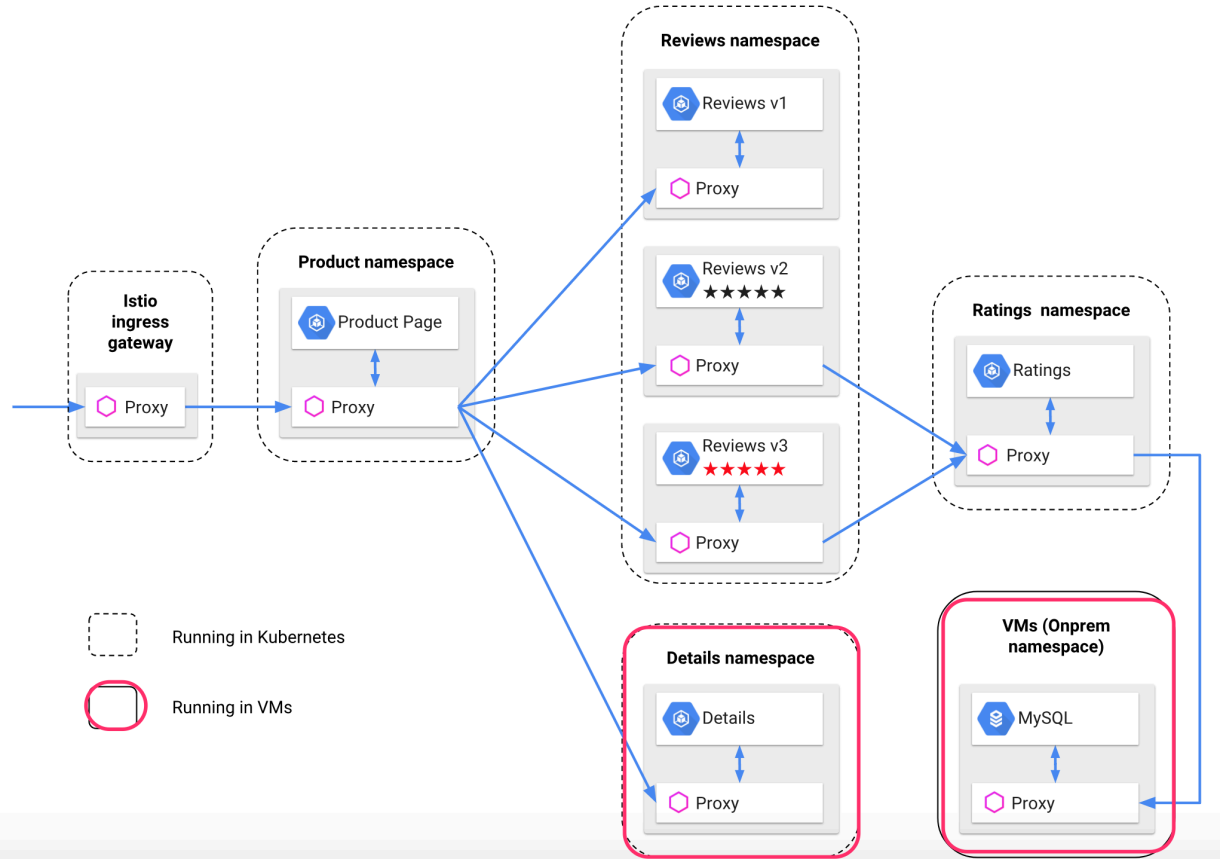
安装示例到Kubernetes集群中

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` 。

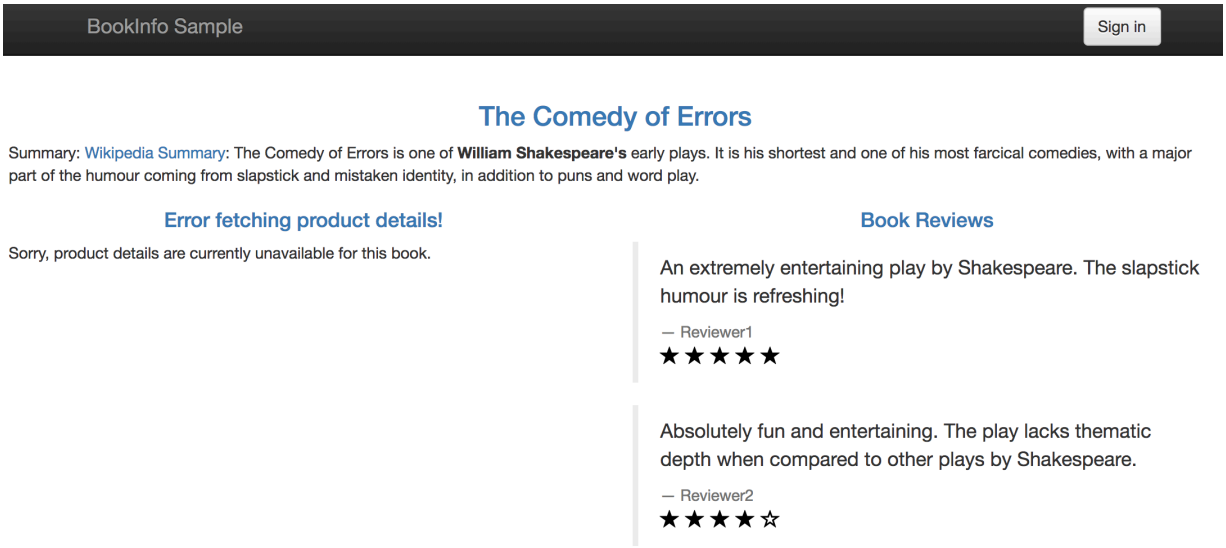
本示例中涉及到的文件可以从示例获取到。

```
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=[illegible]
   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/[illegible]
   address=/istio-policy/[illegible]
   address=/istio-telemetry/[illegible]
   address=/istio-pilot/[illegible]
   address=/istio-citadel/[illegible]
   address=/istio-ca/[illegible]
   address=/istio-policy.istio-system/[illegible]
   address=/istio-telemetry.istio-system/[illegible]
   address=/istio-pilot.istio-system/[illegible]
   address=/istio-citadel.istio-system/[illegible]
   address=/istio-ca.istio-system/[illegible]
   ```

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

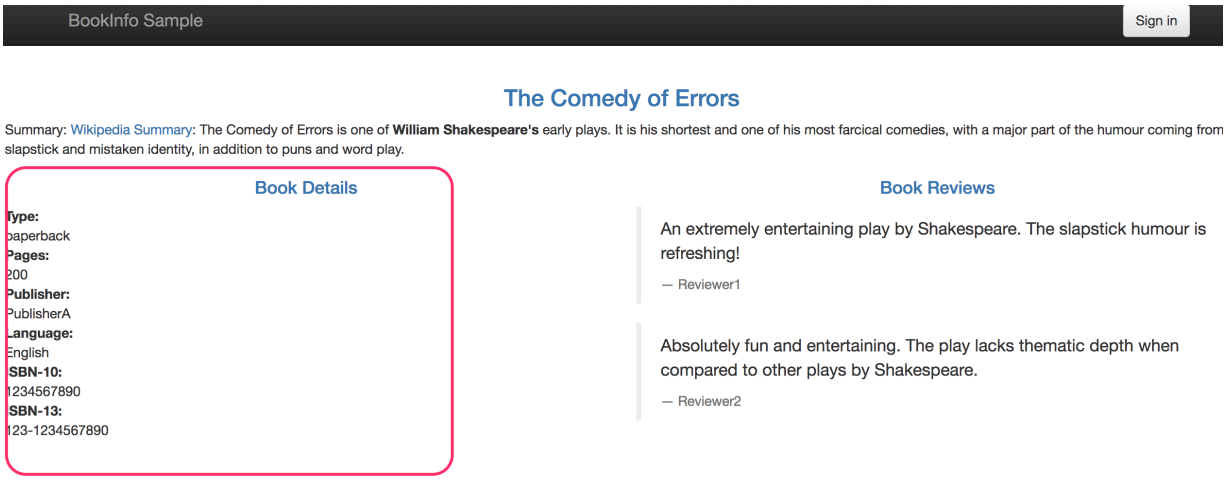
查找虚拟机的 IP 地址，用来加入服务网格：

```
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

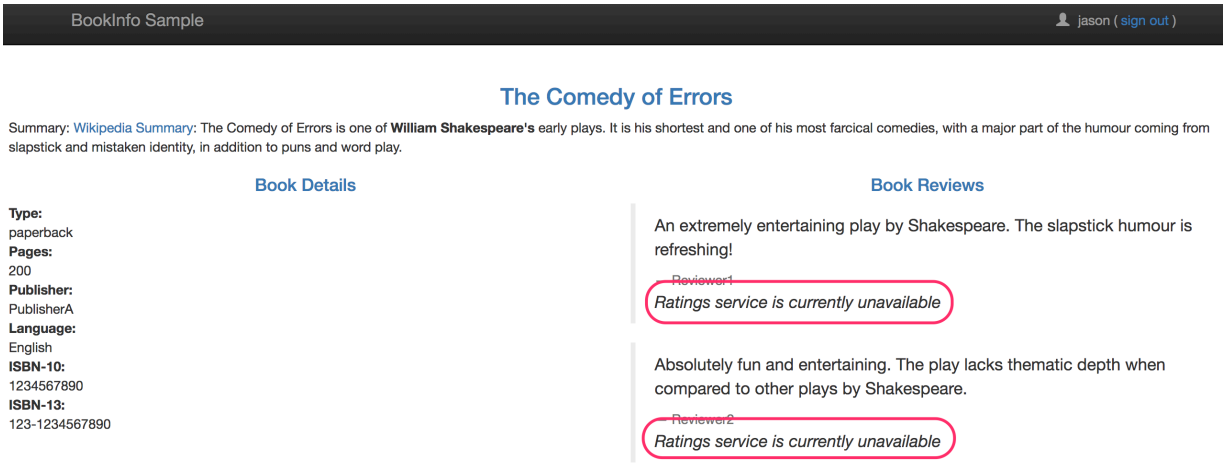


将ratings服务切到数据库版本

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
```

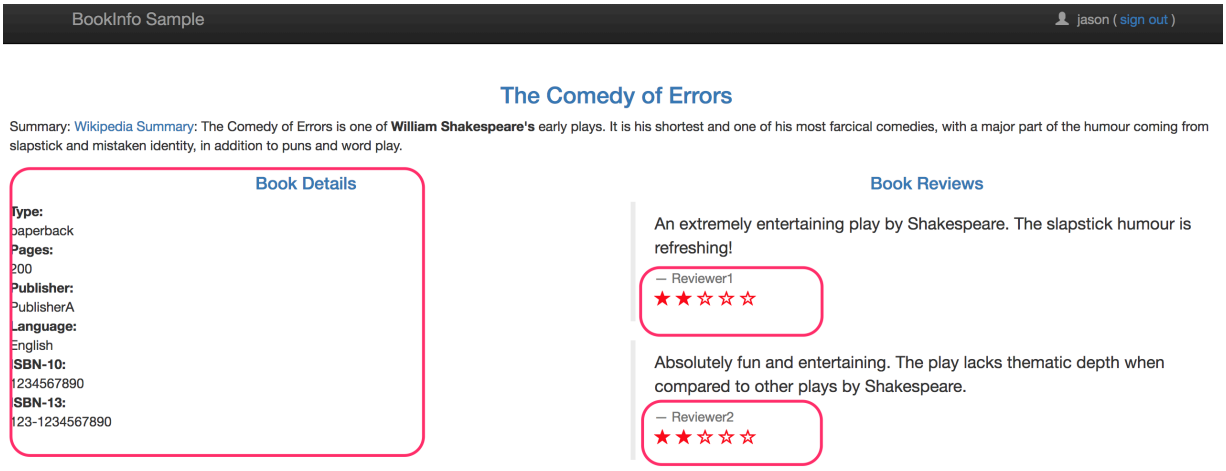
```
systemctl restart istio
```

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。

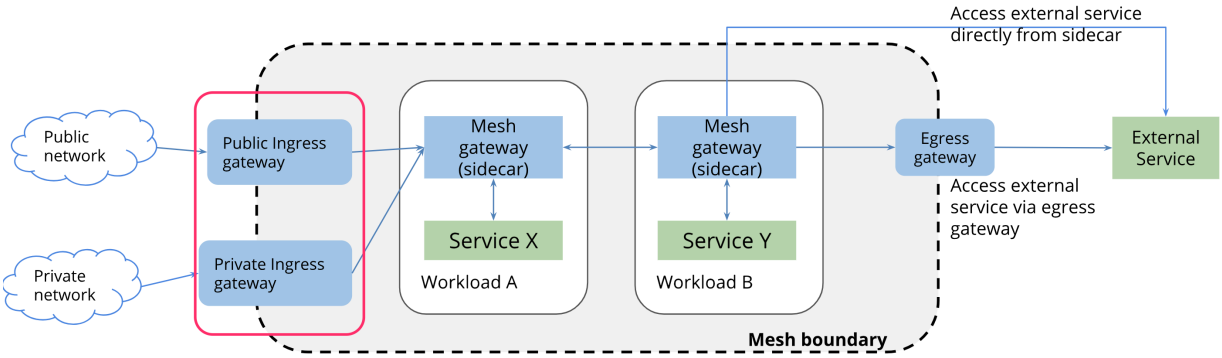
5.5 基于Istio实现TCP入口流量路由的统一管理

本文通过一个示例来讲述如何通过一个简单且标准的Istio规则来控制TCP入口流量的路由，从而实现TCP入口流量路由的统一管理。

背景信息

使用Istio的流量管理模型，将流量与基础设施扩容进行解耦，让运维人员可以通过Pilot指定流量遵循的规则。通过将流量从基础设施扩容中解耦，就可以让 Istio 提供各种独立于应用程序代码之外的流量管理功能。 这些功能都是通过部署的Envoy sidecar代理来实现的。

在一个典型的网格中，通常有一个或多个用于终结外部 TLS 链接，将流量引入网格的负载均衡器（我们称之为 Gateway），然后流量通过sidecar Gateway流经内部服务。下图描绘了网关在网格中的使用情况：



Istio Gateway 为 HTTP/TCP 流量配置了一个负载均衡，多数情况下在网格边缘进行操作，用于启用一个服务的入口（ingress）流量。网格中可以存在任意数量的 Gateway，并且多个不同的 Gateway 实现可以共存。 对于入口流量管理，可能会问为什么不直接使用 Kubernetes Ingress API？ 原因是 Ingress API 无法表达 Istio 的路由需求。 Ingress 试图在不同的 HTTP 代理之间取一个公共的交集，因此只能支持最基本的 HTTP 路由，最终导致需要将代理的其他高级功能放入到注解（annotation）中，而注解的方式在多个代理之间是不兼容的，无法移植。

此外，Kubernetes Ingress本身不支持TCP协议。因此，即使TCP不是NGINX的限制，也无法通过Ingress创建来配置NGINX Ingress Controller以进行TCP负载均衡。当然可以通过创建一个Kubernetes ConfigMap，来使用NGINX的TCP负载均衡功能，具体请参见Support for TCP /UDP Load Balancing 。因此，这种配置在多个代理中无法兼容与移植。

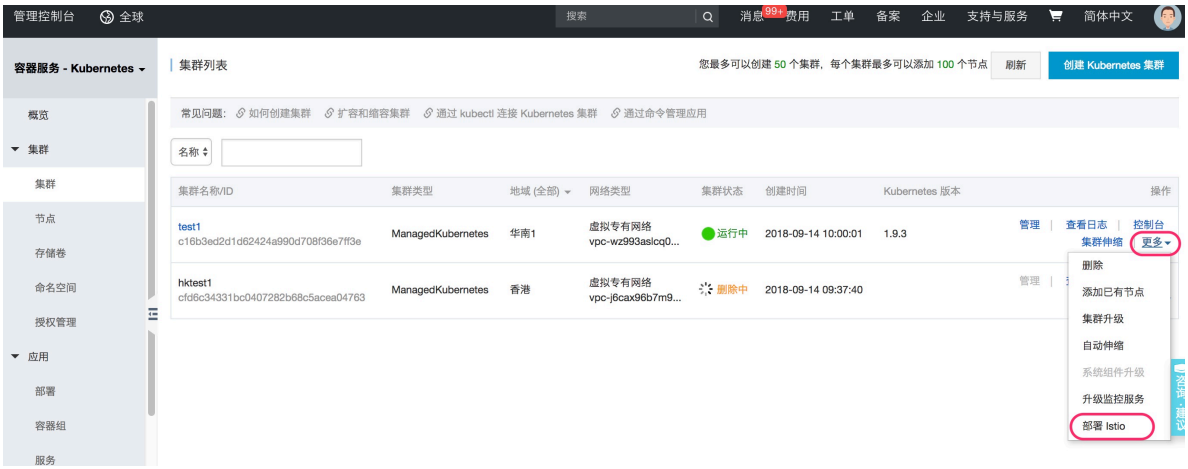
与Kubernetes Ingress 不同，Istio Gateway 通过将 L4-L6 配置与L7配置分离的方式克服了 Ingress 的上述缺点。 Gateway 只用于配置 L4-L6 功能（例如，对外公开的端口、TLS 配置），所有主流的代理均以统一的方式实现了这些功能。 然后，通过在 Gateway 上绑定 VirtualService 的方式，可以使用标准的 Istio 规则来控制进入 Gateway 的 HTTP 和 TCP 流量。

前提条件

- 您已成功创建一个 Kubernetes 集群，且集群版本为1.11.2及以上。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。
- 您已经安装 nc。如果 nc 未被安装，根据操作系统的不同， 使用yum 或 apt-get 等工具进行安装。

部署Istio

1. 登录容器服务管理控制台。
2. 在 Kubernetes 菜单下，选择集群 > 集群，进入集群列表页面。
3. 选择所需的集群并单击操作列更多 > 部署Istio。



4. 根据如下信息，部署Istio：

配置	说明
集群	部署Istio的目标集群。
命名空间	部署Istio的命名空间。
发布名称	发布的Istio名称。
启用 Prometheus 度量日志收集	是否启用Prometheus 收集度量日志。默认情况下启用。
启用 Grafana 度量展示	是否启用Grafana 展示指标的度量数据。默认情况下启用。
启用 Sidecar 自动注入	是否启用 Sidecar 进行容器的自动注入。默认情况下启用。
启用 Kiali 可视化服务网格	是否启用 Kiali 可视化服务网格。默认情况下不启用。 · 用户名：指定用户名称。默认情况下是admin。 · 密码：指定密码。默认情况下是admin。

配置	说明
链路追踪设置	· 启用分布式追踪 Jaeger：启用该选项，则需要开通阿里云日志服务。在日志服务中将自动创建名称为 `istio-tracing-{ClusterID}` 的项目。追踪数据会存储到该项目中。 · 启用链路追踪：启用该选项，则需要开通阿里云链路追踪服务。同时需要指定对应的接入点地址。例如，接入点地址为 `http://tracing-analysis-dc-hz.aliyuncs.com/.../api/v1/spans`，表示启用该选项后，zipkin客户端根据v1版本的 API的公网（或者内网）接入点地址把采集数据传输到链路跟踪。 说明: 如果使用内网接入点，请确保Kubernetes集群与链路追踪服务在相同区域，保证网络互通。
Pilot设置	跟踪采样百分比（0-100）：默认取值为1。
控制Egress流量	· 直接放行对外访问的地址范围：Istio 服务网格内的服务可以直接对外访问的地址范围。默认情况下为空，使用英文半角逗号分隔。 · 拦截对外访问的地址范围：拦截直接对外访问的地址范围。默认情况下，已包含集群 Pod CIDR 与 Service CIDR ，使用英文半角逗号分隔。 · 勾选全部选项则会拦截所有对外访问的地址。 说明: 直接放行对外访问的地址范围的优先级高于拦截对外访问的地址范围。 例如，您将同一个IP地址同时配置在直接放行对外访问的地址与拦截对外访问的地址时，您仍可以直接访问此地址，即直接放行对外访问的地址范围生效。

5. 单击部署Istio，启动部署。

   在部署页面下方，可实时查看部署进展及状态。

TCP Server镜像

你可以直接使用已经构建好的镜像文件：`registry.cn-hangzhou.aliyuncs.com/wangxining/tcptest:0.1`或者安装如下操作构建TCP Server镜像。

1. 从`https://github.com/osswangxining/Istio-TCPRoute-Sample`地址克隆代码库。
2. 切换到代码目录可以看到一个Dockerfile。

```
.
├── Dockerfile
├── Godeps
│   ├── Godeps.json
│   └── Readme
├── LICENSE
├── README.md
├── k8s
│   ├── deployment.yml
│   ├── destination-rule-all.yaml
│   ├── gateway.yaml
│   ├── service.yml
│   └── virtualservice.yaml
├── tcp-echo.go
└── vendor
    └── github.com
        └── nu7hatch
            └── gouuid
                ├── COPYING
                ├── README.md
                └── uuid.go

6 directories, 14 files
```

3. 运行以下命令构建镜像。

```
docker build -t {######} .
```

部署应用

1. 执行如下命令，部署服务。

```
cd k8s
kubectl apply -f deployment.yml
kubectl apply -f service.yml
```

说明：

上面的命令会创建1个Service（`tcp-echo`）与2个Deployment（`tcp-echo-v1`和`tcp-echo-v2`），其中这2个Deployment都包含了标签*app: tcp-echo*，并且Service（`tcp-echo`）对应到上述2个Deployment：

```
 selector:
     app: "tcp-echo"
```

2. 执行`kubectl get pods --selector=app=tcp-echo`，确认TCP Server的Pod启动。

```
NAME                              READY     STATUS    RESTARTS   AGE
tcp-echo-v1-7c775f57c9-frprp      2/2       Running   0          1m
tcp-echo-v2-6bcfd7dcf4-2sqhf      2/2       Running   0          1m
```

3. 执行`kubectl get service --selector=app=tcp-echo`，确认TCP Server的服务。

```
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
tcp-echo   ClusterIP   172.19.46.255   <none>        3333/TCP   17h
```

定义 gateway

创建并拷贝内容到*gateway.yaml*文件中，并执行`kubectl apply -f gateway.yaml`，创建2个Gateway，其中一个Gateway监听31400端口，另一个Gateway监听31401端口。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: tcp-echo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 31400
      name: tcp
      protocol: TCP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
```

```
metadata:
  name: tcp-echo-gateway-v2
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 31401
      name: tcp
      protocol: TCP
    hosts:
    - "*"
```

这两个 Gateway共用一个 ingressgateway 服务，该 ingressgateway 使用了 Loadbalancer 方式暴露，可提供对外使用的IP地址。

istio-ingressgateway	LoadBalancer	2018-10-17 14:18:37	172.19.68.224	istio-ingressgateway:80 TCP istio-ingressgateway:31380 TCP istio-ingressgateway:443 TCP istio-ingressgateway:31390 TCP istio-ingressgateway:31400 TCP istio-ingressgateway:31400 TCP istio-ingressgateway:31401 TCP istio-ingressgateway:31401 TCP istio-ingressgateway:15011 TCP istio-ingressgateway:31285 TCP istio-ingressgateway:8060 TCP istio-ingressgateway:32571 TCP istio-ingressgateway:853 TCP istio-ingressgateway:30849 TCP istio-ingressgateway:15030 TCP istio-ingressgateway:31258 TCP istio-ingressgateway:15031 TCP istio-ingressgateway:30671 TCP

创建Istio规则

创建并拷贝如下内容到*destination-rule-all.yaml*和*virtualservice.yaml*文件中。并分别执行`kubectl apply -f destination-rule-all.yaml`和`kubectl apply -f virtualservice.yaml`命令。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: tcp-echo
spec:
  hosts:
  - "*"
  gateways:
  - tcp-echo-gateway
  - tcp-echo-gateway-v2
  tcp:
  - match:
    - port: 31400
      gateways:
        - tcp-echo-gateway
    route:
    - destination:
        host: tcp-echo.default.svc.cluster.local
        subset: v1
```

```
        port:
          number: 3333
  - match:
    - port: 31401
      gateways:
        - tcp-echo-gateway-v2
    route:
    - destination:
        host: tcp-echo.default.svc.cluster.local
        subset: v2
        port:
          number: 3333
```

体验TCP路由功能

1. **查看Ingress Gateway的地址。**

   单击左侧导航栏中的服务，在右侧上方选择对应的集群和命名空间，在列表中找到istio-ingressgateway的外部端点地址。

2. **打开终端，运行以下命令。**

```
nc INGRESSGATEWAY_IP 31400
```

3. **输入文字进行如下交互，可以看到该端口的TCP流量转发到了版本v1对应的Pod。**

```
Welcome, you are connected to node cn-beijing.i-2zeij4aznsu1dvd4mj5c
.
Running on Pod tcp-echo-v1-7c775f57c9-frprp.
In namespace default.
With IP address 172.16.2.90.
Service default.
hello, app1
hello, app1
continue..
continue..
```

4. **查看版本v1的Pod的日志。**

```
kubectl logs -f tcp-echo-v1-7c775f57c9-frprp -c tcp-echo-container
 | grep Received
2018/10/17 07:32:29 6c7f4971-40f1-4f72-54c4-e1462a846189 - Received
Raw Data: [104 101 108 108 111 44 32 97 112 112 49 10]
2018/10/17 07:32:29 6c7f4971-40f1-4f72-54c4-e1462a846189 - Received
Data (converted to string): hello, app1
2018/10/17 07:34:40 6c7f4971-40f1-4f72-54c4-e1462a846189 - Received
Raw Data: [99 111 110 116 105 110 117 101 46 46 10]
2018/10/17 07:34:40 6c7f4971-40f1-4f72-54c4-e1462a846189 - Received
Data (converted to string): continue..
```

5. **打开另外一个终端，运行以下命令。**

```
nc INGRESSGATEWAY_IP 31401
```

6. **输入文字进行如下交互，可以看到该端口的TCP流量转发到了版本v2对应的Pod。**

```
Welcome, you are connected to node cn-beijing.i-2zeij4aznsu1dvd4mj5b
.
Running on Pod tcp-echo-v2-6bcfd7dcf4-2sqhf.
```

```
In namespace default.
With IP address 172.16.1.95.
Service default.
hello, app2
hello, app2
yes,this is app2
yes,this is app2
```

7. 查看版本v2的POD的日志。

```
kubectl logs -f tcp-echo-v2-6bcfd7dcf4-2sqhf -c tcp-echo-container
 | grep Received
2018/10/17 07:36:29 1a70b9d4-bbc7-471d-4686-89b9234c8f87 - Received
Raw Data: [104 101 108 108 111 44 32 97 112 112 50 10]
2018/10/17 07:36:29 1a70b9d4-bbc7-471d-4686-89b9234c8f87 - Received
Data (converted to string): hello, app2
2018/10/17 07:36:37 1a70b9d4-bbc7-471d-4686-89b9234c8f87 - Received
 Raw Data: [121 101 115 44 116 104 105 115 32 105 115 32 97 112 112
50 10]
2018/10/17 07:36:37 1a70b9d4-bbc7-471d-4686-89b9234c8f87 - Received
Data (converted to string): yes,this is app2
```

5.6 基于Istio实现服务的灰度发布

为解决微服务的分布式应用架构在运维、调试、和安全管理等维度存在的问题，可通过部署Istio创建微服务网络，并提供负载均衡、服务间认证以及监控等能力。阿里云容器服务Kubernetes基于Istio提供了服务治理能力，对应用服务提供多版本管理、灵活的流量治理策略，以支持多种灰度发布场景。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见#unique_11。
- 您已成功在该集群中部署Istio。

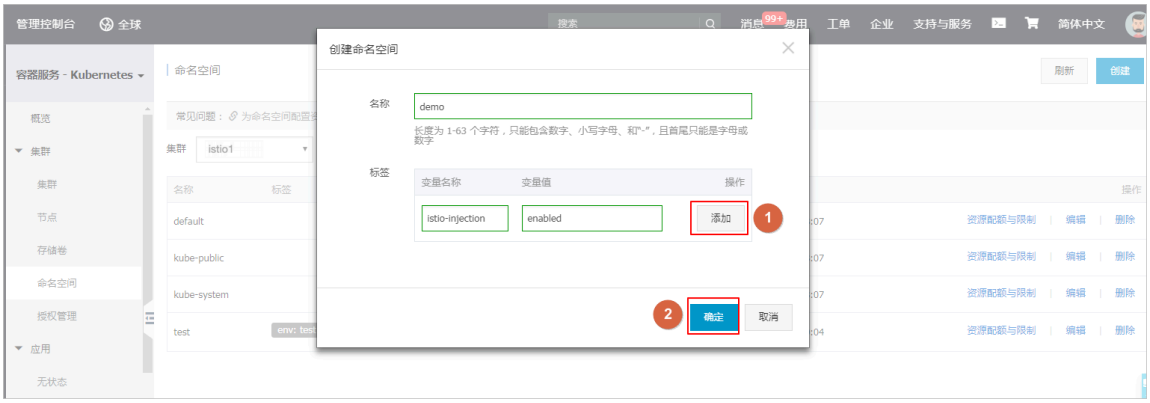
操作步骤

1. 启用Sidecar自动注入。

   a. 登录 容器服务管理控制台。

   b. 单击左侧导航栏中的集群 > 命名空间，进入命令空间页面。

   c. 单击创建，创建一个新的命名空间。



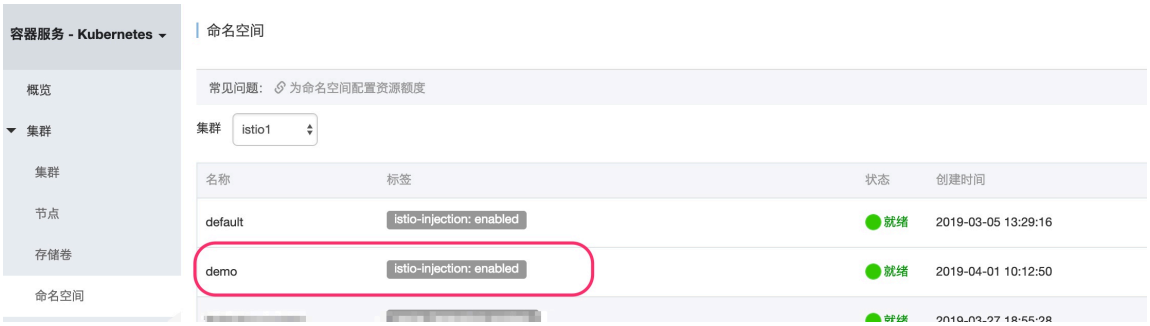
   d. 在弹出的创建命名空间对话框中，输入名称为demo，添加以下标签后，单击添加。

      - 变量称为`istio-injection`
      - 变量值为`enabled`

> **说明:**
> 注意一定要单击添加后再单击确定按钮，以确保输入的标签添加成功。

   e. 单击确定。

   f. 查看添加的命名空间。

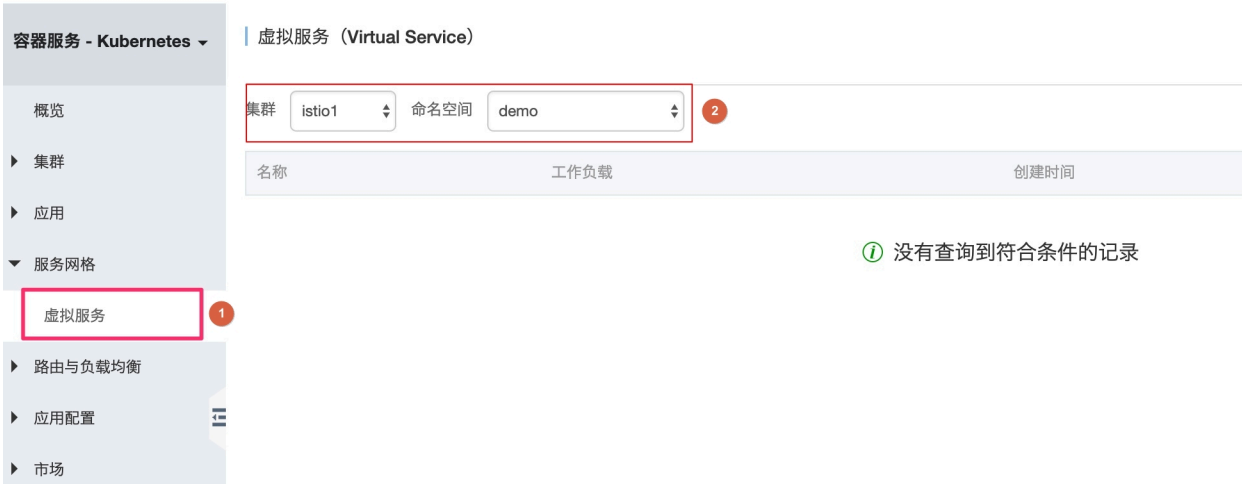


2. 创建虚拟服务（Virtual Service）。

   a. 登录 容器服务管理控制台。

   b. 单击左侧导航栏中服务网格 > 虚拟服务，进入虚拟服务（Virtual Service）页面。

   c. 单击创建，创建一个新的虚拟服务。



   d. 在创建应用页面进行设置，然后单击下一步 进入容器配置页面。

      - 应用名称：设置应用的名称。
      - 应用版本：设置应用的版本，此处用于做多版本管理。
      - 部署集群：设置应用部署的集群。
      - 命名空间：设置应用部署所处的命名空间，此处是指步骤1所创建的命名空间demo。
      - 副本数量：即应用包含的Pod数量。

   e. 设置容器配置。本例中指定镜像名称为nginx，镜像版本为1.14，其他配置可以参考镜像创建无状态Deployment应用中的设置容器配置步骤。完成容器配置后，单击下一步进入服务配置页面。

f. 设置服务配置。

A. 在服务栏单击创建，在弹出的对话框中进行配置。

- 名称：可自主设置，例如`nginx`。
- 类型：虚拟集群 IP。选择该项，才能通过Istio服务网格来对服务进行治理。
- 端口映射：需要添加服务端口的名称、服务端口和容器端口，及TCP协议。服务端口必须进行命名。端口名称只允许是<协议>[-<后缀>-]模式，其中<协议>部分可选择范围包

括 grpc、http、http2、https、mongo、redis、tcp、tls 以及 udp，Istio 可以通过对这些协议的支持来提供路由能力。

- 注解：为该服务添加一个注解（annotation）。
- 标签：您可为该服务添加一个标签，标识该服务。

B. 配置完成后，单击创建。

g. 单击创建，所有配置完成后，默认进入创建完成页面，会列出应用包含的对象，您可以单击查看应用详情进行查看。

3. **创建服务客户端sleep应用。**

   a. **将如下内容保存为*sleep.yaml*文件。**

```
apiVersion: v1
kind: Service
metadata:
  name: sleep
  labels:
    app: sleep
spec:
  ports:
  - port: 80
    name: http
  selector:
    app: sleep
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: sleep
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: sleep
    spec:
      containers:
      - name: sleep
        image: pstauffer/curl
        command: ["/bin/sleep", "3650d"]
        imagePullPolicy: IfNotPresent
```

   b. **在命名空间demo下执行如下命令。**

```
kubectl apply -f sleep.yaml -n demo
```

   c. **执行`kubectl exec -it -n demo <podName> bash`命令，登录到sleep应用的pod中，执行如下命令调用nginx服务。**

```
for i in `seq 1000`
```

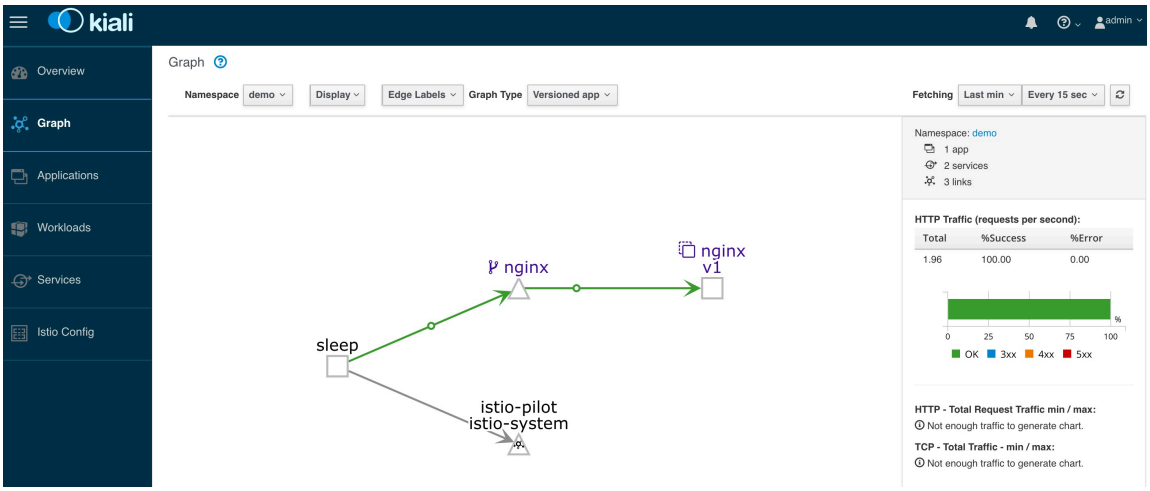
```
do
curl -I http://nginx.demo:8080;
echo '';
sleep 1;
done;
```

当出现如红框所示内容时，显示调用成功返回。

```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

d. 如果在安装Istio时打开启用 Kiali 可视化服务网格功能，可以通过访问Kiali控制台，查看调用效果。

请先执行`kubectl port-forward svc/kiali -n istio-system 20001:20001`命令，在本地打开20001端口。在浏览器中输入*http://localhost:20001*，打开Kiali控制台，查看调用效果。

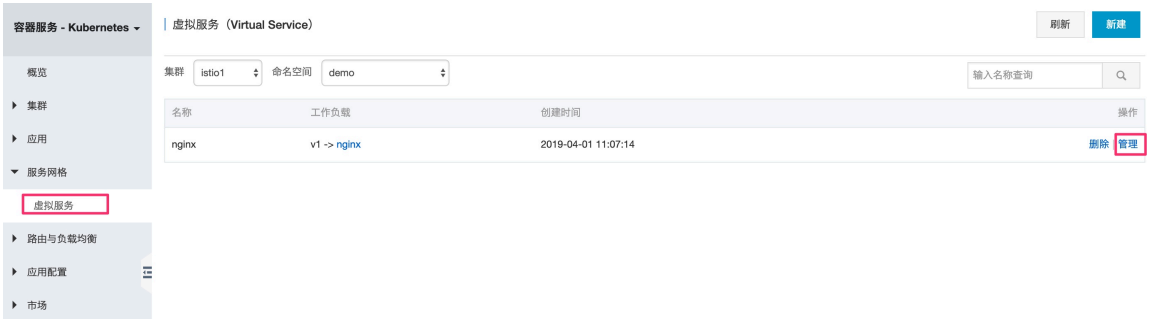


4. 增加灰度版本。

a. 登录 容器服务管理控制台。

b. 单击左侧导航栏中服务网格 > 虚拟服务，进入虚拟服务（Virtual Service）页面。

c. 单击对应虚拟服务右侧的管理。



d. 进入详细页面后，单击增加灰度版本，来增加灰度版本。

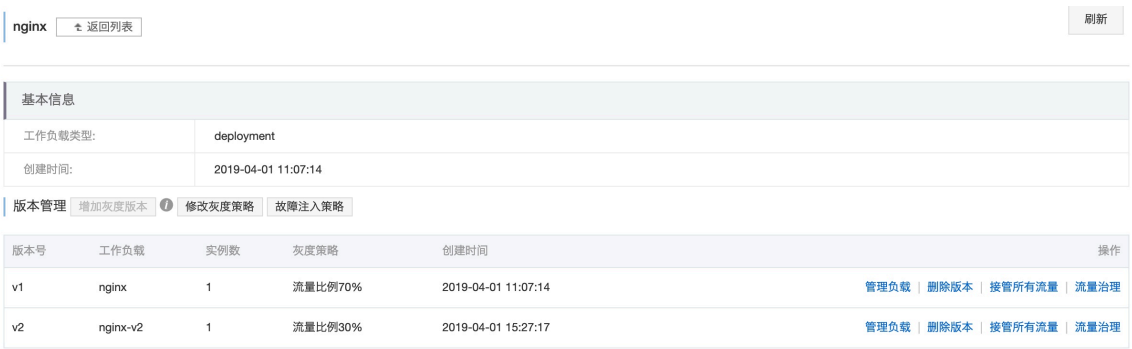
e. 设置应用版本、副本数量。然后单击下一步 进入容器配置页面。

f. 设置容器配置。本例中指定镜像名称为nginx，镜像版本为1.15，其他配置可以参考镜像创建无状态Deployment应用中的设置容器配置步骤。完成容器配置后，单击下一步。

g. 指定灰度策略，例如策略类型选择基于流量比例发布，v2流量配比为30%，v1流量配比为70%。

h. 单击创建，所有配置完成后，单击服务网格 > 虚拟服务，进入虚拟服务（Virtual Service）页面。

i. 单击对应服务右侧的管理，进入详情页面。



j. 执行`kubectl exec -it -n demo <podName> bash`命令，登录到sleep应用的pod中，执行如下命令调用nginx服务。

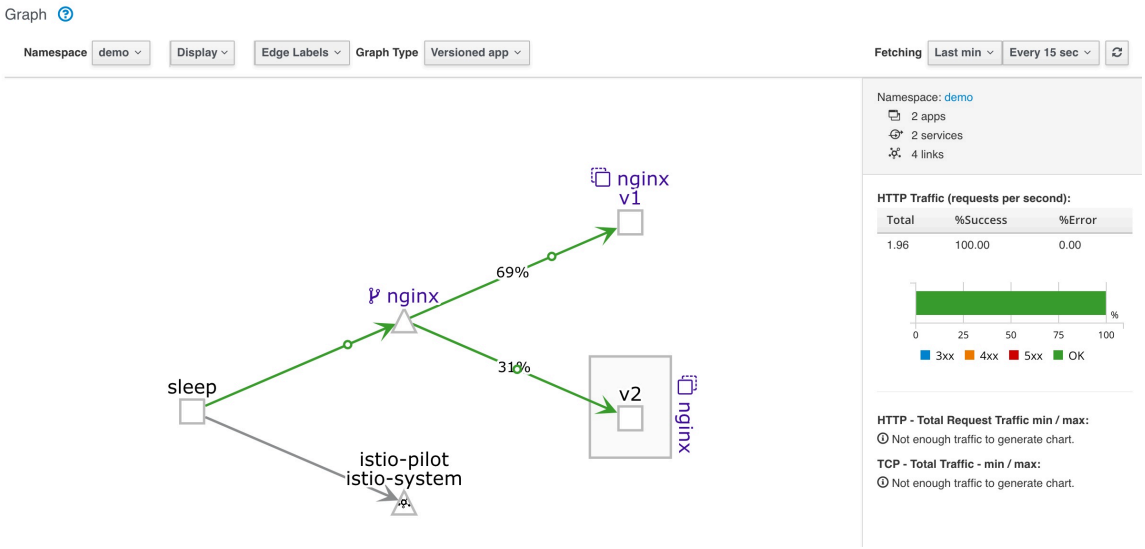
```
for i in `seq 1000`
do
curl -I http://nginx.demo:8080;
echo '';
sleep 1;
done;
```

当出现如红框所示内容时，显示调用成功返回。

```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

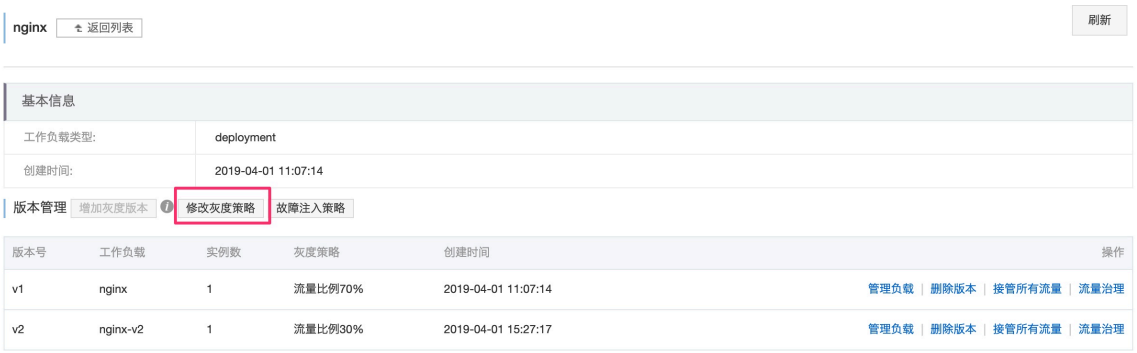
k. 如果在安装Istio时打开启用 Kiali 可视化服务网格功能，可以通过访问Kiali控制台，查看调用效果（如果已经打开请忽略）。

请先执行`kubectl port-forward svc/kiali -n istio-system 20001:20001`命令，在本地打开20001端口。在浏览器中输入*http://localhost:20001*，打开Kiali控制台，可以看到流量按照比例分发到不同的版本上。

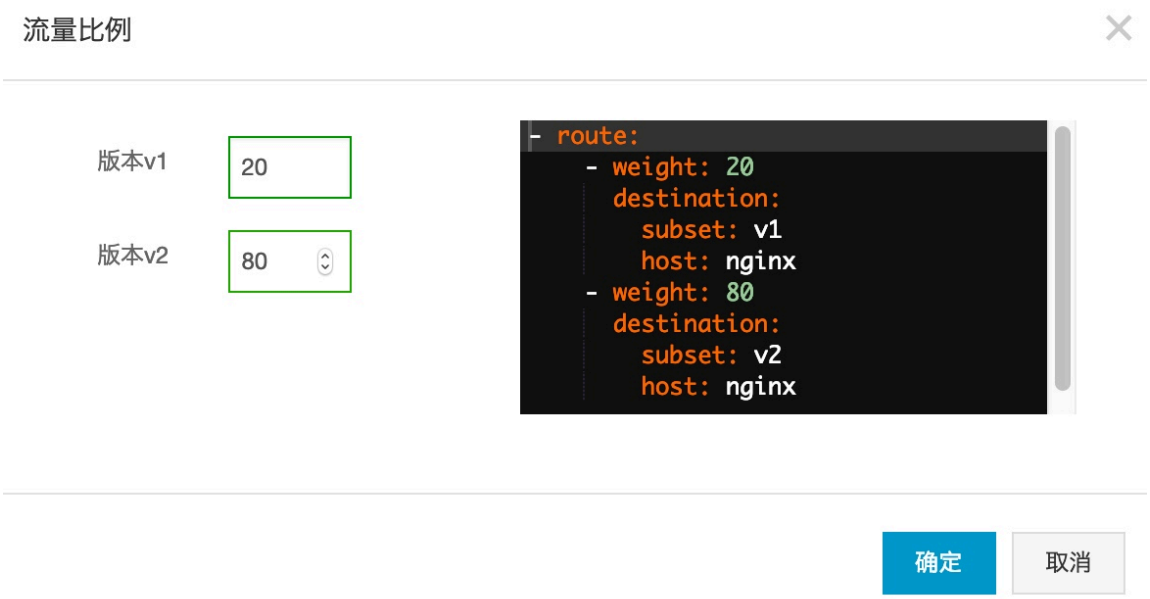


5. 修改灰度策略。

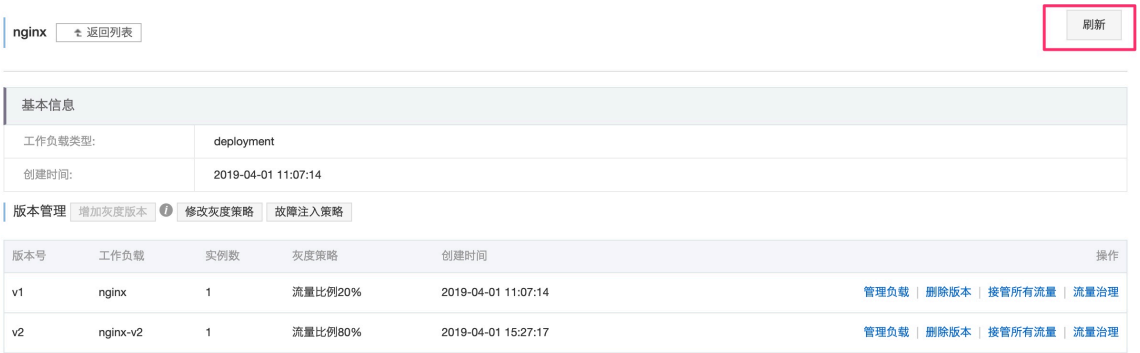
a. 在详情页中单击修改灰度策略，在弹出的对话框中进行配置。



b. 修改流量比例，版本v1为20%，版本v2为80%。单击确定。



c. 如果页面没有刷新，可以单击页面右上方的刷新。



d. 执行`kubectl exec -it -n demo <podName> bash`命令，登录到sleep应用的pod中，执行如下命令调用nginx服务。

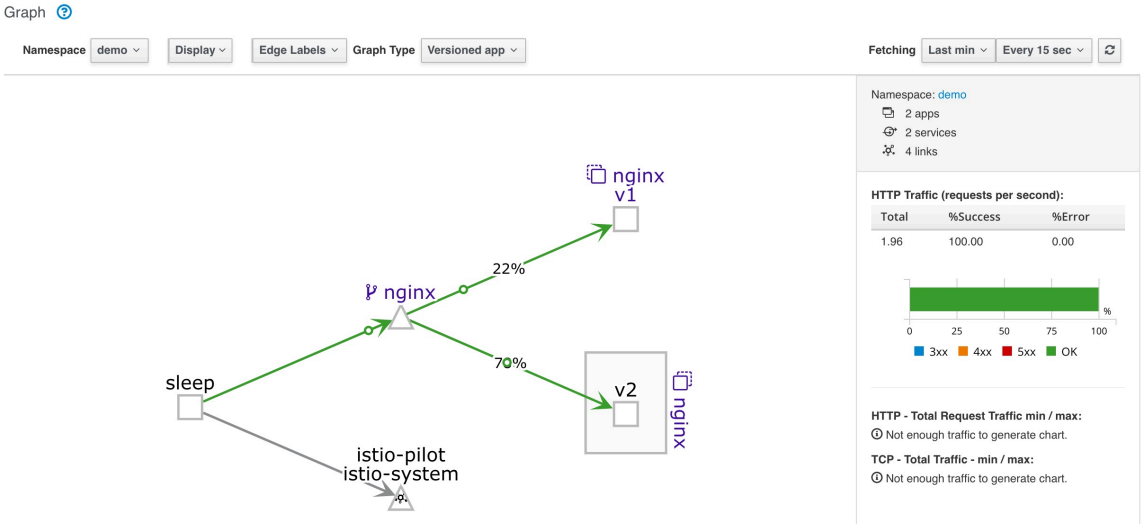
```
for i in `seq 1000`
do
curl -I http://nginx.demo:8080;
```

```
    echo '';
    sleep 1;
    done;
```

当出现如红框所示内容时，显示调用成功返回。

```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

e. 在浏览器中输入*http://localhost:20001*，打开Kiali控制台，可以看到如下调用效果。



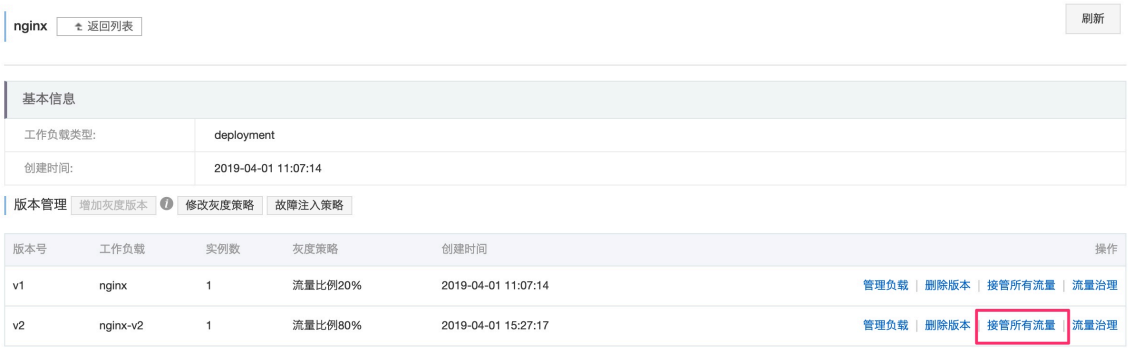
6. 接管旧版本流量。

   a. 登录 容器服务管理控制台。

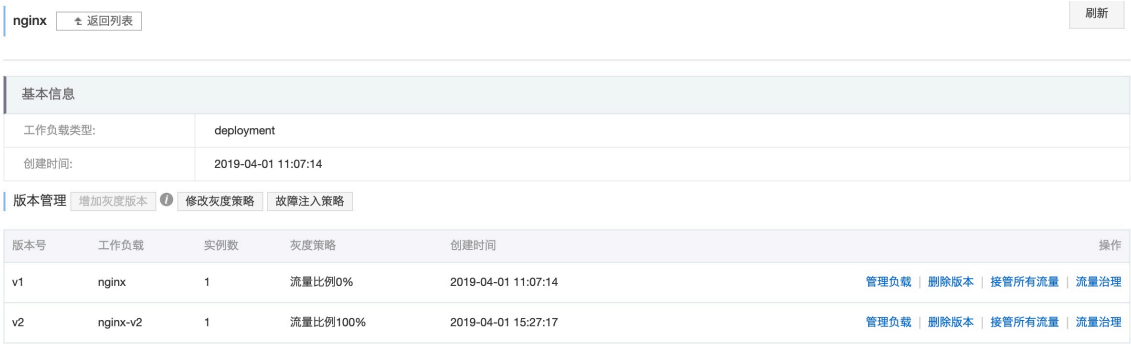
   b. 单击左侧导航栏中服务网格 > 虚拟服务，进入虚拟服务（Virtual Service）页面。

   c. 单击对应虚拟服务右侧的管理，进入详细页面。

   d. 在版本管理区域单击版本号v2右侧的接管所有流量，版本v2将会接管所有流量。



   e. 此时详情页面中，版本号v2对应的灰度策略为流量比例100%。



   f. 执行`kubectl exec -it -n demo <podName> bash`命令，登录到sleep应用的pod中，执行如下命令调用nginx服务。

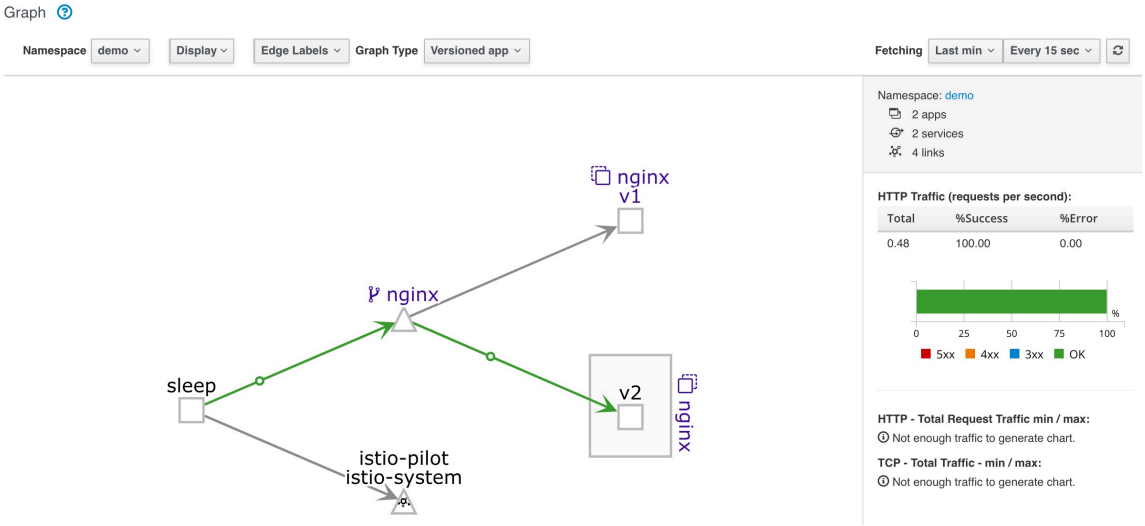
```
for i in `seq 1000`
do
curl -I http://nginx.demo:8080;
echo '';
sleep 1;
```

```
    done;
```

当出现如红框所示内容时，显示调用成功返回。

```
HTTP/1.1 200 OK
content-type: text/html; charset=utf-8
content-length: 5719
server: istio-envoy
date: Tue, 02 Apr 2019 08:21:13 GMT
x-envoy-upstream-service-time: 24
```

g. 在浏览器中输入*http://localhost:20001*，再次打开Kiali控制台，可以看到如下调用效果。



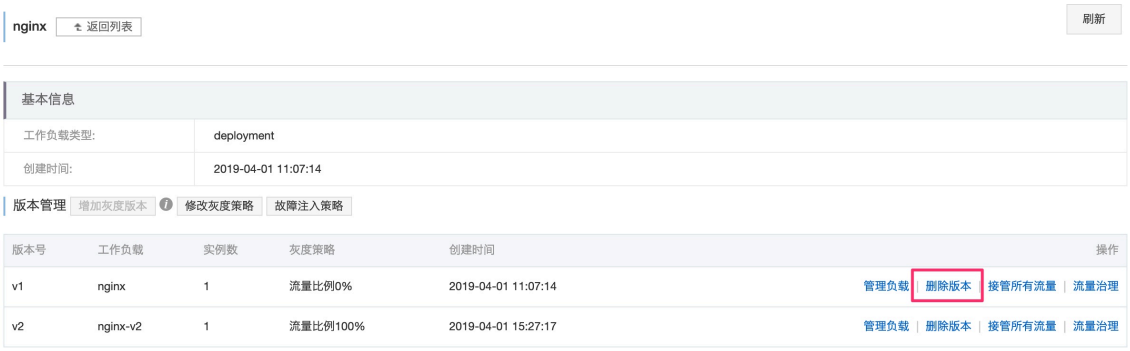
7. 下线旧版本。

   a. 登录 容器服务管理控制台。

   b. 单击左侧导航栏中服务网格 > 虚拟服务，进入虚拟服务（Virtual Service）页面。

   c. 单击对应虚拟服务右侧的管理，进入详细页面。

   d. 在版本管理区域单击版本号v1右侧的删除版本，删除v1版本。



   e. 旧版本下线之后，只有版本v2在使用中。如果有新版本v3需要灰度，则可以按照如上方式再次进行。

5.7 自定义Istio网关

本文提供了部署Istio自定义入口网关的步骤说明，以及如何使用cert-manager进行证书管理。

Istio Gateway提供多个自定义入口网关的支持能力，通过开放一系列端口用于承载网格边缘的进入连接，同时可以使用不同loadbalancer来隔离不同的入口流量。cert-manager可用于使用存储在Kubernetes Secret资源中的任意签名密钥对来获取证书。

前提条件

- Kubernetes+Istio环境已部署完成。

  阿里云容器服务Kubernetes 1.11.5目前已经支持 Istio 1.0.5的一键部署，可以通过容器服务管理控制台快速创建Kubernetes集群以及部署Istio。

- 通过CloudShell 连接 Kubernetes 集群。

生成签名密钥对

CA Issuer不会自动创建和管理签名密钥对。你可以通过以下任意一种方法获取：

- 如果您已经知道签名密钥对，需要将密钥命名为*ca.key*，证书命名为*ca.crt*。
- 如果您并不知道签名密钥对，您可以通过如下操作生成x509类型的密钥和证书。

1. 执行如下命令，生成CA密钥。

```
$ docker run -it -v $(pwd):/export frapsoft/openssl genrsa -out /
export/ca.key 2048
```

2. 执行如下命令，生成CA的自签名证书。

```
$ docker run -it -v $(pwd):/export frapsoft/openssl req -x509 -new
 -nodes -key /export/ca.key -subj "/CN=${COMMON_NAME}" -days 3650
 -reqexts v3_req -extensions v3_ca -out /export/ca.crt
```

将签名密钥对保存为Secret

通过以下命令将在默认命名空间中创建包含签名密钥对的Secret：

```
kubectl create secret tls ca-key-pair \
   --cert=ca.crt \
   --key=ca.key \
   --namespace=default
```

说明：

- 创建一个使用此密钥对生成签名证书的颁发者Issuer，为了确保颁发者Issuer可以引用到上述密钥对，我们需要将其存储在Kubernetes Secret资源中。
- 颁发者Issuer是命名空间资源，因而他们只能在自己的命名空间中引用Secrets。因此，我们将密钥对放入与Issuer相同的名称空间中。

创建引用Secret的Issuer

创建并拷贝以下内容到*certmanager-secret-issuer.yaml*文件中，并执行`kubectl apply -f certmanager-secret-issuer.yaml`命令，创建一个使用此密钥对生成签名证书的颁发者Issuer。

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: Issuer
metadata:
  name: ca-issuer
  namespace: default
spec:
  ca:
```

```
    secretName: ca-key-pair
```

获得签名证书

现在可以创建以下证书资源，该资源指定所需的证书。为了使用Issuer获取证书，您必须在与Issuer相同的命名空间中创建Certificate资源，因为Issuer是命名空间资源，如本例所示。如果您想要跨多个名称空间重用签名密钥对，那么就可以使用一个集群ClusterIssuer。首先通过以下命令为域名`myexample.com`创建证书：

创建并拷贝以下内容到*`myexample-certificate.yaml`*文件中，并执行`kubectl apply -f myexample-certificate.yaml`命令，创建签名证书。

```
apiVersion: certmanager.k8s.io/v1alpha1
kind: Certificate
metadata:
  name: myexample-certificate
  namespace: default
spec:
  secretName: istio-myexample-customingressgateway-certs
  issuerRef:
    name: ca-issuer
    # 可以通过引用ClusterIssuer类型的颁发者Issuer；默认情况使用只适用于命名空间的Issuer
    kind: Issuer
  commonName: myexample.com
  organization:
  - MyExample CA
  dnsNames:
  - myexample.com
  - www.myexample.com
```

创建证书资源后，cert-manager将尝试使用颁发者ca-issuer获取证书。如果成功，证书将存储在与证书资源相同的命名空间`defaultistio-myexample-customingressgateway-certs`中。

检查证书与密钥

由于我们已指定commonName字段，因此*`myexample.com`*将是证书的通用名称，并且通用名称和dnsNames阵列的所有元素都将是主题备用名称 （SAN）。如果我们没有指定公共名称，那么dnsNames列表的第一个元素将用作公共名称，dnsNames列表的所有元素也将是SAN。

创建上述证书后，你可以通过执行如下命令检查是否已成功获取。

如下所示查看了证书*`myexample-certificate`*：

```
kubectl describe certificate myexample-certificate
Name:         myexample-certificate
Namespace:    default
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"certmanager.k8s.io/v1alpha1","kind":"
Certificate","metadata":{"annotations":{},"name":"myexample-certificat
e","namespace":"...
API Version:  certmanager.k8s.io/v1alpha1
```

```
Kind:         Certificate
Metadata:
  Creation Timestamp:  2019-01-14T08:38:20Z
  Generation:          1
  Resource Version:    19727
  Self Link:           /apis/certmanager.k8s.io/v1alpha1/namespaces/
default/certificates/myexample-certificate
  UID:                 bf47b776-17d7-11e9-bafe-00163e069e12
Spec:
  Common Name:  myexample.com
  Dns Names:
    myexample.com
    www.myexample.com
  Issuer Ref:
    Kind:  Issuer
    Name:  ca-issuer
  Organization:
    MyExample CA
  Secret Name:  istio-myexample-customingressgateway-certs
Status:
  Conditions:
    Last Transition Time:  2019-01-14T08:38:22Z
    Message:               Certificate issued successfully
    Reason:                CertIssued
    Status:                True
    Type:                  Ready
Events:
  Type    Reason      Age   From          Message
  ----    ------      ----  ----          -------
  Normal  IssueCert   80s   cert-manager  Issuing certificate...
  Normal  CertIssued  80s   cert-manager  Certificate issued
successfully
```

当最后一行出现`Certificate issued successfully`时，表示证书创建成功。

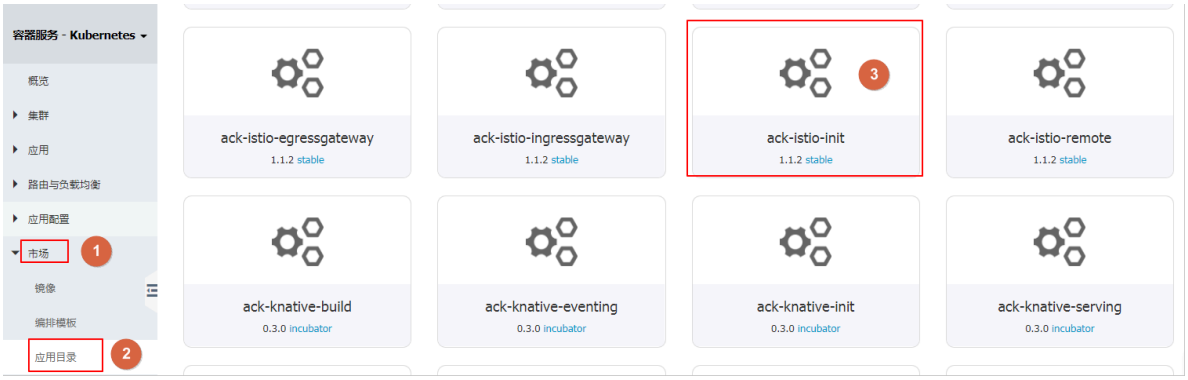
您还可以执行`kubectl get secret istio-myexample-customingressgateway-certs -o yaml`，检查Issuer是否成功，如果成功，可以看到base64编码的签名TLS密钥对。

获得证书后，cert-manager将继续检查其有效性，并在接近到期时尝试更新。例如，证书上的`Not After`字段值a，当a＜当前时间+30天时，cert-manager认为证书即将到期。对于基于CA的颁发者，cert-manager将颁发证书，其中`Not After`字段设置为当前时间+365天。

部署Istio-init

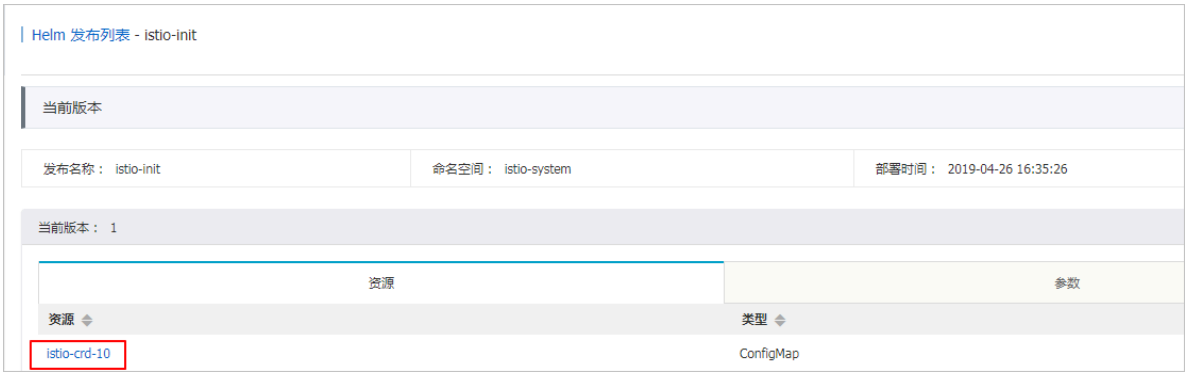
1. 登录 容器服务管理控制台。

2. 选择市场 > 应用目录，在右侧选中ack-istio-init。



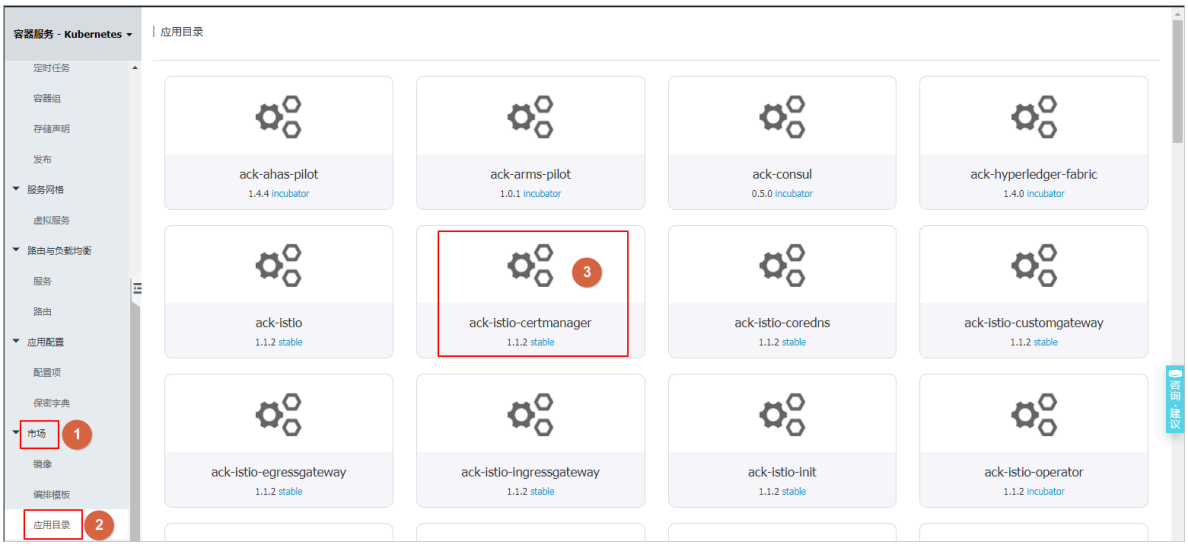
3. 在应用目录 - ack-istio-init中，在右侧的创建页面，在右侧选择对应的集群，同时可以看到命名空间已设定为 `istio-system`，发布名称已设定为`istio-init`，然后单击创建。

   几秒钟后，在弹出Helm 发布列表 - istio-init页面，可以看到Istio CRD在集群中被创建出来。

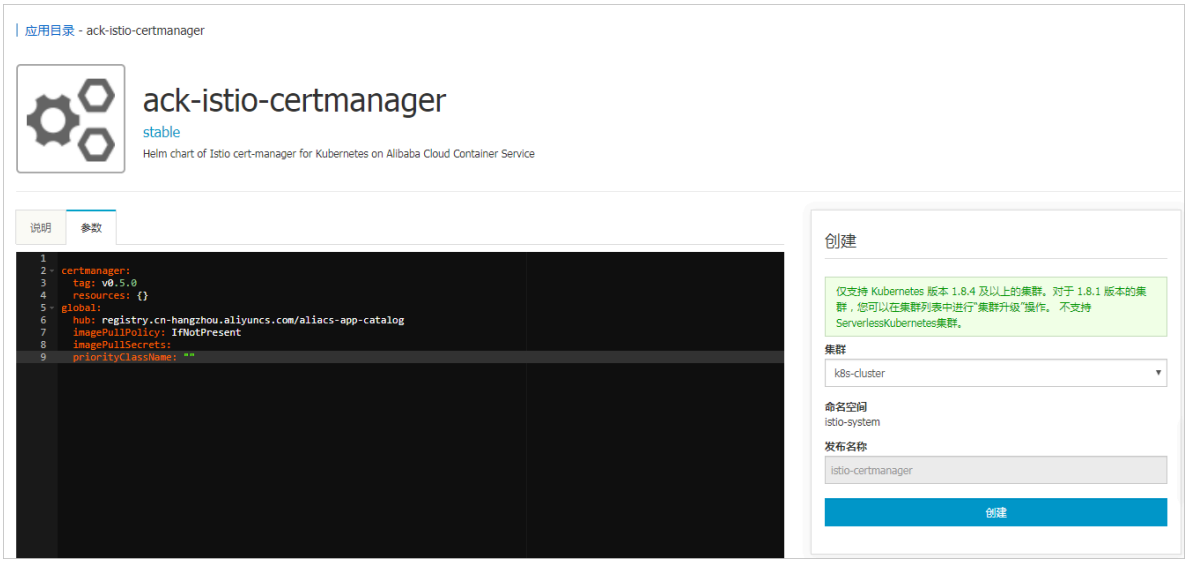


通过应用目录简便部署 Istio certmanager

1. 在 Kubernetes 菜单下，在左侧导航栏选择市场 > 应用目录，在右侧选中ack-istio-certmanager。

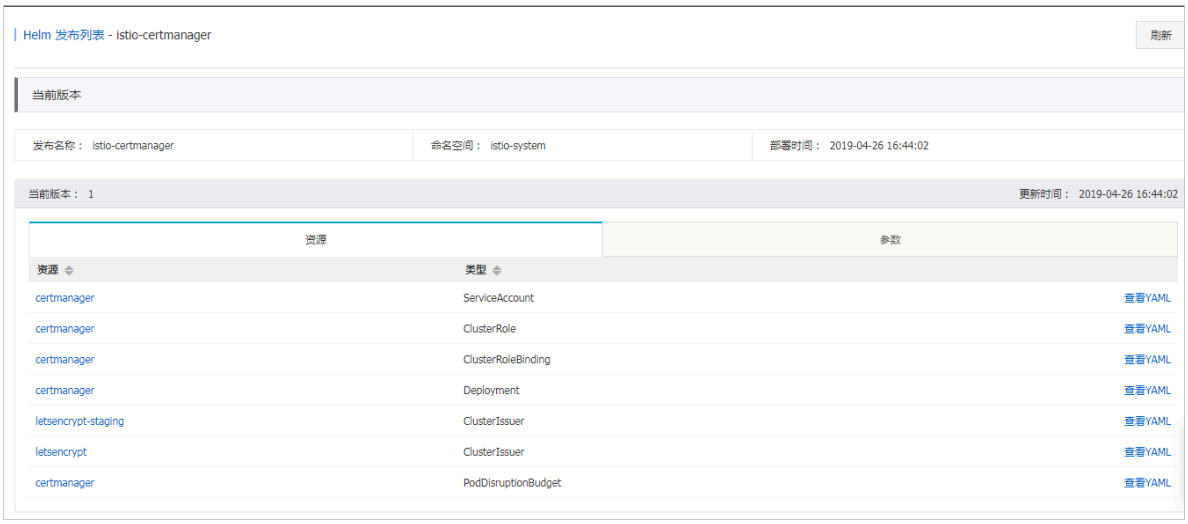


2. 在应用目录 - ack-istio-certmanager中，单击参数，可以通过修改参数配置进行定制化，如下所示。



3. 在右侧的创建页面，在右侧选择对应的集群，同时可以看到命名空间已设定为 `istio-system`，发布名称已设定为`istio-certmanager`，然后单击创建。

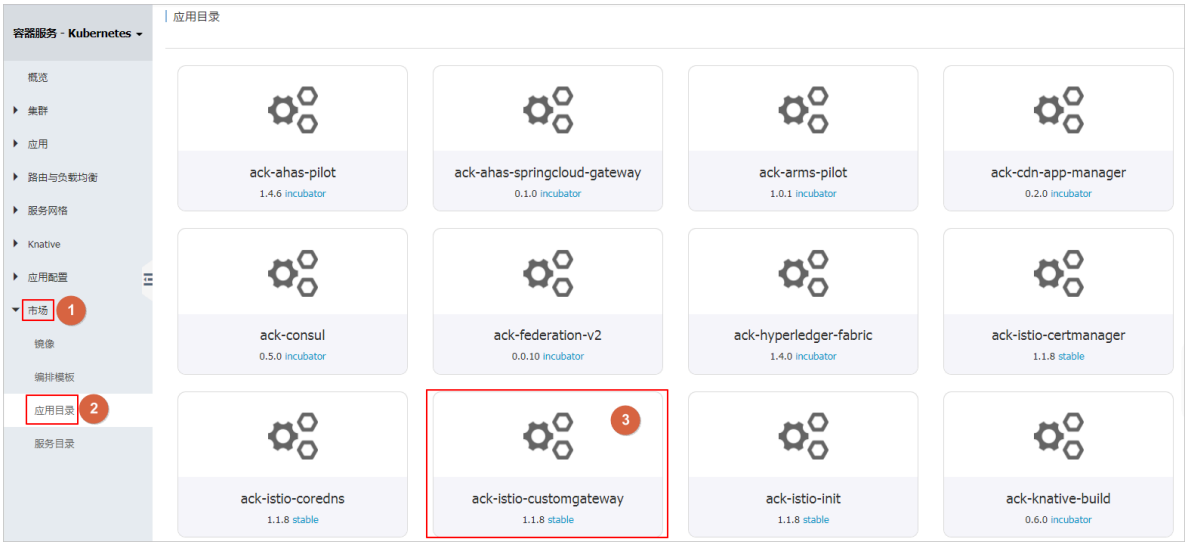
   几秒钟后，在弹出Helm 发布列表 - istio-certmanager页面，Istio certmanager发布在集群中被创建出来，如下图所示容器组certmanager的启动日志，可以看到certmanager已经成功启动。



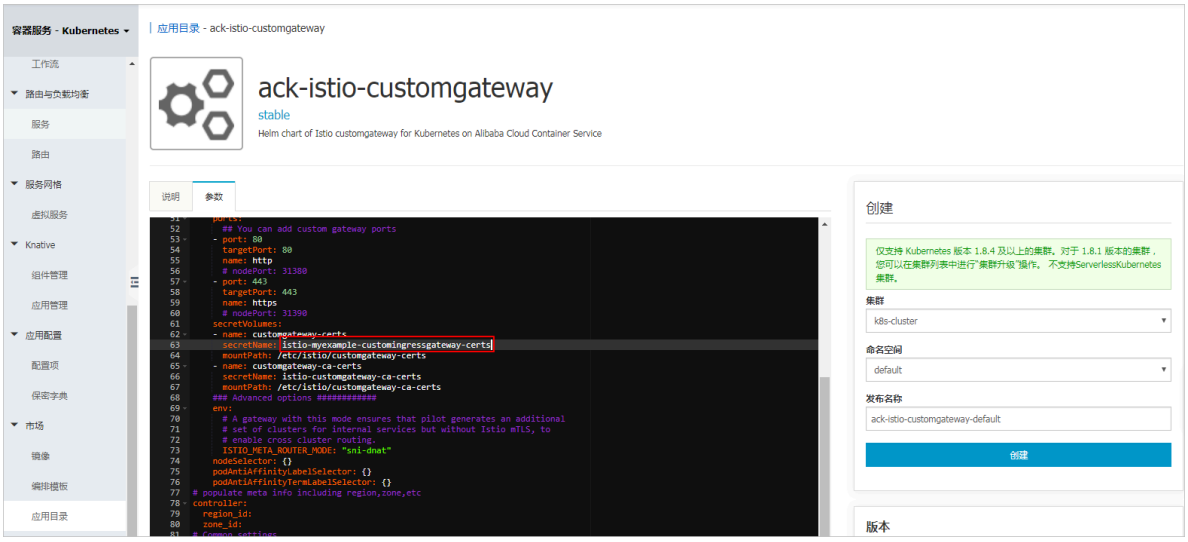
部署自定义网关

Gateway描述了在网格边缘操作的负载均衡器，用于接收传入或传出的 HTTP/TCP 连接。

1. 在 Kubernetes 菜单下，在左侧导航栏选择市场 > 应用目录，在右侧选中ack-istio-customgateway。



2. 在应用目录 - ack-istio-customgateway中，单击参数，将`istio-ingressgateway-certs`的`secretName`修改为上述创建出的 `istio-myexample-customingressgateway-certs`，修改如下所示。



3. 在右侧的创建页面，在右侧选择对应的集群，选择与保密字典`istio-myexample-customingressgateway-certs`相同的命名空间（即上文中设定的`default`），发布名称已设定为`myexample-customingressgateway`，然后单击创建。

   几秒钟后，在弹出Helm 发布列表 - istio-ingressgateway页面，自定义的Istio 网关发布在集群中被创建出来，其中网关配置设置代理以充当负载平衡器，为入口公开端口80和443（https）。如下图所示。

名称	类型	创建时间	集群IP	内部端点	外部端点	操作
istio-customgateway	LoadBalancer	2019-01-14 16:52:26	172.19.12.75	istio-customgateway:80 TCP istio-customgateway:31144 TCP istio-customgateway:443 TCP istio-customgateway:30441 TCP	[redacted]1:80 [redacted]1:443	详情 | 更新 | 查看YAML | 删除

定义内部服务

本示例中的内部服务是基于nginx实现的，首先为 NGINX 服务器创建配置文件。以域名`myexample.com`的内部服务为例，定义请求根路径直接返回字样*`Welcome to myexample.com ! This is one custom Istio Ingress Gateway powered by cert-manager!`*及状态码200。

myexample-nginx.conf的具体内容如下：

```
events {
}

http {
  log_format main '$remote_addr - $remote_user [$time_local]  $status
 '
  '"$request" $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';
  access_log /var/log/nginx/access.log main;
  error_log  /var/log/nginx/error.log;

  server {
    listen 80;

    location / {
        return 200 'Welcome to myexample.com! This is one custom Istio
 Ingress Gateway powered by cert-manager!';
        add_header Content-Type text/plain;
    }
  }
```

```
}
```

1. 执行如下命令，创建 Kubernetes ConfigMap 存储 Nginx 服务器的配置。

```
kubectl create configmap myexample-nginx-configmap --from-file=nginx
.conf=./myexample-nginx.conf
```

2. 执行如下命令，设置命名空间`default`，启用sidecar自动注入。

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

> **说明:**
> 确保该sidecar自动注入的Label需要在IngressGateway创建之后再进行标注，以确保IngressGateway不会自动注入。或者不启用自动注入，通过手工注入完成，具体参见手工注入。

3. 创建并拷贝如下内容到*myexampleapp.yaml*文件中，并执行`kubectl apply -f myexampleapp.yaml`命令，创建域名myexample.com的内部服务。

```
apiVersion: v1
kind: Service
metadata:
  name: myexampleapp
  labels:
    app: myexampleapp
spec:
  ports:
  - port: 80
    protocol: TCP
  selector:
    app: myexampleapp
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myexampleapp
spec:
  selector:
    matchLabels:
      app: myexampleapp
  replicas: 1
  template:
    metadata:
      labels:
        app: myexampleapp
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config
          mountPath: /etc/nginx
          readOnly: true
      volumes:
      - name: nginx-config
```

```
        configMap:
          name: myexample-nginx-configmap

EOF
```

创建自定义网关配置对象

创建并拷贝如下内容到*istio-myexample-customingressgateway.yaml*文件中，并执行`kubectl apply -f istio-myexample-customingressgateway.yaml`命令，创建Istio自定义网关配置对象。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  annotations:
  name: istio-myexample-customingressgateway
  namespace: default
spec:
  selector:
    istio: ingressgateway
  servers:
  - hosts:
    - '*.myexample.com'
    port:
      name: http
      number: 80
      protocol: HTTP
    tls:
      httpsRedirect: true
  - hosts:
    - '*.myexample.com'
    port:
      name: https
      number: 443
      protocol: HTTPS
    tls:
      mode: SIMPLE
      privateKey: /etc/istio/ingressgateway-certs/tls.key
      serverCertificate: /etc/istio/ingressgateway-certs/tls.crt
EOF
```

创建VirtualService

创建并拷贝如下内容到*istio-myexample-customvirtualservice.yaml*文件中，并执行`kubectl apply -f istio-myexample-customvirtualservice.yaml`命令，创建链接到istio-myexample-customingressgateway 的 VirtualService。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: istio-myexample-customvirtualservice
spec:
  hosts:
  - "www.myexample.com"
  gateways:
  - istio-myexample-customingressgateway
  http:
```

```
  - route:
    - destination:
        host: myexampleapp
        port:
          number: 80
EOF
```

通过网关访问服务

1. 以域名myexample.com为例，执行`kubectl get svc -l istio=ingressgateway`命令，获取对应的自定义网关服务的公网IP地址。

```
NAME                    TYPE           CLUSTER-IP     EXTERNAL-IP
  PORT(S)                        AGE
istio-ingressgateway   LoadBalancer   172.19.XX.XX   106.14.XX.XX
80:31144/TCP,443:30441/TCP   11m
```

说明:

将环境变量 `INGRESS_HOST` 和 `SECURE_INGRESS_PORT` 这两个的取值替换成你实际环境的地址值，即步骤1中查到的 `EXTERNAL-IP` 的值。

```
INGRESS_HOST=106.14.XX.XX
SECURE_INGRESS_PORT=443
```

2. 执行如下命令，检查 istio-ingressgateway Pod 是否正确的加载了证书和私钥。

```
kubectl exec -it -n default $(kubectl -n default get pods -l istio=
ingressgateway -o jsonpath='{.items[0].metadata.name}') -- ls -al /
etc/istio/ingressgateway-certs
```

此时，tls.crt 和 tls.key 都应该保存在这个目录中。

3. 执行如下命令，检查 Ingress gateway 证书中的 Subject 字段的正确性。

```
kubectl exec -i -n default $(kubectl get pod -l istio=ingressgateway
 -n default -o jsonpath='{.items[0].metadata.name}')  -- cat /etc
/istio/ingressgateway-certs/tls.crt | openssl x509 -text -noout |
grep 'Subject:'
        Subject: O=MyExample CA, CN=myexample.com
```

4. 执行如下命令，检查 Ingress gateway 的代理能够正确访问证书。

```
kubectl exec -ti $(kubectl get po -l istio=ingressgateway -n default
 -o jsonpath={.items[0]..metadata.name}) -n default -- curl  127.0.0
.1:15000/certs
{
    "ca_cert": "",
    "cert_chain": "Certificate Path: /etc/istio/ingressgateway-certs
/tls.crt, Serial Number: c181438895a781c98759fb56b9cc1508, Days
until Expiration: 364"
}
```

此时，使用cert-manager部署自定义入口网关的所有步骤已完成。

预期结果

执行如下命令，通过 HTTPS 协议访问 myexample.com 服务。

```
curl -k -H Host:www.myexample.com --resolve www.myexample.com:443:106.
14.XX.XX  https://www.myexample.com
Welcome to myexample.com! This is one custom Istio Ingress Gateway
powered by cert-manager!
```

5.8 Istio CoreDNS

CoreDNS 及其Plugin扩展

CoreDNS 是一个 CNCF 下的孵化级项目，它的前身是 SkyDNS，主要目的是构建一个快速灵活的 DNS 服务器，让用户可以通过不同方式访问和使用 DNS 内的数据。基于 Caddy 服务器框架，CoreDNS 实现了一个插件链的架构，将大量逻辑抽象成插件 Plugin 的形式暴露给使用者，每个插件都执行 DNS 功能，例如 Kubernetes 的 DNS 服务发现、Prometheus 监控等。

除了插件化之外，CoreDNS 还有以下特性：

- 配置简单化：引入表达力更强的 DSL，即 Corefile 形式的配置文件（也是基于 Caddy 框架开发）。
- 一体化的解决方案：区别于 kube-dns，CoreDNS 编译出来就是一个单独的二进制可执行文件，内置了 Cache、Backend Storage、Health Check 等功能，无需第三方组件来辅助实现其他功能，从而使得部署更方便，内存管理更为安全。

Corefile 介绍

Corefile 是 CoreDNS 的配置文件（源于 Caddy 框架的配置文件 Caddyfile），定义了以下内容：

- 服务器以什么协议监听在哪个端口（可以同时定义多个 server 监听不同端口）
- 服务器负责哪个 zone 的 authoritative DNS 解析
- 服务器将加载哪些插件

一个典型的 Corefile 格式如下所示。

```
ZONE:[PORT] {
    [PLUGIN] ...
}
```

Corefile 的各参数含义如下：

参数	含义
ZONE	用于定义服务器负责的 zone

参数	含义
PORT	监听端口，可选项，默认为 53
PLUGIN	定义服务器所要加载的插件plugin，每个 plugin 可以有多个参数

Plugin工作机制

当 CoreDNS 启动后，它将根据配置文件启动不同 server ，每台 server 都拥有自己的插件链。当有 DNS 请求时，它将依次经历如下 3 步逻辑：

- 如果有当前请求的 server 有多个 zone，将采用贪心原则选择最匹配的 zone。
- 一旦找到匹配的 server，按照 plugin.cfg 定义的顺序执行插件链上的插件。
- 每个插件将判断当前请求是否应该处理，将有以下几种可能：
  - 请求被当前插件处理

    插件将生成对应的响应并回给客户端，此时请求结束，下一个插件将不会被调用，如 whoami 插件。
  - 请求不被当前插件处理

    直接调用下一个插件。如果最后一个插件执行错误，服务器返回 SERVFAIL 响应。
  - 请求被当前插件以 Fallthrough 形式处理

    如果请求在该插件处理过程中有可能将跳转至下一个插件，该过程称为 fallthrough，并以关键字 fallthrough 来决定是否允许此项操作，例如 host 插件，当查询域名未位于 `/etc/hosts`，则调用下一个插件。

    - 请求在处理过程被携带 Hint。
    - 请求被插件处理，并在其响应中添加了某些信息（hint）后继续交由下一个插件处理。这些额外的信息将组成对客户端的最终响应，如 metric 插件。

CoreDNS 与 Kubernetes

从 Kubernetes 1.11开始，CoreDNS 作为 Kubernetes 的 DNS 插件进入GA状态，Kubernetes 推荐使用 CoreDNS 作为集群内的 DNS 服务。在阿里云容器服务 Kubernetes 1.11.5 及更新版本中默认安装使用 CoreDNS 作为 DNS 服务。

执行`kubectl -n kube-system get configmap coredns -oyaml`命令可以查看 CoreDNS 配置信息。

```
apiVersion: v1
data:
  Corefile: |-
    .:53 {
```

```
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           upstream
           fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        proxy . /etc/resolv.conf
        cache 30
        reload
    }
kind: ConfigMap
metadata:
  creationTimestamp: "2018-12-28T07:28:34Z"
  name: coredns
  namespace: kube-system
  resourceVersion: "2453430"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: 2f3241d5-0a72-11e9-99f1-00163e105bdf
```

配置文件各参数的含义如下：

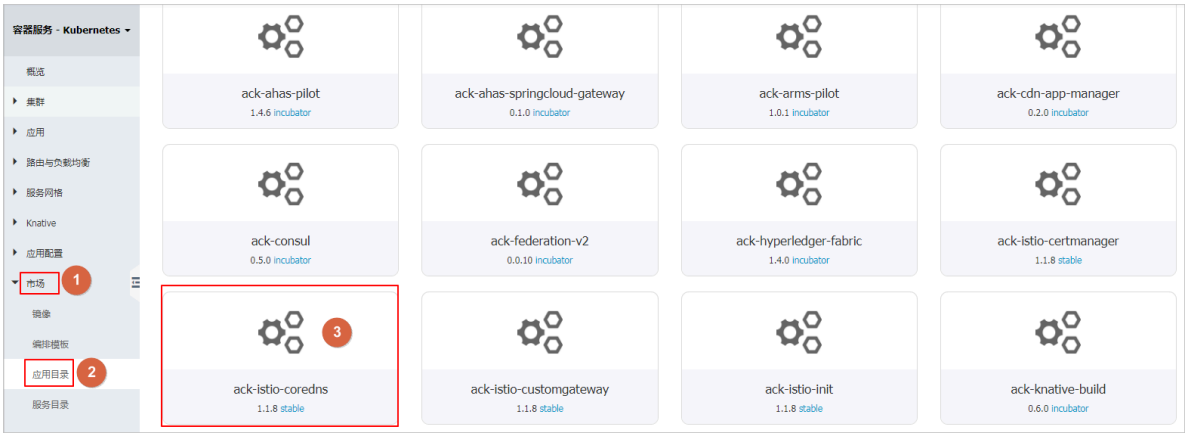
名称	含义
errors	错误会被记录到标准输出
health	健康状况，可以通过*http://localhost:8080/health*查看
kubernetes	根据服务的IP响应DNS查询请求，Cluster Domain默认为cluster.local
prometheus	可以通过*http://localhost:9153/metrics*获取prometheus格式的监控数据
proxy	本地无法解析后，向上级地址进行查询，默认使用宿主机的*/etc/resolv.conf*配置
cache	缓存时间

通过应用目录简便部署 Istio CoreDNS

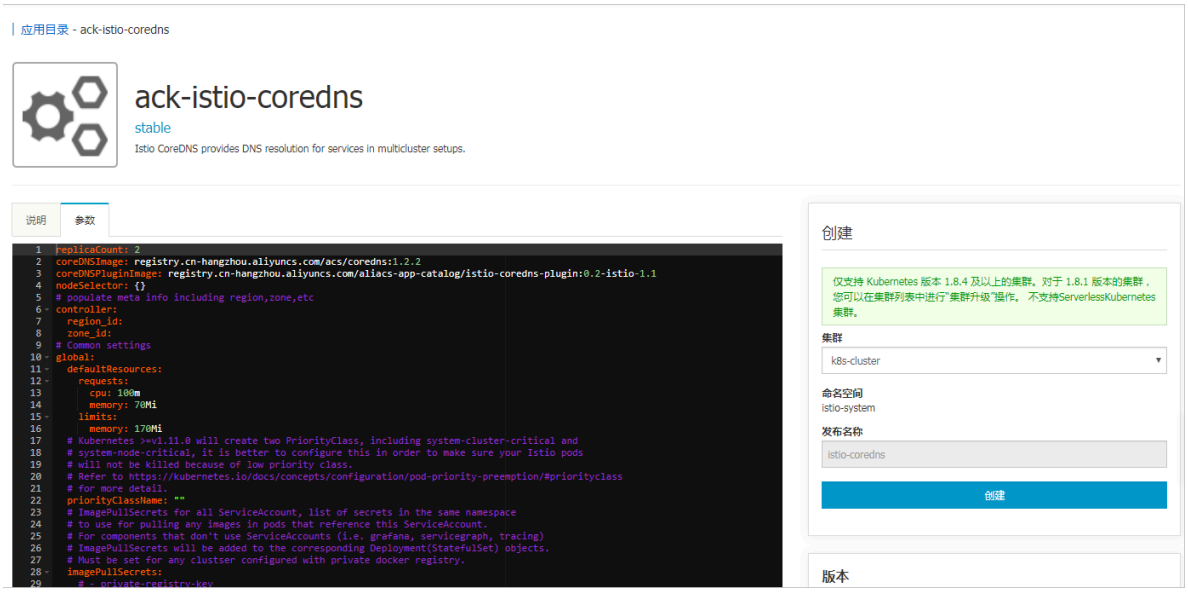
阿里云容器服务 Kubernetes 1.11.5目前已经上线，可以通过容器服务管理控制台非常方便地快速创建 Kubernetes 集群。具体过程可以参考创建Kubernetes集群。

1. 登录 容器服务管理控制台。

2. 选择市场 > 应用目录，在右侧选中ack-istio-coredns。



3. 在应用目录 - ack-istio-coredns中，单击参数，可以通过修改参数配置进行定制化，如下所示。

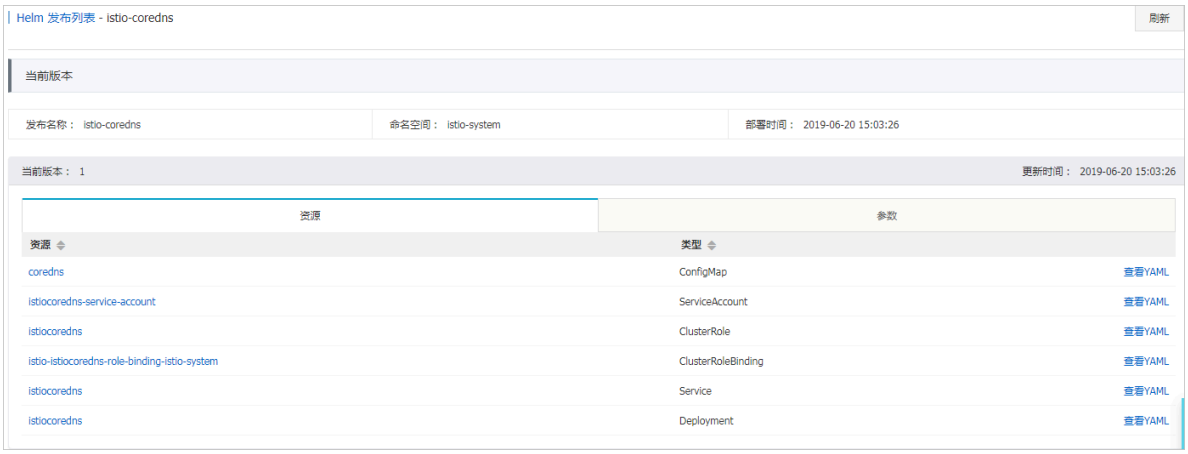


参数描述如下：

参数	含义	参数值	默认值
replicaCount	指定副本数	number	1
coreDNSImage	指定coredns镜像名称	valid image tag	coredns/coredns:1.2.2
coreDNSPluginImage	指定插件镜像名称	valid image name	registry.cn-hangzhou.aliyuncs.com/aliacs-app-catalog/istio-coredns-plugin:v1.0-istio-1.1

4. 修改完成后，在右侧的创建页面，在右侧选择对应的集群，同时可以看到命名空间已设定为`istio-system`，发布名称已设定为`istio-coredns`，单击创建。

   几秒钟后，在弹出Helm 发布列表 - istio-coredns页面，可以看到 Istio CoreDNS 在集群中被创建出来。



更改集群 CoreDNS 配置

1. 可以通过执行`kubectl get svc istiocoredns -n istio-system`命令，来获取`istiocoredns`服务的Cluster IP。

```
NAME            TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)
     AGE
istiocoredns    ClusterIP   172.19.XX.XX    <none>         53/UDP,53/
TCP    44m
```

   更新集群的coredns的配置项configmap， 将此`istiocoredns`服务作为 `*.global`域的上游DNS服务，在此配置项configmap中添加域 `*.global`，如下所示。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |-
    .:53 {
        errors
        health
        kubernetes cluster.local in-addr.arpa ip6.arpa {
           pods insecure
           upstream
           fallthrough in-addr.arpa ip6.arpa
        }
        prometheus :9153
        proxy . /etc/resolv.conf
        cache 30
        reload
    }
    global:53 {
         errors
```

```
        cache 30
        proxy . {replace this with the actual cluster IP of this
istiocoredns service}
```

2. 更改完成后，集群的 CoreDNS 容器会重新加载这些配置内容，可以通过如下命令行查看加载日志。

```
# kubectl get -n kube-system pod | grep coredns
coredns-8645f4b4c6-5frkg                                          1/1
  Running   0          20h
coredns-8645f4b4c6-lj59t                                          1/1
  Running   0          20h

# kubectl logs -f -n kube-system coredns-8645f4b4c6-5frkg
....
2019/01/08 05:06:47 [INFO] Reloading
2019/01/08 05:06:47 [INFO] plugin/reload: Running configuration MD5
 = 05514b3e44bcf4ea805c87cc6cd56c07
2019/01/08 05:06:47 [INFO] Reloading complete

# kubectl logs -f -n kube-system coredns-8645f4b4c6-lj59t
....
2019/01/08 05:06:31 [INFO] Reloading
2019/01/08 05:06:32 [INFO] plugin/reload: Running configuration MD5
 = 05514b3e44bcf4ea805c87cc6cd56c07
2019/01/08 05:06:32 [INFO] Reloading complete
```

创建 ServiceEntry 验证 DNS 解析

ServiceEntry 能够在 Istio 内部的服务注册表中加入额外的条目，从而让网格中自动发现的服务能够访问和路由到这些手工加入的服务。ServiceEntry 描述了服务的属性，即包括 DNS 名称、虚拟 IP、端口、协议以及端点等。这类服务可能是网格外的 API，或者是处于网格内部但却不存在于平台的服务注册表中的条目。例如，需要和 Kubernetes 服务沟通的一组基于虚拟机VM提供的服务。

本例中，使用了通配符定义 hosts，并指定了address，如下所示。

1. 通过 kubectl 连接 Kubernetes 集群，创建并拷贝以下内容到 *external-svc-serviceentry.yaml* 文件中，并执行`kubectl apply -f external-svc-serviceentry.yaml`命令，创建 ServiceEntry。

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: external-svc-serviceentry
spec:
  hosts:
  - '*.test.global'
  addresses:
  - 127.255.0.2
  ports:
  - number: 8080
    name: http1
    protocol: HTTP
  location: MESH_INTERNAL
```

```
  resolution: DNS
  endpoints:
  - address: 47.111.38.80
    ports:
      http1: 15443
```

2. 执行如下命令查看容器`istiocoredns`，可以看到上述 ServiceEntry 的域名映射关系已经被加载。

```
# kubectl get po -n istio-system | grep istiocoredns
istiocoredns-cdc56b67-ngtkr                 2/2     Running   0
     1h

# kubectl logs --tail 2 -n istio-system istiocoredns-cdc56b67-ngtkr
 -c istio-coredns-plugin
2019-01-08T05:27:22.897845Z    info    Have 1 service entries
2019-01-08T05:27:22.897888Z    info    adding DNS mapping: .test.
global.->[127.255.XX.XX]
```

3. 执行如下命令，使用镜像 tutum/dnsutils 创建测试容器。

```
# kubectl run dnsutils -it --image=tutum/dnsutils bash
```

4. 进入容器命令行后，执行 dig 查看相应的域名解析。

```
# dig +short 172.19.XX.XX A service1.test.global
127.255.XX.XX
# dig +short 172.19.XX.XX A service2.test.global
127.255.XX.XX
```

5.9 通过Istio从外部访问Bookinfo应用

本文详细讲述如何部署Bookinfo应用，并通过 Istio从外部访问该应用。

前提条件

1. 您已成功创建一个 Kubernetes 集群，参见创建Kubernetes集群，且集群版本为v1.12及以上。
2. 您已在该集群成功部署Istio，并且启用 Sidecar 自动注入，参见部署Istio。

操作步骤

部署Bookinfo应用

1. 通过CloudShell 连接 Kubernetes 集群。
2. 创建并拷贝内容到*book-info.yaml*文件中，并执行`kubectl apply -f book-info.yaml`命令，部署Bookinfo示例应用。

```
apiVersion: v1
kind: Service
metadata:
  name: details
  labels:
```

```
    app: details
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: details
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: details-v1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: details
        version: v1
    spec:
      containers:
      - name: details
        image: istio/examples-bookinfo-details-v1:1.8.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
##############################################################################
# Ratings service
##############################################################################
apiVersion: v1
kind: Service
metadata:
  name: ratings
  labels:
    app: ratings
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: ratings
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: ratings-v1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: ratings
        version: v1
    spec:
      containers:
      - name: ratings
        image: istio/examples-bookinfo-ratings-v1:1.8.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
##############################################################################
# Reviews service
```

```
##################################################################################
apiVersion: v1
kind: Service
metadata:
  name: reviews
  labels:
    app: reviews
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: reviews
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: reviews-v1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: reviews
        version: v1
    spec:
      containers:
      - name: reviews
        image: istio/examples-bookinfo-reviews-v1:1.8.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: reviews-v2
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: reviews
        version: v2
    spec:
      containers:
      - name: reviews
        image: istio/examples-bookinfo-reviews-v2:1.8.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: reviews-v3
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: reviews
        version: v3
    spec:
```

```
      containers:
      - name: reviews
        image: istio/examples-bookinfo-reviews-v3:1.8.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
##############################################################################
# Productpage services
##############################################################################
apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: productpage
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: productpage-v1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: productpage
        version: v1
    spec:
      containers:
      - name: productpage
        image: istio/examples-bookinfo-productpage-v1:1.8.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
```

3. 执行`kubectl get pod,svc`命令，列出所有的 pods 和 svc。

```
NAME                                    READY   STATUS    RESTARTS
AGE
pod/details-v1-5d88f495b7-cvxk5         2/2     Running   0
19h
pod/productpage-v1-774fd75c99-5l898     2/2     Running   0
19h
pod/ratings-v1-64664b6bcf-j7lzh         2/2     Running   0
19h
pod/reviews-v1-fd7c6fdf5-4px4g          2/2     Running   0
19h
pod/reviews-v2-56b67454cc-fmtl8         2/2     Running   0
19h
pod/reviews-v3-86878d875-6xztb          2/2     Running   0
19h

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT
(S)    AGE
```

```
service/details       ClusterIP   172.21.XX.XX   <none>        9080/
TCP   19h
service/kubernetes    ClusterIP   172.21.XX.XX      <none>
443/TCP    21h
service/productpage   ClusterIP   172.21.XX.XX    <none>        9080
/TCP   19h
service/ratings       ClusterIP   172.21.XX.XX    <none>        9080
/TCP   19h
service/reviews       ClusterIP   172.21.XX.XX    <none>        9080
/TCP   19h
```

可以看到，每个pod里都有两个容器。

> **说明:**
>
> 部署Istio时，默认开启启用 Sidecar 自动注入，Istio自动为在每个pod中注入了一个Sidecar，使得容器比设置的数量多1个，因此上述每个pod中有两个容器。

部署 Ingress Gateway

1. 创建并拷贝内容到*bookinfo-gateway.yaml*文件中，并执行`$ kubectl apply -f bookinfo-gateway.yaml`命令，部署Ingress Gateway示例应用。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - "*"
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "*"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
```

```
        host: productpage
        port:
          number: 9080
```

2. 执行如下命令，获取外部负载均衡器的IP和端口信息作为 INGRESS_HOST 和 INGRESS_PORT 。

```
export INGRESS_HOST=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
export INGRESS_PORT=$(kubectl -n istio-system get service istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http")].port}')
```

3. 执行如下命令，如果返回200，则说明已经配置正确。

```
curl -o /dev/null -s -w "%{http_code}\n" http://${INGRESS_HOST}:${INGRESS_PORT}/productpage
```

此时，你可以根据在浏览器访问对应的地址。

5.10 通过应用目录快速启动自定义的Gateway

本文讲述如何通过应用目录，快速启动自定义的Gateway。

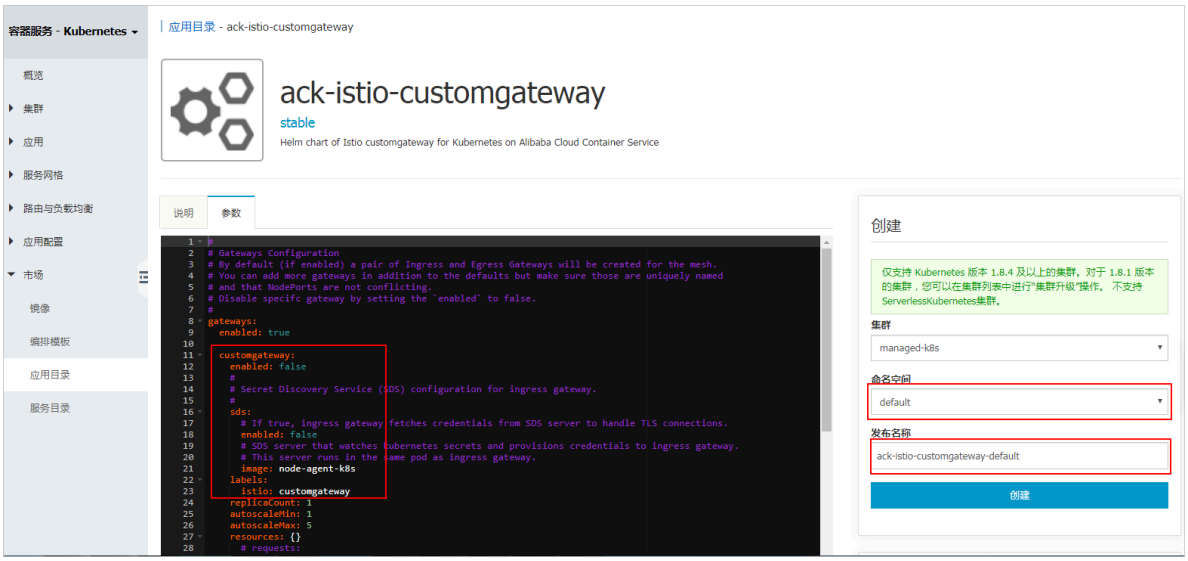
前提条件

- 请确保目标集群已部署Istio。
- 每个命名空间下，有且只能部署一个Gateway。
- 请确保自定义的 Gateway 的发布名称在同一个命名空间下唯一。

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市场 > 应用目录，在右侧选中ack-istio-customgateway。

3. 在应用目录 - ack-istio-customgateway中，单击参数，可以通过修改参数配置进行定制化，如下所示。



说明:

参数区域中有两个`customgateway`，您可以自定义两个customgateway，为了管理方便，建议`customgateway`和Istio的值`customgateway`保持一致。并且在同一个namespace下，名称要唯一性。

4. 在右侧的创建页面，在右侧选择对应的集群，同时可以看到命名空间已设定为`default`，发布名称已设定为`ack-istio-customgateway-default`，然后单击创建。
5. 几秒钟后，在弹出Helm 发布列表 - ack-istio-customgateway-default页面，Istio 应用Gateway 发布在集群中被创建出来。

5.11 使用HTTPS来访问Ingress Gateway

本文详细讲述如何使用HTTPS来访问Ingress Gateway。

环境准备

1. 通过CloudShell 连接 Kubernetes 集群。
2. 执行如下命令`curl --version | grep LibreSSL`命令，确认测试环境。

   如果输出结果包含LibreSSL的版本信息，表示使用的是用LibreSSL库编译的curl，即，表示环境已准备就绪。

```
curl 7.54.0 (x86_64-apple-darwin18.0) libcurl/7.54.0 LibreSSL/2.6.4
zlib/1.2.11 nghttp2/1.24.1
```

3. 部署一个可用的Bookinfo应用，请参考部署Bookinfo应用。

4. 执行如下命令，设置访问入口。

```
export INGRESS_HOST=$(kubectl -n istio-system get service istio-
ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
export INGRESS_PORT=$(kubectl -n istio-system get service istio-
ingressgateway -o jsonpath='{.spec.ports[?(@.name=="http")].port}')
export SECURE_INGRESS_PORT=$(kubectl -n istio-system get service
istio-ingressgateway -o jsonpath='{.spec.ports[?(@.name=="https")].
port}')
```

5. 执行如下命令，清理旧的 Gateway 和 VirtualService。

```
kubectl delete gateway --all
kubectl delete virtualservice --all
```

生成证书

通过CloudShell执行如下命令，使用https://github.com/nicholasjackson/mtls-go-example来生成需要的证书。

```
1. git clone https://github.com/nicholasjackson/mtls-go-example
2. pushd mtls-go-example
3. ./generate.sh bookinfo.com  &lt;password&gt;
4. mkdir ~+1/bookinfo.com && mv 1_root 2_intermediate 3_application
4_client ~+1/bookinfo.com
5. popd
```

配置TLS Ingress Gateway

1. 执行如下命令，创建secret来保存证书的内容。

```
kubectl create -n istio-system secret tls istio-ingressgateway-certs
 --key bookinfo.com/3_application/private/bookinfo.com.key.pem --
cert bookinfo.com/3_application/certs/bookinfo.com.cert.pem
```

2. 创建并拷贝内容到*book-info-gateway.yaml*文件中，并执行`kubectl apply -f book-info-gateway.yaml`命令，定义一个Gateway对象bookinfo-gateway。

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: bookinfo-gateway
spec:
  selector:
    istio: ingressgateway # use istio default ingress gateway
  servers:
  - port:
      number: 443
      name: https
      protocol: HTTPS
    tls:
      mode: SIMPLE
      serverCertificate: /etc/istio/ingressgateway-certs/tls.crt
      privateKey: /etc/istio/ingressgateway-certs/tls.key
    hosts:
```

```
    - "bookinfo.com"
```

3. 创建并拷贝内容到*book-info-vs.yaml*文件中，并执行`kubectl apply -f book-info-vs.yaml`命令，配置一个关联此Gateway的路由规则。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: bookinfo
spec:
  hosts:
  - "bookinfo.com"
  gateways:
  - bookinfo-gateway
  http:
  - match:
    - uri:
        exact: /productpage
    - uri:
        exact: /login
    - uri:
        exact: /logout
    - uri:
        prefix: /api/v1/products
    route:
    - destination:
        host: productpage
        port:
        number: 9080
```

4. 执行如下命令，访问服务。

```
curl -v -HHost:bookinfo.com --resolve bookinfo.com:$SECURE_ING
RESS_PORT:$INGRESS_HOST --cacert bookinfo.com/2_intermediate/certs/
ca-chain.cert.pem https://bookinfo.com:$SECURE_INGRESS_PORT/status/
418
```

> **说明:**
>
> 如果要在浏览器里访问服务，请先配置好DNS或者添加hosts条目。

5.12 基于istio的VirtualService和Destination完成蓝绿和灰度发布

本文详细讲述如何通过istio的VirtualService和Destination配合来完成蓝绿和灰度发布。

环境准备

- 部署 VirtualService 和Destination

1. 通过CloudShell 连接 Kubernetes 集群。
2. 创建并拷贝如下内容到*virtual-service.yaml*文件中，并执行`kubectl apply -f virtual-service.yaml`命令，部署 VirtualService。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
```

```
metadata:
  name: productpage
spec:
  hosts:
  - productpage
  http:
  - route:
    - destination:
        host: productpage
        subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: ratings
spec:
  hosts:
  - ratings
  http:
  - route:
    - destination:
        host: ratings
        subset: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: details
spec:
  hosts:
  - details
  http:
  - route:
    - destination:
        host: details
        subset: v1
```

3. **创建并拷贝如下内容到*destination.yaml*文件中，并执行`kubectl apply -f destination.yaml`命令，部署 Destination。**

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: productpage
spec:
  host: productpage
  subsets:
  - name: v1
    labels:
```

```
      version: v1
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: reviews
spec:
  host: reviews
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  - name: v3
    labels:
      version: v3
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: ratings
spec:
  host: ratings
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
  - name: v2-mysql
    labels:
      version: v2-mysql
  - name: v2-mysql-vm
    labels:
      version: v2-mysql-vm
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: details
spec:
  host: details
  subsets:
  - name: v1
    labels:
      version: v1
  - name: v2
    labels:
      version: v2
```

- **设置地址转发**

```
kubectl port-forward -n istio-system svc/istio-ingressgateway 3000:
80
```

此时，浏览器访问*localhost:3000/productpage*，可以看到book reviews一栏只有留言，没有评分，productpage使用的是v1版本的reviews服务。

蓝绿发布v2版本

经过前面的部署，reviews的v2版本已经在运行，现在要做的只是用蓝绿部署的方式，让v2版本上线即可。

本例中，部署完成后刷新页面，可以看到book reviews已经有了评分，而且评分的星星为黑色。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
  - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v2
```

按比例灰度发布v3版本

您可以让v2，v3版本同时在线，且两个版本各处理50%的流量。

本例中，部署完成后，多次刷新页面，可以看到book reviews一栏v2，v3版本随机出现，v3版本的评分中实心星星为红色。

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: reviews
spec:
  hosts:
    - reviews
  http:
  - route:
    - destination:
        host: reviews
        subset: v2
      weight: 50
    - destination:
        host: reviews
        subset: v3
      weight: 50
```

按http header灰度发布v3版本

只按照流量来简单的灰度还是有很多场景不能满足，您还可以按照用户来灰度，不同的用户访问不同的页面。

本例中，部署完成后，多次刷新页面，评分区的星星始终为黑色。

```
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: v3
  - route:
    - destination:
        host: reviews
        subset: v2
```

此时，您可以单击右上角Sign in，以用户名jason登录（不需要输入密码），您可以看到评分区的星星已经变成红色。

说明:

登录并访问后端服务时，会带上`end-user=XXX`的http header，使用jason登录后，就匹配到了上面yaml的规则，使得流量被引导到了v3版本的reviews。

6 运维

6.1 通过集群检查定位集群问题

本文介绍如何通过容器服务提供的集群检查功能快速定位Kubernetes集群常见问题。

前提条件

- 已成功创建Kubernetes集群，参见创建Kubernetes集群。
- Kubernetes集群处于正常运行中状态。（在集群列表中确认集群状态为运行中）。

操作步骤

1. 登录容器服务管理控制台。
2. 选择集群 > 集群，在集群列表页面，单击右侧的更多 > 集群检查，进入概览页面。

3. 选择测评 > 一键检查，或单击工具箱区域的一键检查，进入一键检查页面。

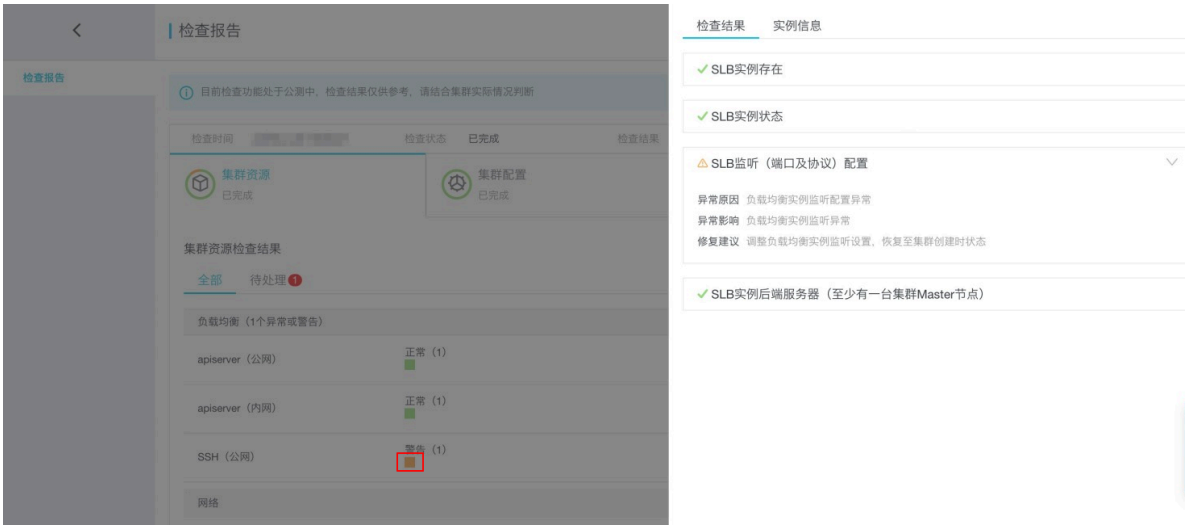
- 集群信息：呈现集群的名称、类型、运行状态、以及最近一次检查的结果。
- 一键检查：按照时间由近及远的顺序，呈现前5次检查时间及检查结果（如果集群首次执行检查任务，该区域呈现的是执行一键检查，您也可以单击该按钮进入一键检查页面）。
- 工具箱：提供的集群检查功能，当前一键检查功能已上线，后续将为您提供节点检查等更多集群检查功能。

4. 单击执行一键检查，在弹出的一键检查页面，确认并勾选注意事项，单击执行检查，启动集群检查任务。

5. 您可以看到集群检查的进度，单击查看详情，在检查报告页面，查看集群资源和集群配置的状态。

6. 已经完成的检查任务，可以根据检查报告提示进行相应处理：

- 正常：无需处理。
- 警告：需要确认，如果会引起集群异常的情况需要处理。
- 异常：尽快处理，避免集群无法正常工作。



针对单项检查结果，可以单击状态颜色框，查看被检查对象信息及具体检查结果，检查结果包含检查内容、可能影响以及推荐的修复方案，您可以参考影响及修复方案酌情处理。

6.2 Master节点的资源升配

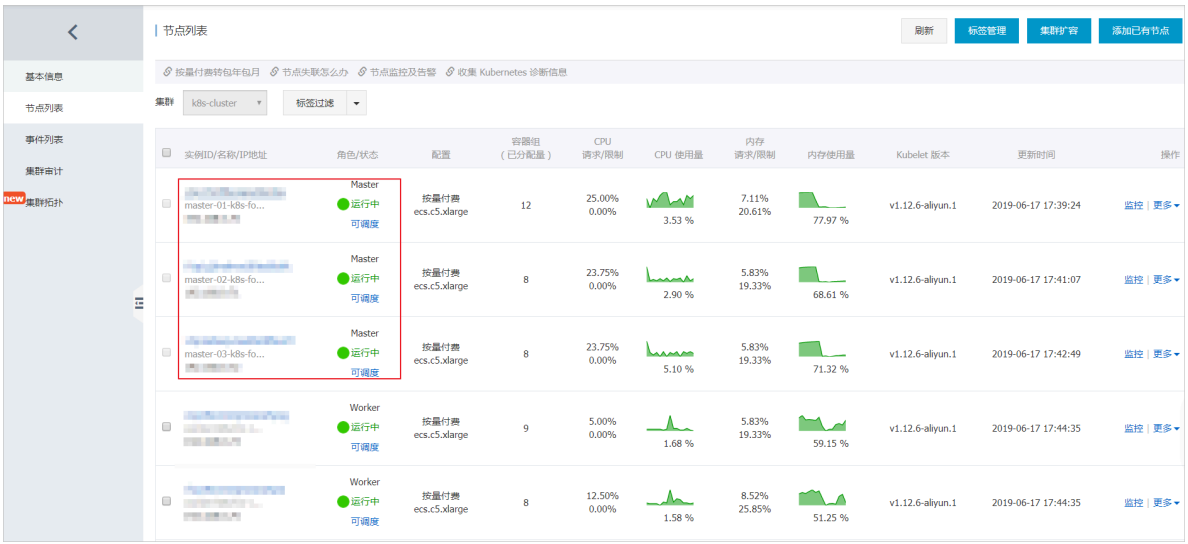
Kubernetes 专有版集群的Master节点由用户自己管控，随着集群规模的不断扩大，用户可能会遇到Master节点的配置不足的情况，本文为您介绍如何一步步升级Master的配置。

前提条件

您已经成功创建Kubernetes集群，参见创建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，选择集群 > 节点，进入节点列表。



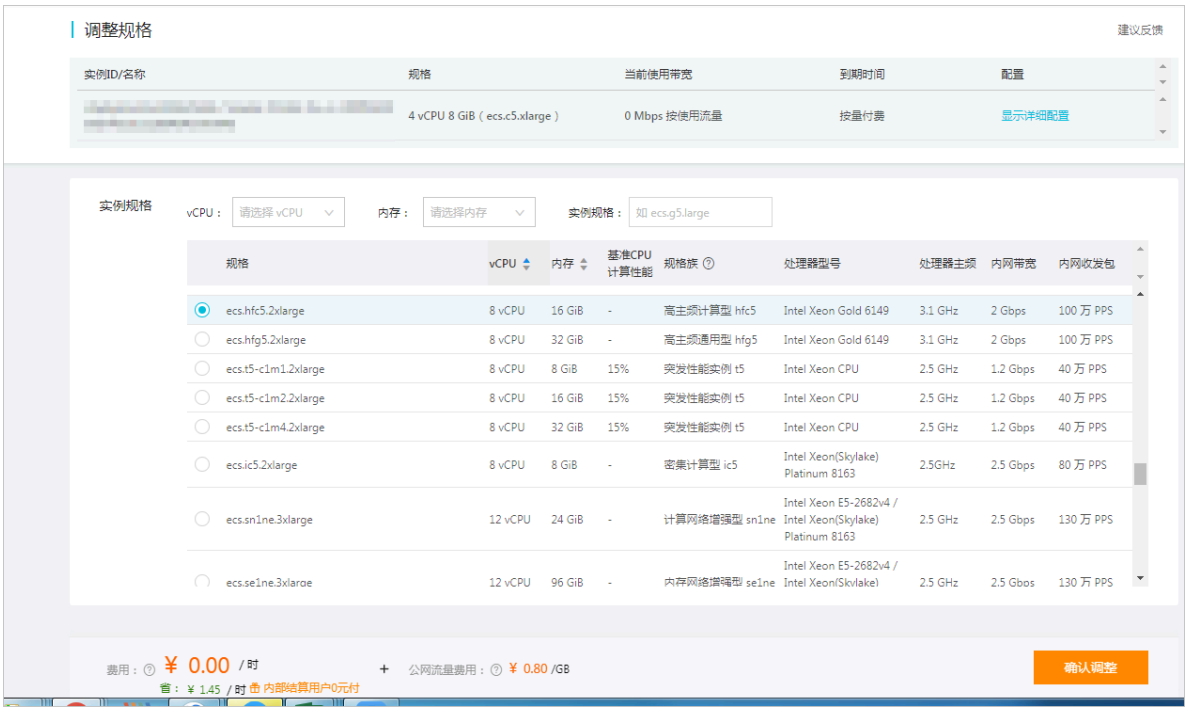
3. **找到集群的三个master节点，单击master节点的实例ID，进入实例详情页面（本文档以master-01为例）。**

   此时，您可以看到实例规格等信息。

4. 在实例的配置信息区域，单击更改实例规格。

如果更改实例规格为灰显，说明实例正在运行，您需要单击界面右上角的停止，停止运行该实例，再进行下一步。

5. 在弹出的调整规格页面，选择合适的规格，单击确认调整。节点规格请参见选择Master节点规格。

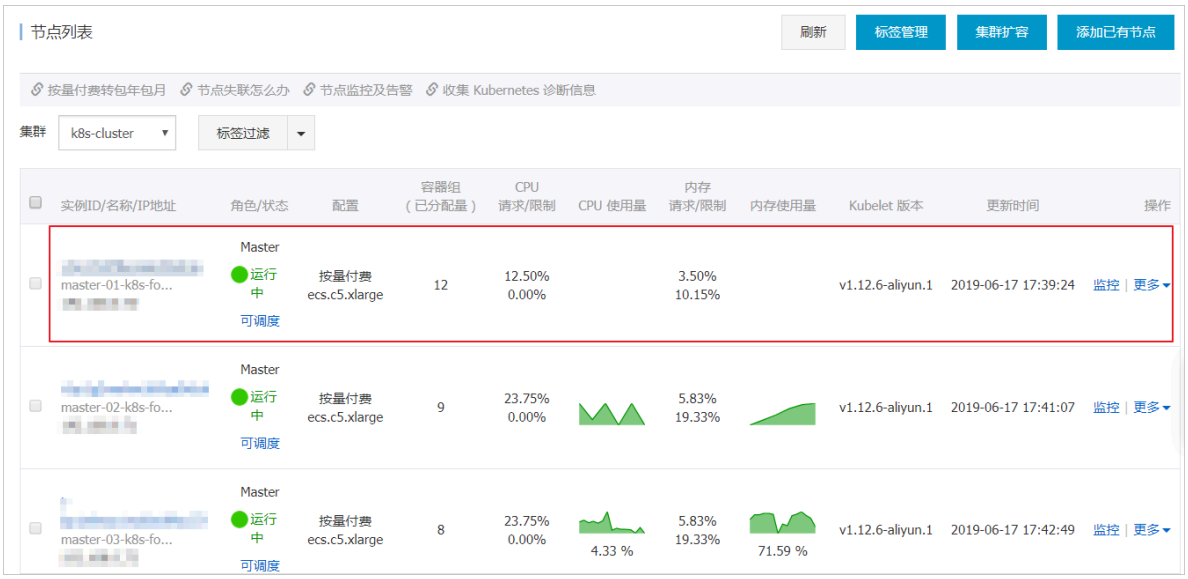


6. 在弹出的调整规格成功页面，单击确认。

此时自动跳转到实例列表页面。

可以看到master-01节点的规格已经调整成功。

7. 勾选目标实例，单击启动，等待升配的master-01节点自动加入集群且状态变成运行中，如图所示。



8. 重复步骤4-7，直至所有Master节点均已升级到期望的配置。

> **说明:**
> 本例仅介绍了按量付费类型的Master节点的升配方式，关于更多类型升配方式，请参见升降配方式汇总。

6.3 Worker节点的资源升配

随着Kubernetes集群负载的增加，用户常常面临资源不足的情况，此时您可以对集群的基础资源做一些扩展。例如，在集群规模达到10台以上的规模后，您可以优先考虑升级Worker节点的配置，这样既可以提高资源的利用率，也可以降低集群运维的复杂度。本文为您介绍如何升级Worker节点的配置。

前提条件

您已经成功创建Kubernetes集群，参见创建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，选择集群 > 集群，进入集群列表页面。
   - 如果您想通过购买新节点的方法升级配置，请参见扩容集群，并按照如下步骤操作。
     a. 在目标集群右侧单击操作列集群扩容，进入集群扩容页面。
     b. 设置需要扩容的Worker节点的数量。
     c. 设置新增的Worker节点配置。
     d. 设置新增的Worker节点的登录方式。
     e. 设置RDS白名单。
     f. 为新增节点绑定标签。
     g. 配置完成后，单击提交。
   - 如果您想通过添加已有节点的方法升级配置，请按照如下步骤操作。
     a. 在目标集群右侧单击操作列更多 > 添加已有节点。

        进入添加节点页面，您可以选择自动添加或手动添加的方式，添加现有云服务器实例。请参见添加已有节点。本例以自动添加为例进行升配。
     b. 添加方式选择自动添加，在已有云服务器的列表中，选择所需的ECS云服务器，然后单击下一步。
     c. 填写实例信息，设置CUP Policy、登录密码和标签，然后单击下一步。
     d. 在弹出的对话框中，单击确定，选择的 ECS 云服务器会自动添加到该集群中。
3. 选择集群 > 节点，待新增的节点的状态为运行中时，此时Worker节点升配完成。

后续步骤

如果您想要移除节点，您可以在节点列表，选择目标集群，在目标节点右侧操作列单击更多 > 移除，在弹出的对话框中单击确定。

说明:

- **为了确保升级过程中应用的稳定，建议您先加入新节点，再移除多余节点。**
- **移除的过程中，为了让应用更稳定，可以配置pdb。**

7 Serverless

7.1 使用GPU容器实例

本文主要为您介绍如何使用GPU容器实例，这里以使用 Tensorflow 进行图片识别为例进行演示。该功能适用于Serverless kubernetes 集群和在 Kubernetes 集群创建的虚拟节点。

背景信息

容器服务ACK Serverless（Serverless Kubernetes）基于ECI（弹性容器实例）正式推出GPU容器实例支持，让用户以 serverless 的方式快速运行AI计算任务，极大降低AI平台运维的负担，显著提升整体计算效率。

AI计算离不开GPU已经是行业共识，然而从零开始搭建GPU集群环境是件相对复杂的任务，包括GPU规格购买、机器准备、驱动安装、容器环境安装等。GPU资源的serverless交付方式，充分的展现了serverless的核心优势，其向用户提供标准化而且“开箱即用”的资源供给能力，用户无需购买机器也无需登录到节点安装GPU驱动，极大降低了AI平台的部署复杂度，让客户关注在AI模型和应用本身而非基础设施的搭建和维护，让使用GPU/CPU资源就如同打开水龙头一样简单方便，同时按需计费的方式让客户按照计算任务进行消费， 避免包年包月带来的高成本和资源浪费。

在容器服容器服务ACK Serverless中创建挂载GPU的Pod，通过Annotation指定所需GPU的类型，同时在resource.limits中指定GPU的个数即可（也可指定instance-type）。每个pod独占GPU，GPU实例的收费与ECS GPU类型收费一致，不产生额外费用。目前阿里云ECI提供的GPU规格及类型请参见ECI GPU使用示例

> **说明:**
> 目前创建挂载 vGPU 的 Pod 暂不支持该功能。

前提条件

您已经创建一个Serverless Kubernetes集群或已在Kubernetes 集群创建一个虚拟节点。

操作步骤

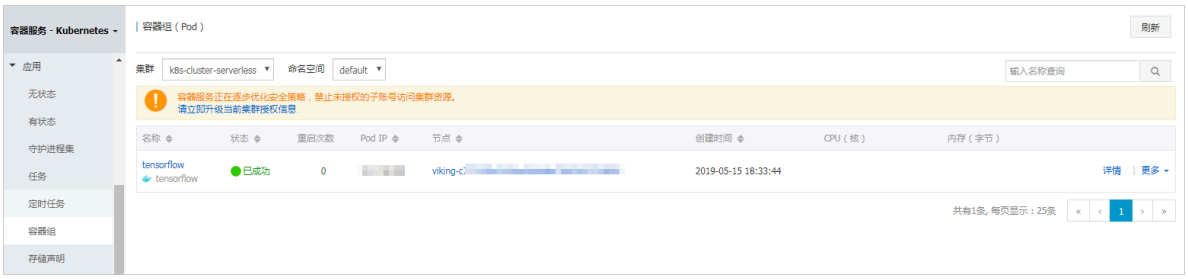
下面以 Serverless kubernetes 集群中在使用Tensorflow进行图片识别。

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 > 无状态，单击右上角的使用模板创建。
3. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

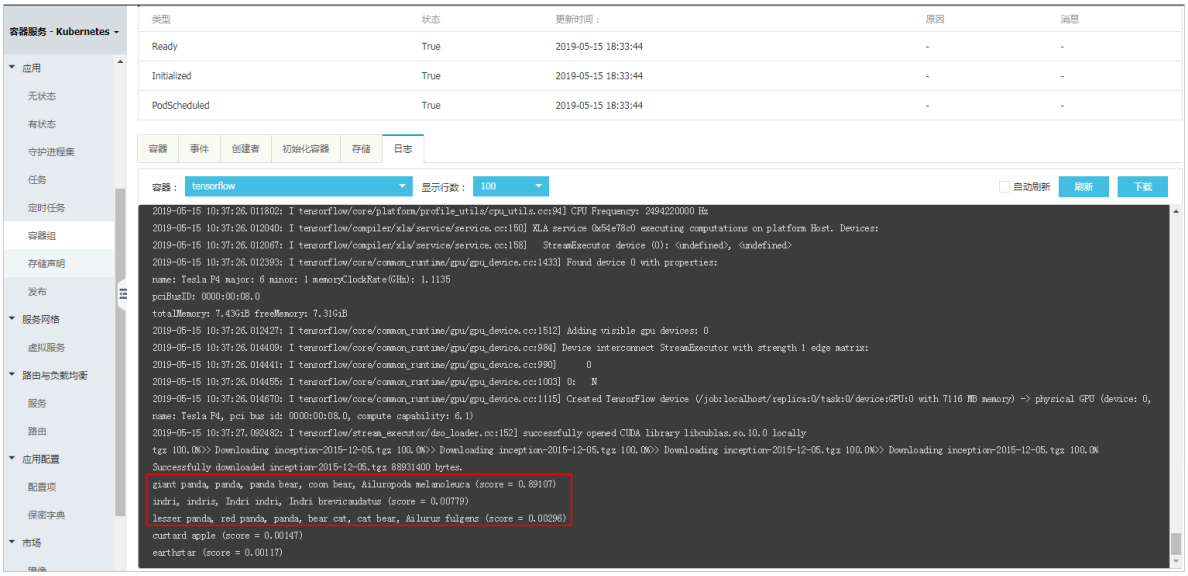
   您可以使用如下 yaml 示例模板创建Pod（本例中，Pod中指定GPU类型为P4，GPU个数为1）。

```
apiVersion: v1
kind: Pod
metadata:
  name: tensorflow
  annotations:
    k8s.aliyun.com/eci-gpu-type : "P4"
spec:
  containers:
  - image: registry-vpc.cn-hangzhou.aliyuncs.com/ack-serverless/
tensorflow
    name: tensorflow
    command:
    - "sh"
    - "-c"
    - "python models/tutorials/image/imagenet/classify_image.py"
    resources:
      limits:
        nvidia.com/gpu: "1"
  restartPolicy: OnFailure
```

4. 几分钟后，单击左侧导航栏中的应用 > 容器组，当出现如下内容时，表示Pod创建成功。



5. 单击目标Pod，在容器组 - tensorflow中的日志页签下，看到如下红框内容时，表示图片识别成功。



如果您想要在容器服务ACK的虚拟节点中进行使用该功能，请参考虚拟节点需要把Pod指定调度到虚拟节点上，或把Pod创建在有*virtual-node-affinity-injection=enabled label*的namespace中，然后使用如下示例文件替换步骤4的yaml文件即可。示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: tensorflow
  annotations:
    k8s.aliyun.com/eci-gpu-type : "P4"
spec:
  containers:
  - image: registry-vpc.cn-hangzhou.aliyuncs.com/ack-serverless/
tensorflow
    name: tensorflow
    command:
    - "sh"
    - "-c"
    - "python models/tutorials/image/imagenet/classify_image.py"
    resources:
      limits:
        nvidia.com/gpu: "1"
  restartPolicy: OnFailure
  nodeName: virtual-kubelet
```

说明:

基于虚拟节点的方式可以更灵活的支持多种深度学习框架，例如 Kubeflow、arena或其他自定义CRD。

7.2 如何给 Pod 挂载弹性公网IP

本文主要为您介绍如何在 serverless kubernetes 或虚拟节点中给Pod挂载 EIP。

背景信息

阿里云 Serverless Kubernetes 服务/虚拟节点推出 Pod 挂载弹性公网IP功能，此功能使某些Serverless容器应用的部署和服务访问变得更加简单和便利。

- 无需创建 VPC NAT网关即可让单个 Pod 访问公网
- 无需创建 Service 也可让单个 Pod 暴露公网服务
- 可以更加灵活而且动态的绑定 Pod 和 EIP

前提条件

- 您已经创建一个Serverless Kubernetes集群或已在Kubernetes 集群创建一个虚拟节点。
- 请确保该集群的安全组已开放相关端口（本示例中需要开放80端口）。

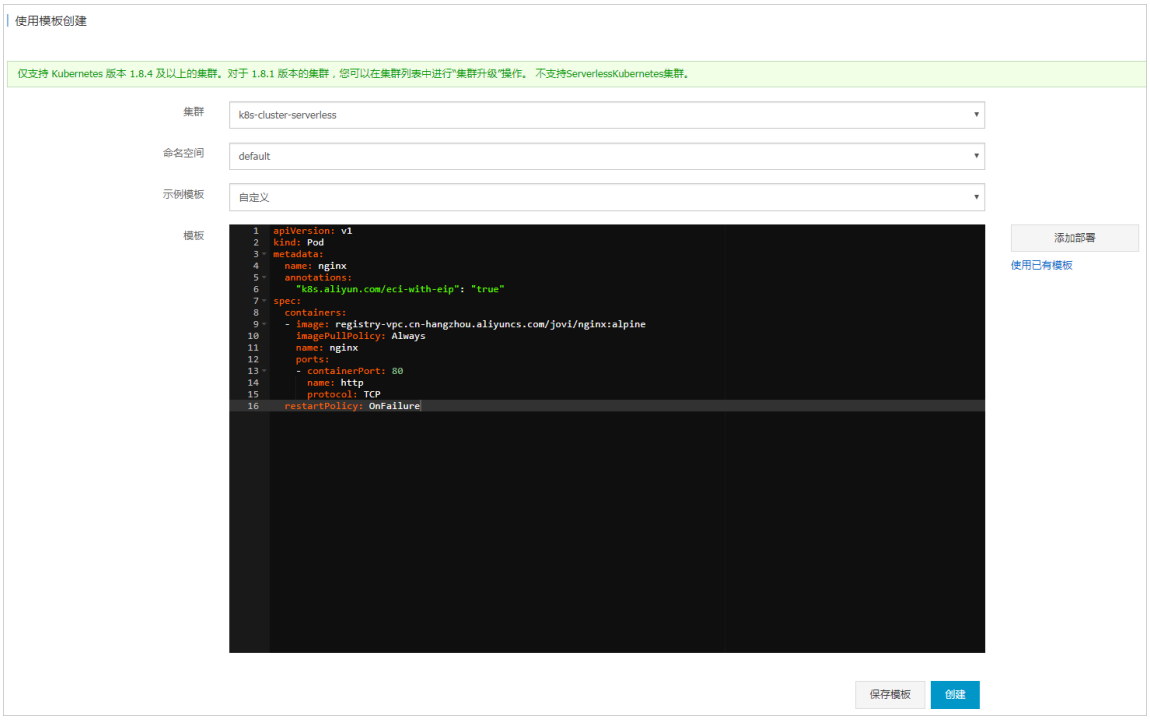
操作步骤

您可以通过以下两种方法，给Pod挂载弹性公网IP。

- 自动分配弹性公网 EIP

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 > 无状态，进入无状态（Deployment）页面。
3. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   您可以使用如下 yaml 示例模板创建 Pod。本例中，通过指定`k8s.aliyun.com/eci-with-eip`为 `true`，Serverless Kubernetes服务/虚拟节点会自动为此Pod分配一个EIP，并且绑定到 Pod上。

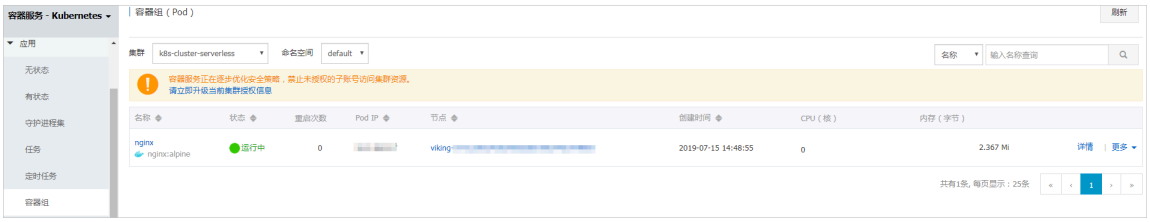



```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  annotations:
    "k8s.aliyun.com/eci-with-eip": "true"
spec:
  containers:
  - image: registry-vpc.cn-hangzhou.aliyuncs.com/jovi/nginx:alpine
    imagePullPolicy: Always
    name: nginx
    ports:
    - containerPort: 80
      name: http
      protocol: TCP
  restartPolicy: OnFailure
```

说明:

- 您也可以通过Annotation `k8s.aliyun.com/eip-bandwidth`指定EIP的带宽，默认值为5，单位为M。
- 如果您创建的是 Deployment，那么 Deployment 中的每一个 Pod 都将会被挂载不同的 EIP，请谨慎使用此操作。

4. 单击左侧导航栏中的应用 > 容器组，查看容器的状态。

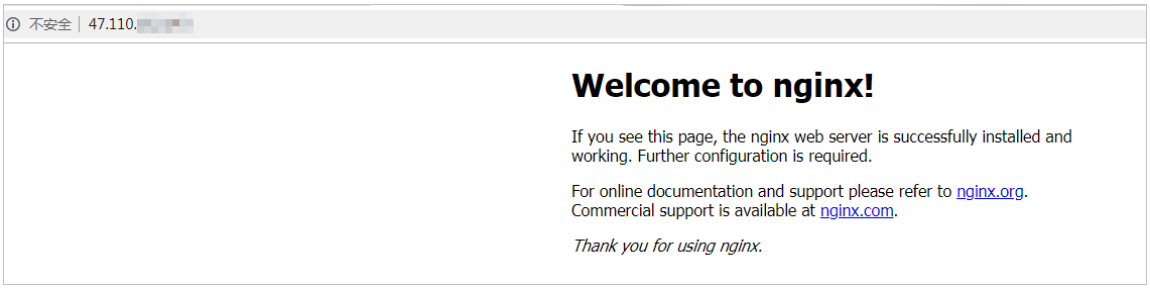


5. 在目标容器组右侧单击更多 > 编辑，弹出编辑 YAML文件。

> **说明：**
> YAML文件中`k8s.aliyun.com/allocated-eipAddress: 47.110.XX.XX`的IP地址即为EIP的公网访问地址。

6. 在浏览器中输入`http://ip##`，您可访问 nginx 欢迎页。

   此处的`http://ip##`为YAML文件中`k8s.aliyun.com/allocated-eipAddress: 47.110.XX.XX`的IP地址。



> **说明：**
> 此方式中 EIP 为动态分配，其生命周期与Pod相同，当删除Pod时，动态分配的EIP也会一并删除。

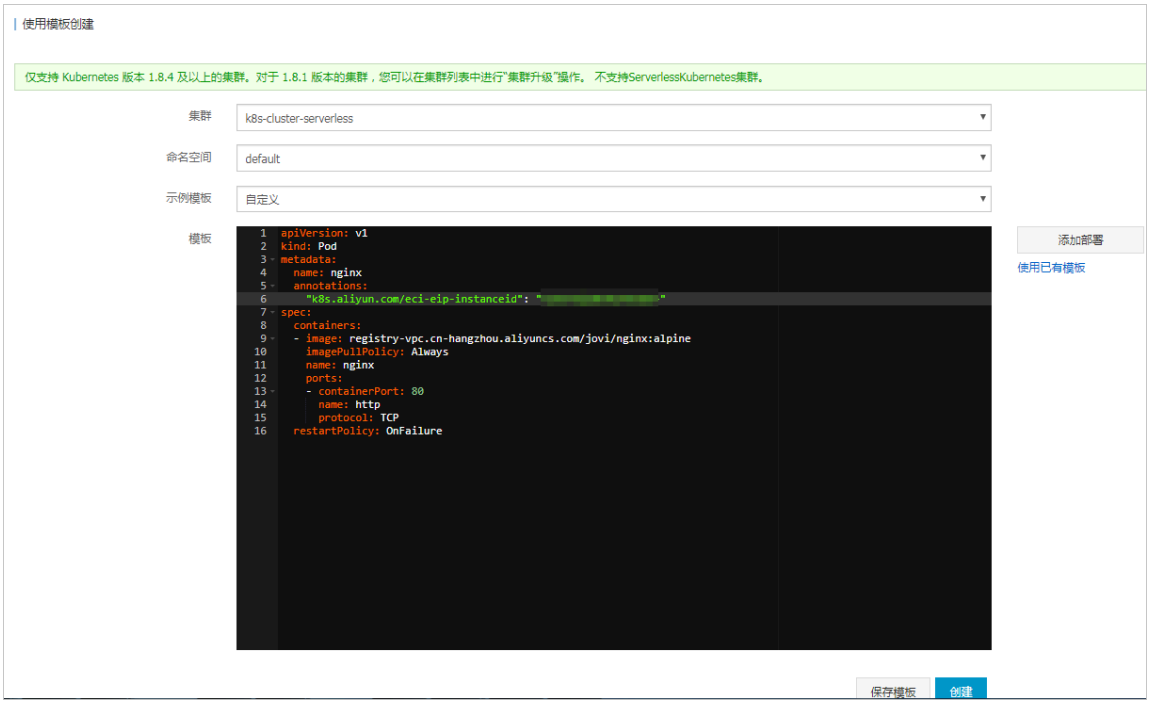
· **指定弹性公网 IP实例 ID**

1. 登录 VPC 管理控制台，购买弹性公网 IP。请参考申请EIP。

> **说明:**
>
> 申请的 EIP 和集群必须在同一区域。

2. 登录容器服务管理控制台。
3. 在 Kubernetes 菜单下，单击左侧导航栏中的应用 > 无状态，进入无状态（Deployment）页面。
4. 选择所需的集群和命名空间，选择样例模板或自定义，然后单击创建。

   您可以使用如下 yaml 示例模板创建 Pod。本例中，通过指定Pod的Annonation `k8s.aliyun.com/eci-eip-instanceid`为EIP实例ID。



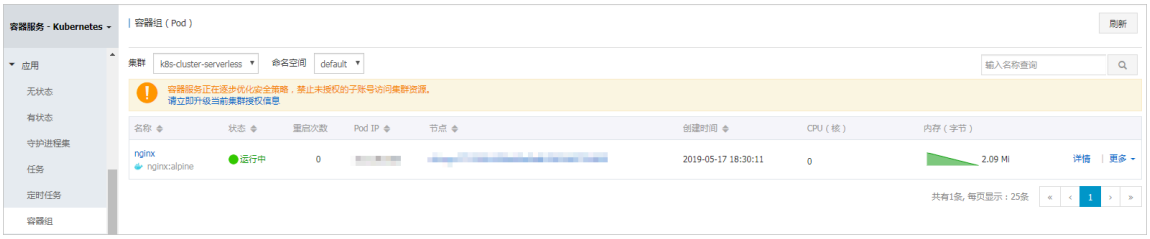

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  annotations:
    "k8s.aliyun.com/eci-eip-instanceid": "<youreipInstanceId>"
spec:
  containers:
```

```
  - image: registry-vpc.cn-hangzhou.aliyuncs.com/jovi/nginx:alpine
    imagePullPolicy: Always
    name: nginx
    ports:
    - containerPort: 80
      name: http
      protocol: TCP
  restartPolicy: OnFailure
```

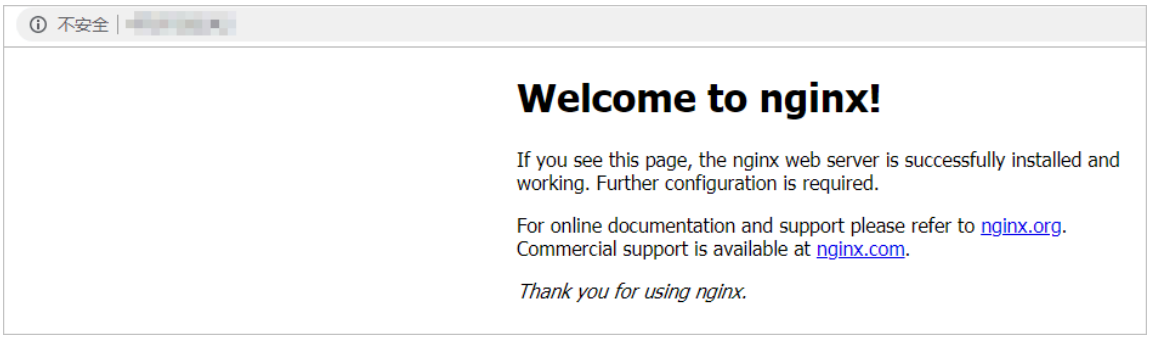
说明:

`<youreipInstanceId>`需要替换成步骤1中获取的EIP实例ID。

5. 单击左侧导航栏中的应用 > 容器组，查看容器的状态。



6. 在浏览器中输入*http://ip##*，您可访问 nginx 欢迎页。



说明:

此处的*http://ip##*为步骤1中申请的EIP的IP地址。

8 弹性伸缩

8.1 使用虚拟节点运行 Job 任务

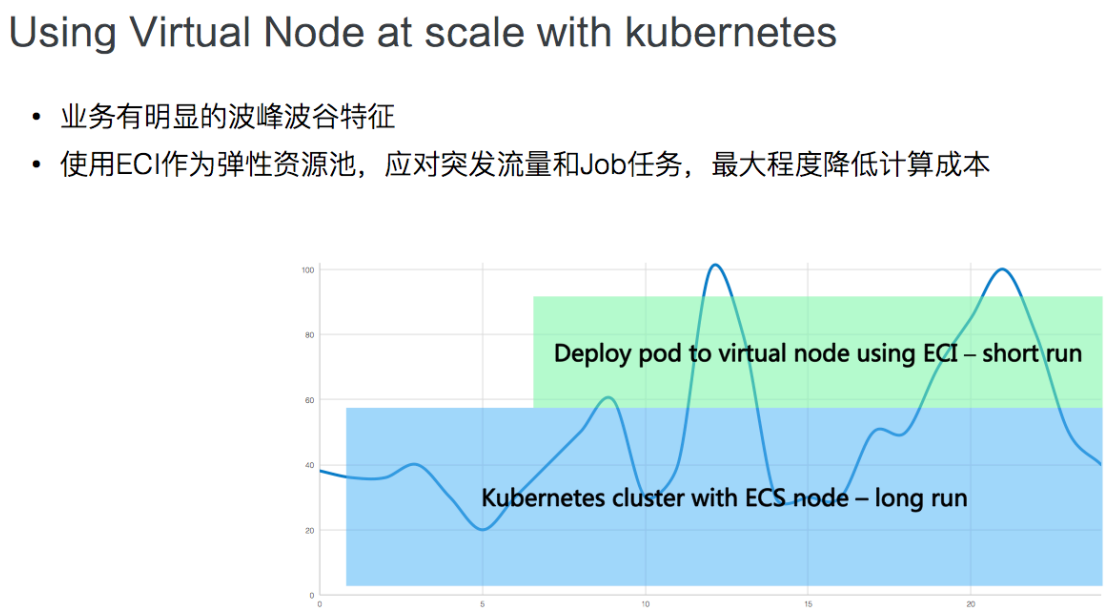
本文主要为您介绍通过虚拟节点运行 Job 任务的方式，您可以用最小的运维成本（无需调整节点数量），来应对集群计算资源高峰压力。

前提条件

- 您已经创建了一个 Kubernetes 托管版集群。请参见创建Kubernetes 托管版集群。
- 您已经在该集群上部署了一个虚拟节点。请参见虚拟节点。
- 您已经给 namespace vk 打上`virtual-node-affinity-injection: enabled`标签。请参见通过配置namespace标签的方式创建Pod。

背景信息

Kuberntes 集群的节点资源配置不足会导致 pod 无法及时运行，购买过多的节点又会导致资源的闲置浪费。而虚拟节点就像是kubernetes集群的“魔法口袋”，让我们摆脱节点计算力不足的烦扰，也避免了节点的闲置浪费，满足无限计算力的想象，pod 按需创建，轻松应对计算的波峰波谷。



本例中，创建的 Kubernetes 托管版集群中，Master节点被托管（用户无需付费），2 台 Worker 分别是 4c8g 规格，所以整个 Kubernetes 托管版集群的可用计算容量是 8c16g。需要运行一个 Job 离线任务做数据处理，Job 的 cpu/mem 需求为 16c32g。此时 Job 无法在您创建

的 Kubernetes 托管版集群中运行，因为其资源需求已经超过 Kubernetes 托管版集群的 8c16g 计算容量。因此，您可以选择把这个 Job 任务调度到虚拟节点上，而不占用集群本身节点的计算资源。

操作步骤

1. 通过 kubectl 连接 Kubernetes 集群。
2. 创建并拷贝以下内容到*job.yaml*，并执行`kubectl -n vk apply -f job.yaml`命令，部署一个job 任务。

```
kind: Job
metadata:
  name: pi
spec:
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000
)"]
        resources:
          requests:
            cpu: 16
            memory: 32Gi
      restartPolicy: Never
  backoffLimit: 4
```

3. 执行如下命令，查看Pod的运行状态。

```
#kubectl -n vk get pod -a
NAME        READY      STATUS       RESTARTS    AGE
pi-7cmwv    0/1        Completed    0           2m

#kubectl -n vk describe pod
Name:                 pi-7cmwv
Namespace:            vk
Priority:             0
PriorityClassName:    <none>
Node:                 virtual-kubelet/
...
Events:
  Type    Reason                  Age    From                 Message
  ----    ------                  ----   ----                 -------
  Normal  Scheduled               3m     default-scheduler    Successful
ly assigned vk/pi-7cmwv to virtual-kubelet
  Normal  SuccessfulMountVolume   2m     kubelet, eci         MountVolum
e.SetUp succeeded for volume "default-token-b2tff"
  Normal  Pulling                 2m     kubelet, eci         pulling
image "perl"
  Normal  Pulled                  2m     kubelet, eci         Successful
ly pulled image "perl"
  Normal  Created                 2m     kubelet, eci         Created
container
```

```
  Normal  Started                 2m    kubelet, eci       Started
container
```

说明:

虚拟节点中的 pod 完全是按需收费，当 pod 执行完成后就会停止收费。

可以看到，使用虚拟节点运行 Job 短时间任务可以节省计算成本，减轻集群运维负担，用户无需关心集群的计算资源是否充足，也无需关心节点的扩容和缩容。

8.2 基于虚拟节点部署 Ingress 应用

本文主要介绍在虚拟节点上部署Ingress应用，使得集群无需创建新节点即可为该应用扩充无限容量，满足业务高峰低谷的弹性需求。

前提条件

您已经部署了一个虚拟节点。请参见虚拟节点。

您已经给namespace vk打上`virtual-node-affinity-injection: enabled`标签。请参见通过配置namespace标签的方式创建Pod。

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 > 无状态，单击右上角的使用模板创建。
3. 选择所需的集群和命名空间vk，选择样例模板或自定义，然后单击创建。

您可以使用如下 yaml 示例模板创建Ingress。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: coffee
spec:
  replicas: 2
  selector:
    matchLabels:
      app: coffee
  template:
    metadata:
      labels:
        app: coffee
    spec:
      containers:
      - name: coffee
        image: nginxdemos/hello:plain-text
        ports:
        - containerPort: 80
---
apiVersion: v1
```

```
kind: Service
metadata:
  name: coffee-svc
spec:
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: coffee
  clusterIP: None
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: tea
spec:
  replicas: 3
  selector:
    matchLabels:
      app: tea
  template:
    metadata:
      labels:
        app: tea
    spec:
      containers:
      - name: tea
        image: nginxdemos/hello:plain-text
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: tea-svc
  labels:
spec:
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: tea
  clusterIP: None
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: cafe-ingress
spec:
  rules:
  - host: cafe.example.com
    http:
      paths:
      - path: /tea
        backend:
          serviceName: tea-svc
          servicePort: 80
      - path: /coffee
        backend:
          serviceName: coffee-svc
```

```
            servicePort: 80
```

预期结果

- **在 Kubernetes 菜单下，单击左侧导航栏的应用 > 无状态，选择目标集群和命名空间，可以看到刚刚创建的**`coffee`**和**`tea`**。**
- **在 Kubernetes 菜单下，单击左侧导航栏的应用 > 容器组，选择目标集群和命名空间，可以看到 Pod 都运行在 Virtual-Kubelet 节点上。**
- **在 Kubernetes 菜单下，单击左侧导航栏的路由与负载均衡 > 路由，选择目标集群和命名空间，可以看到刚刚创建的路由。**
- **您可以执行如下命令，确保可以访问 Ingress 应用。**

```
kubectl -n vk get ing
curl -H "Host:cafe.example.com" <EXTERNAL_IP>/tea
curl -H "Host:cafe.example.com" <EXTERNAL_IP>/coffee
```

9 DevOps

9.1 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

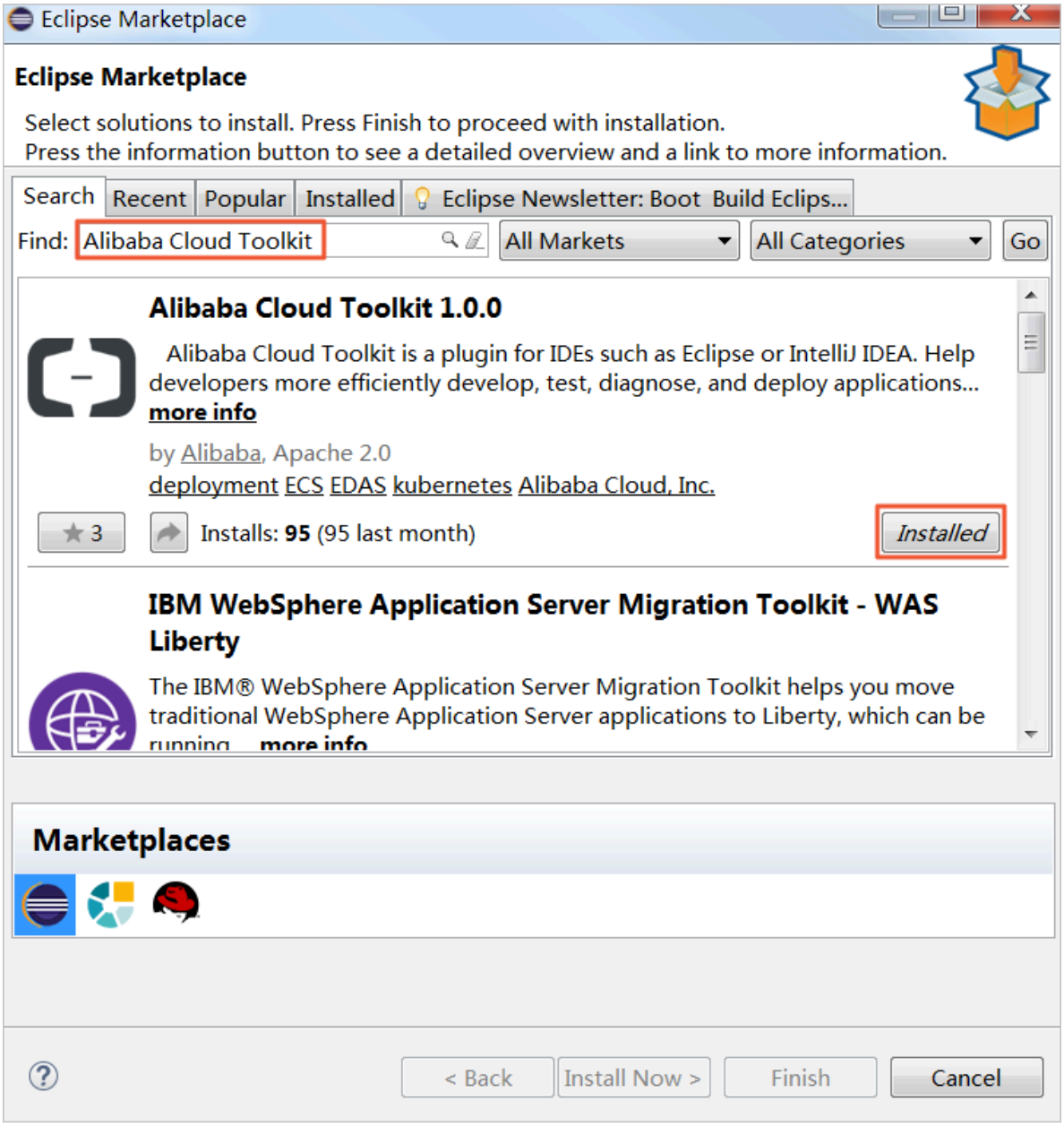
- 下载并安装JDK 1.8 或更高版本。
- 下载并安装适用于 Java EE 开发人员的Eclipse IDE、4.5.0（代号：Mars）或更高版本。

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。



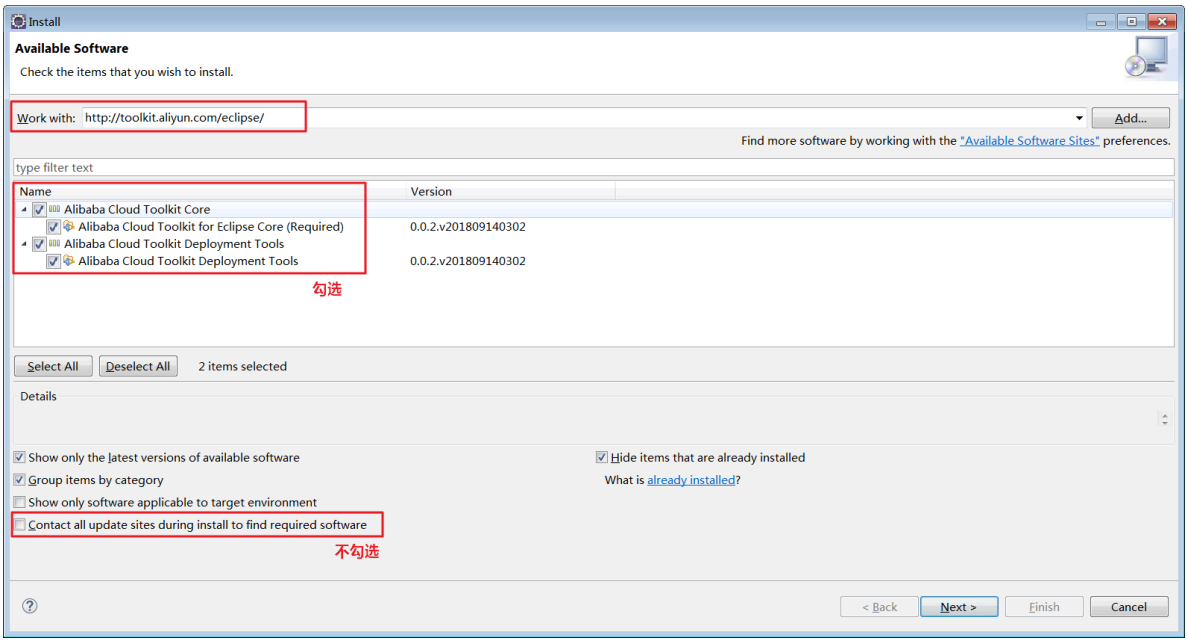
5. 按照Eclipse安装页面的提示，完成后续安装步骤。

直接通过Cloud Toolkit 的官方地址安装

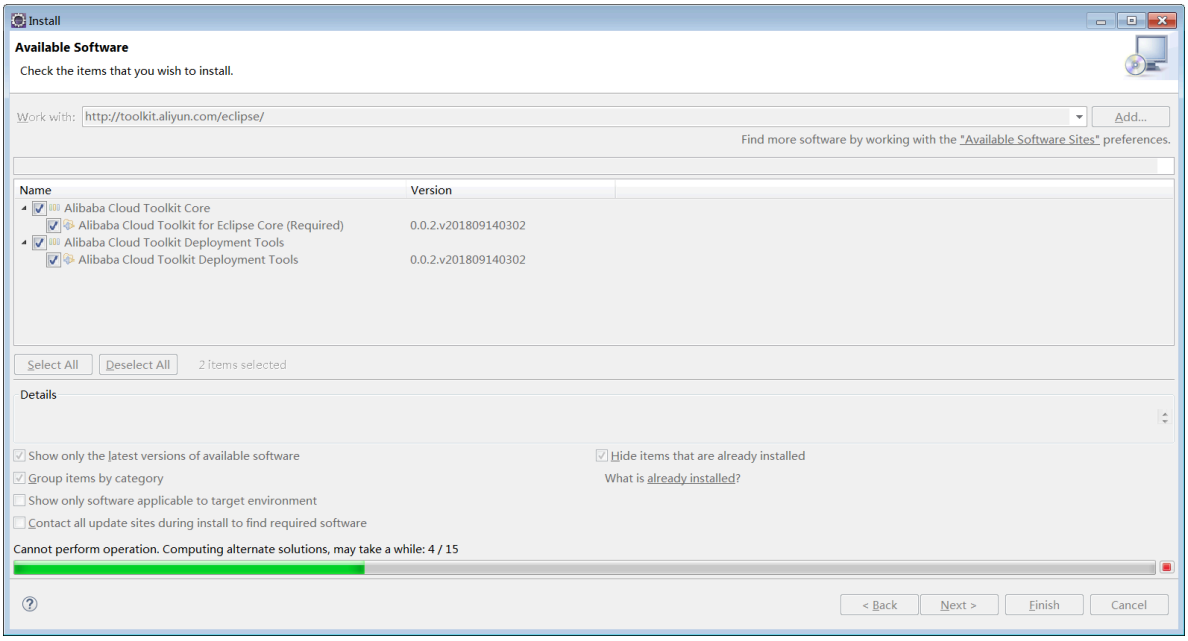
> **说明:**
> 如果您无法连接Eclipse Market服务器，请选择这种安装方式。

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.`



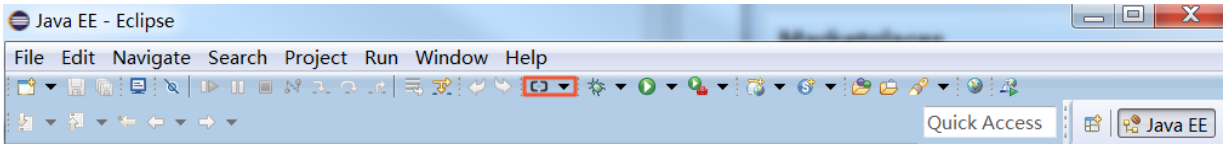
5. 配置完成后，单击Next，Elipse开始安装插件，并显示安装进度。



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！

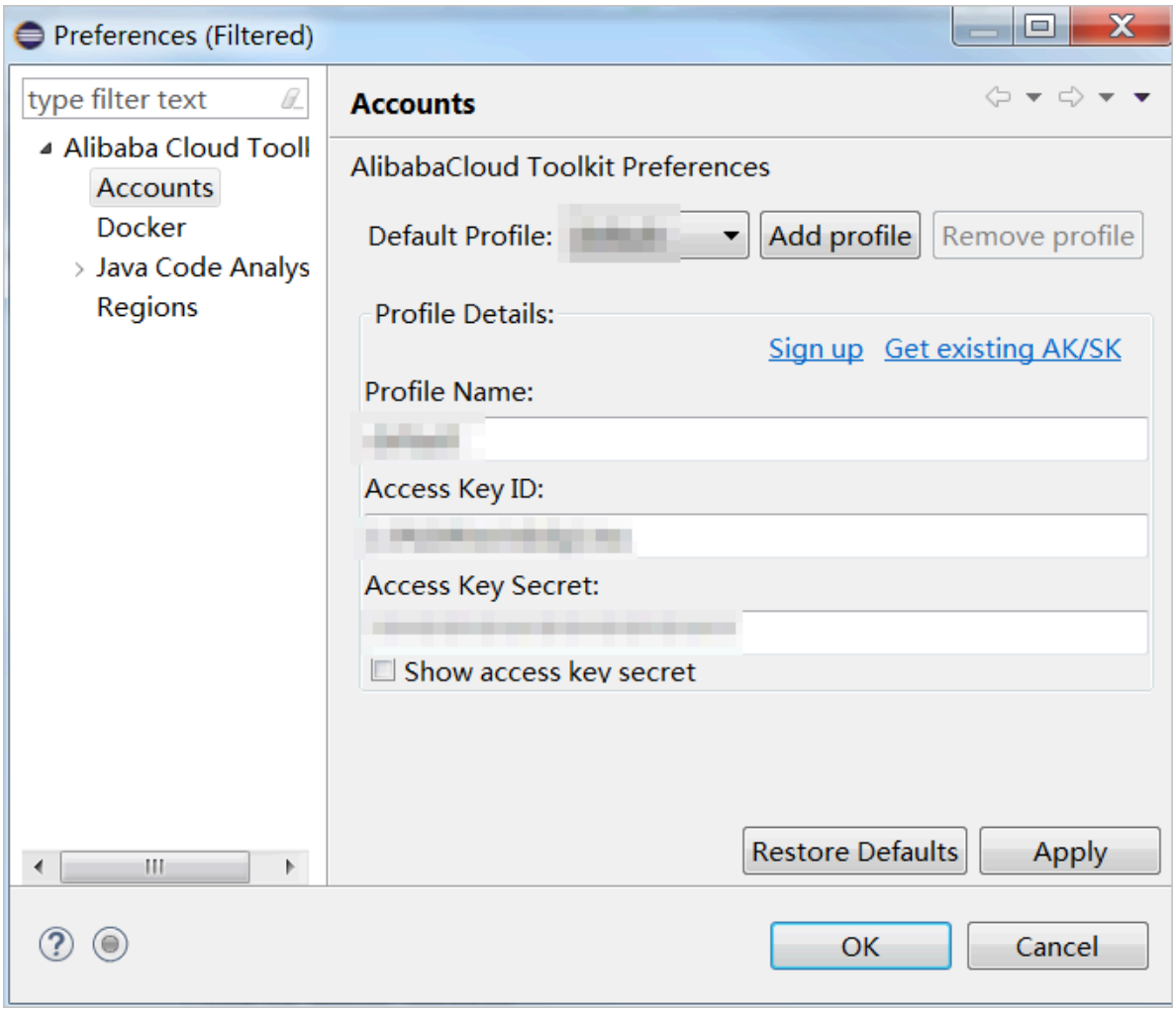
设置Access Key ID和Access Key Secret

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，可参考镜像创建无状态Deployment应用或镜像创建有状态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。

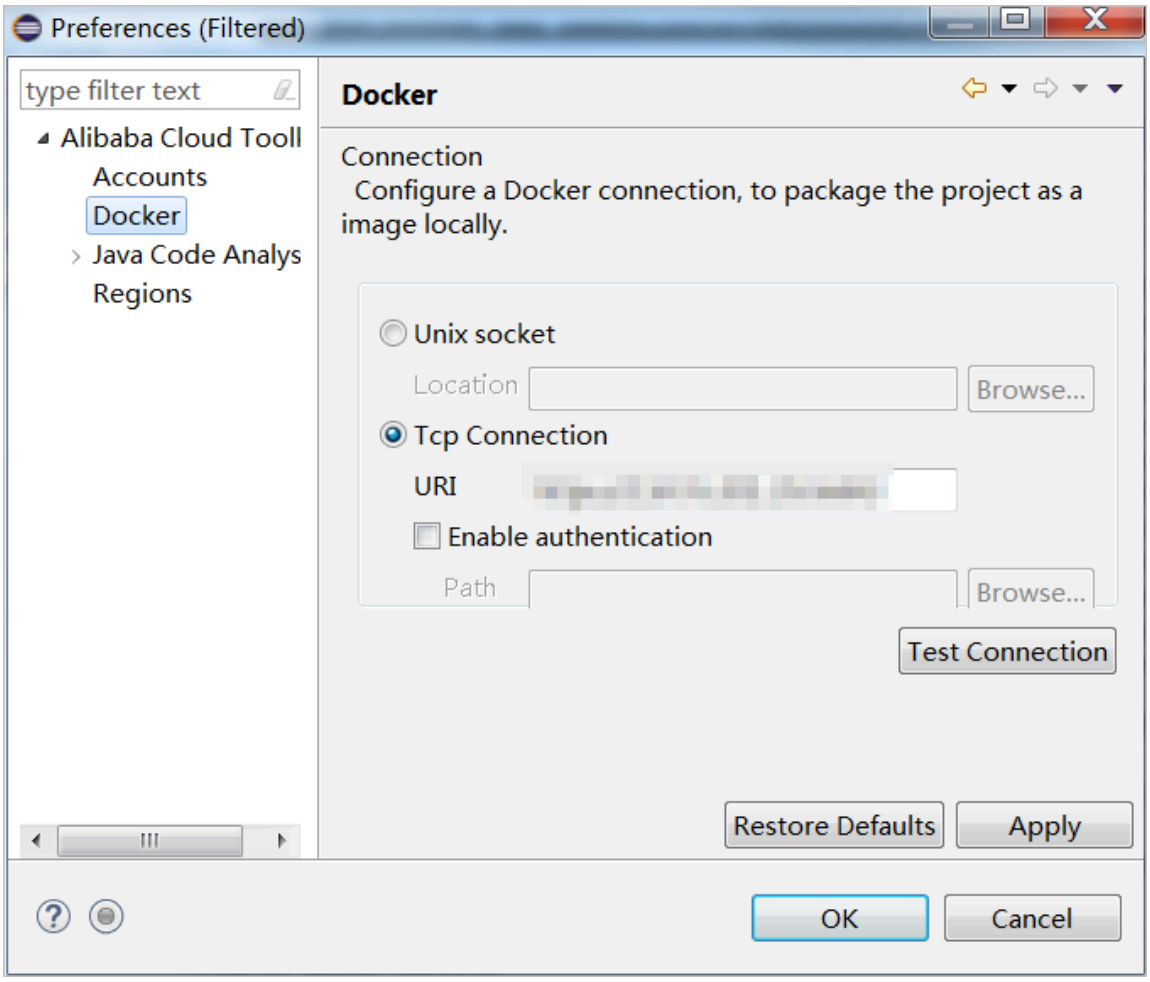
   c. 在Docker界面中设置可连接的 Docker 环境，包括本地和远程两种方式，然后单击OK。

   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`。

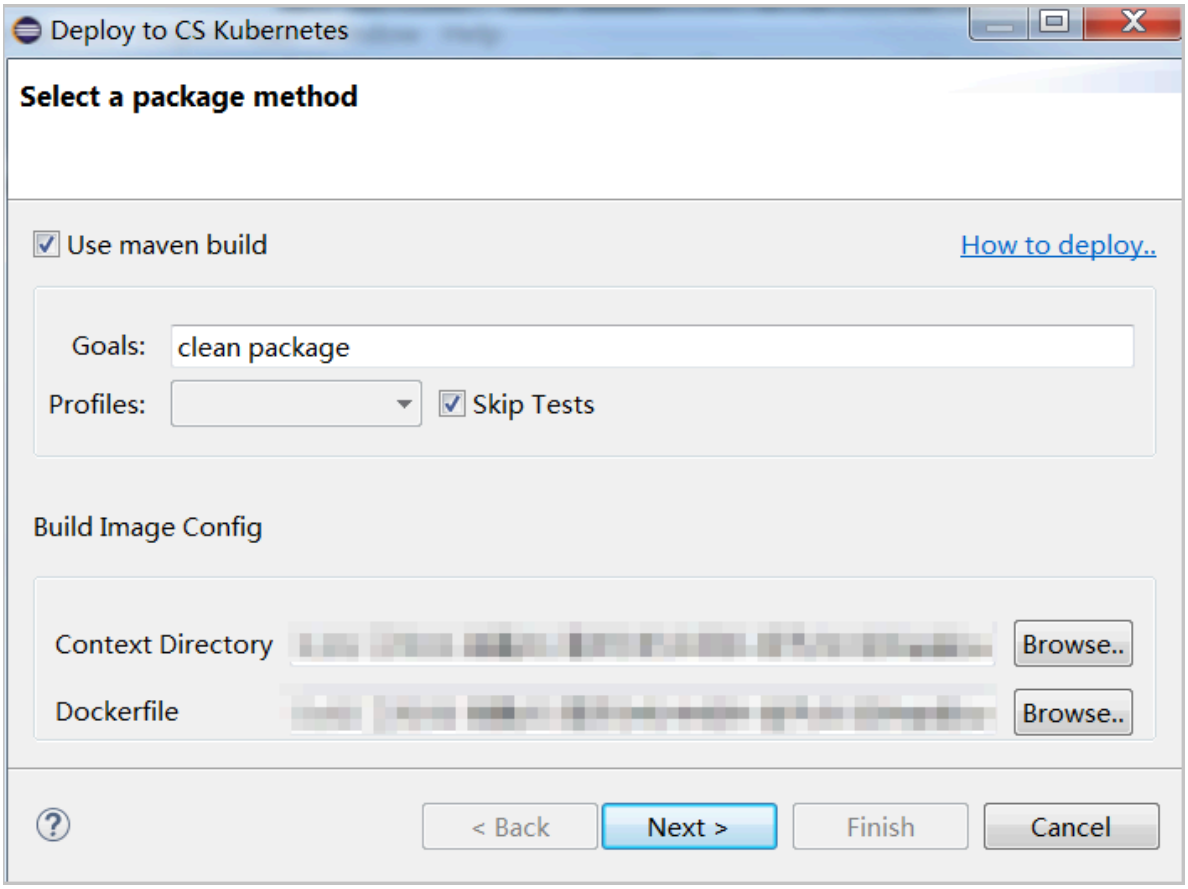
   远程Docker环境

   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。

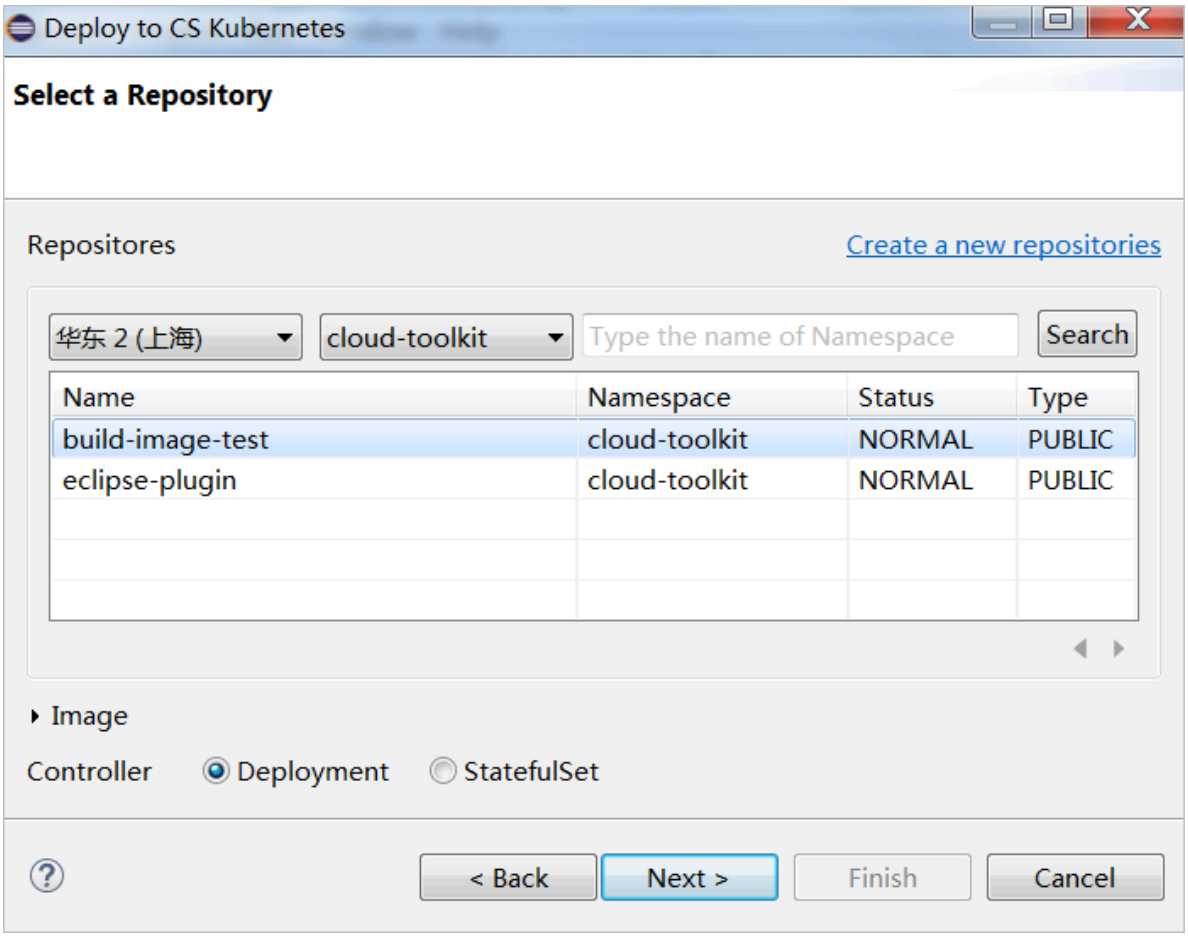
> **说明:**
> 您可以根据您的需要决定是否勾选Use maven build使用Maven构建应用工程。



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。

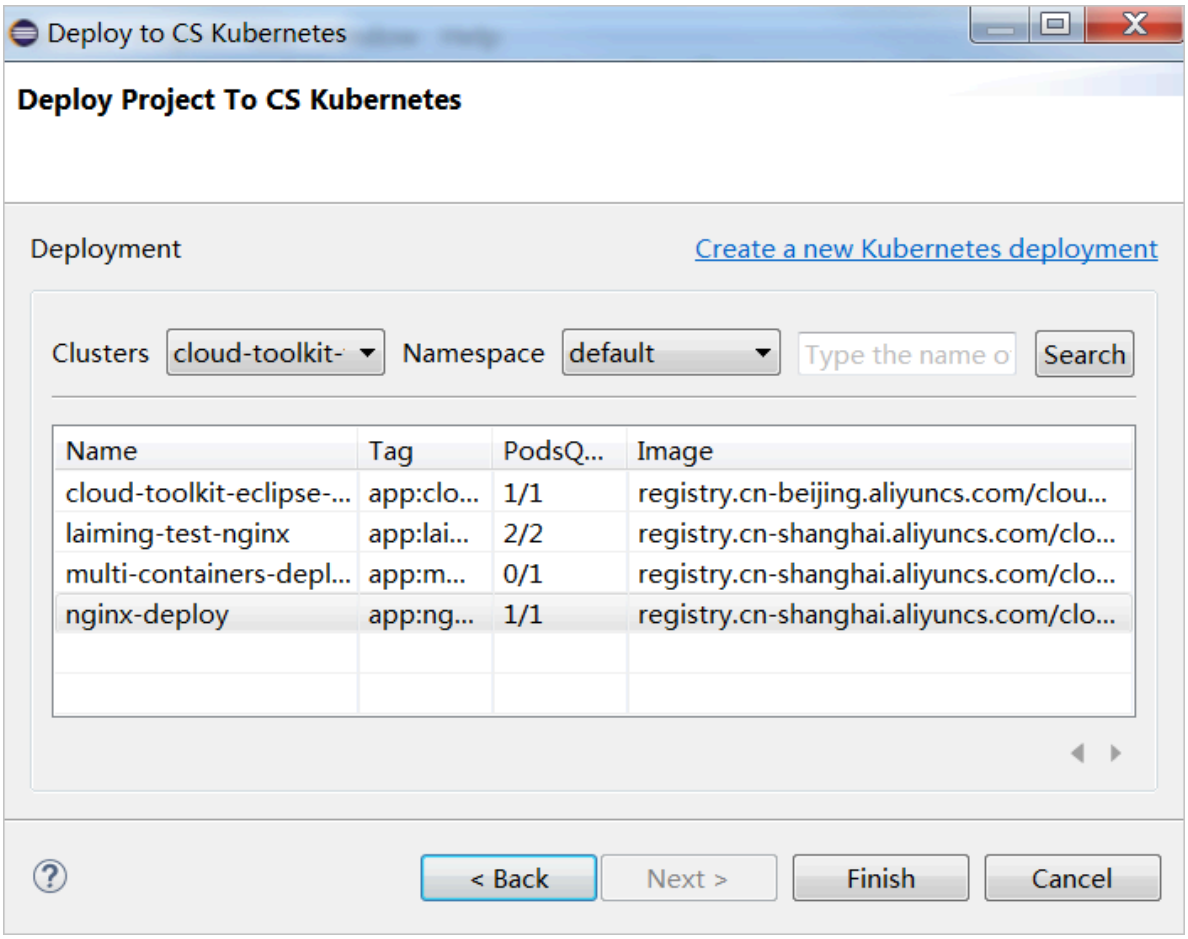


5. 在Deploy Project to CS Kubernetes对话框选择容器服务Kubernetes的Cluster、Namespace和Deployment，然后单击Finish。

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。

9.2 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

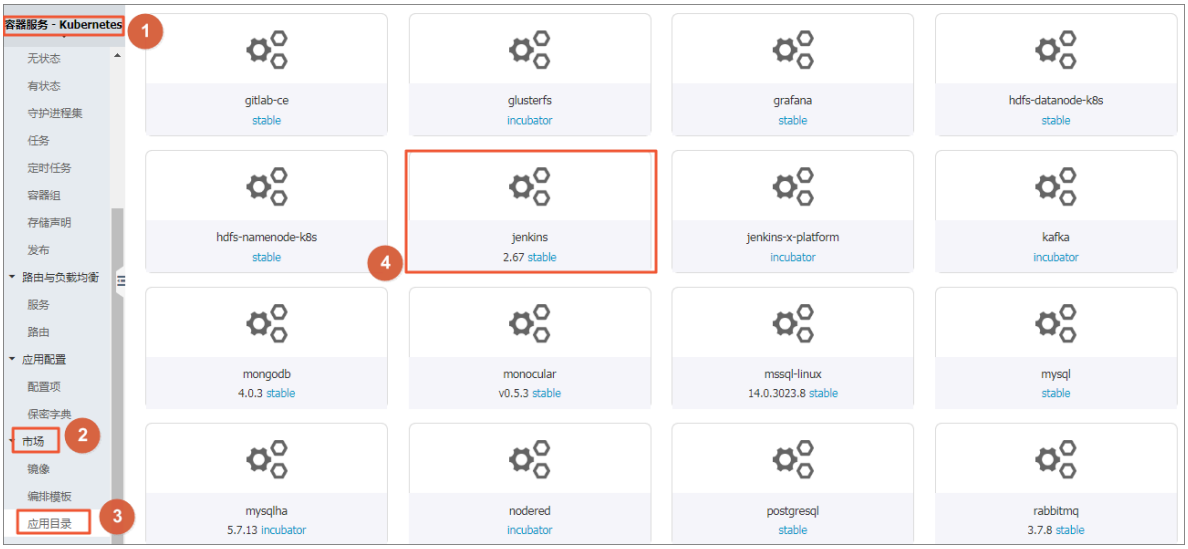
本文主要演示如何在阿里云Kubernetes服务上快速搭建Jenkins持续集成环境，并基于提供的示例应用快速完成应用源码编译、应用镜像构建和推送以及应用部署的流水线。

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#unique_11。

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。

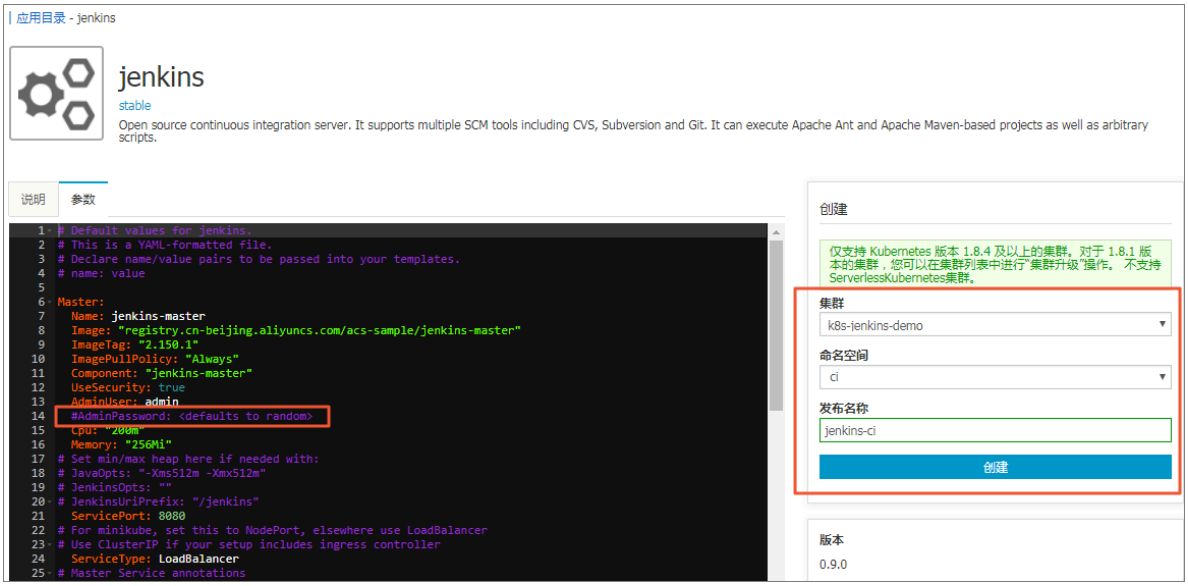
4. 修改`AdminPassword`字段。

> **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
```

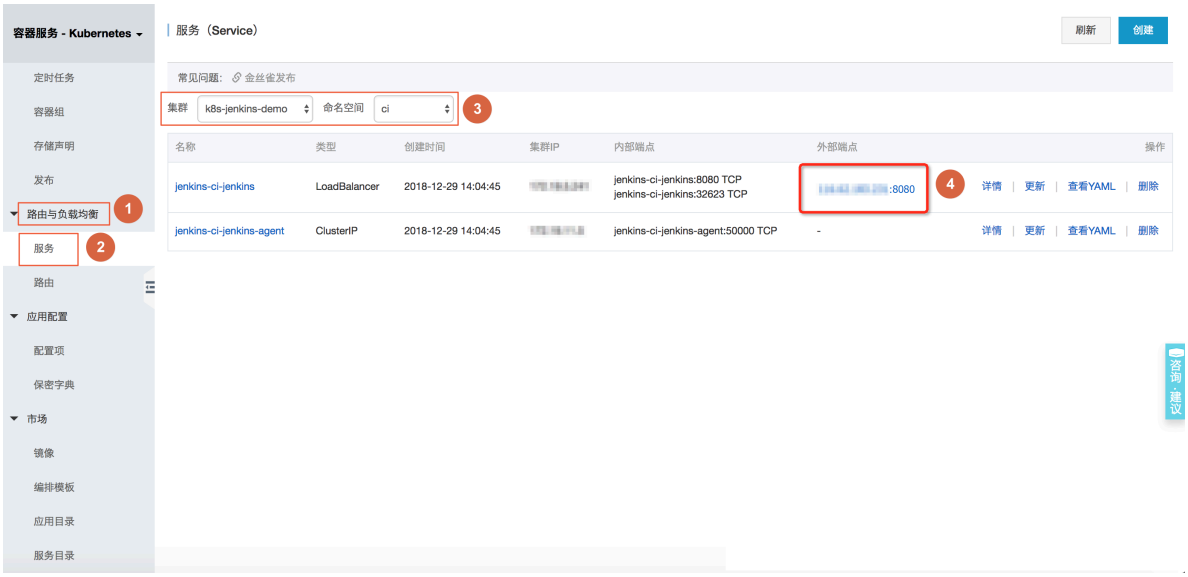
5. 选择Kubernetes集群以及命名空间，填写发布名称，并单击创建。



> **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。



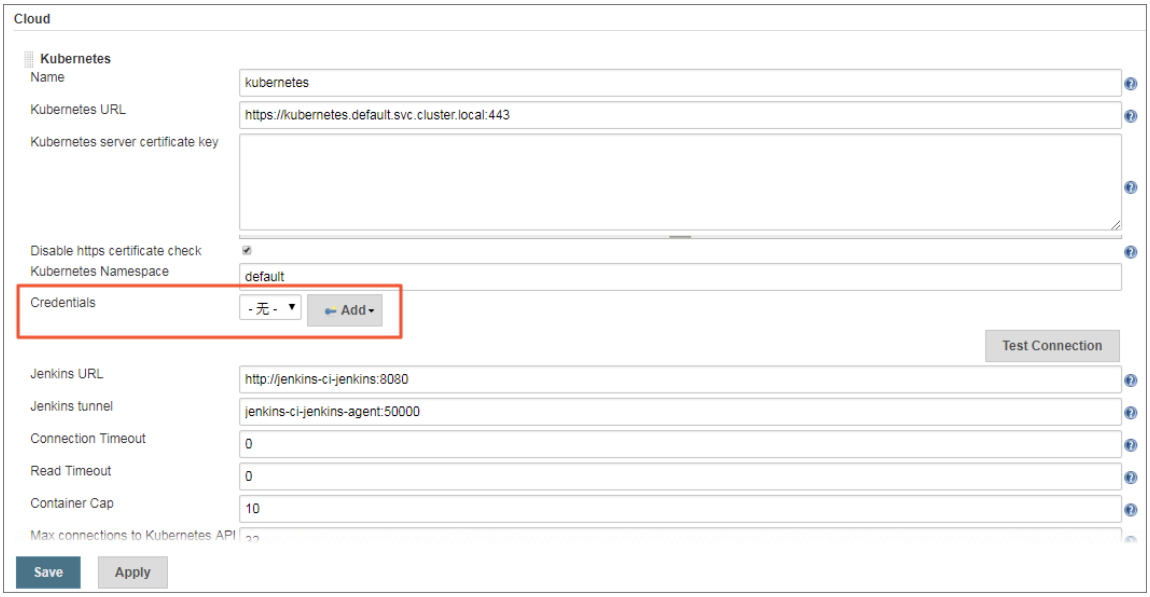
创建集群证书和镜像仓库证书，构建和部署示例应用

1. 创建Kubernetes集群证书。

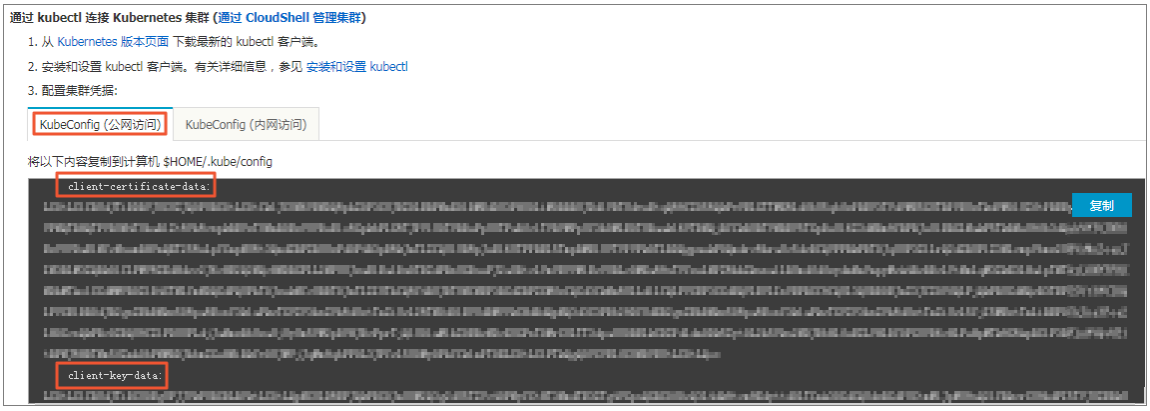
   a. 在Jenkins的左侧导航栏中，选择系统管理。

   b. 在右侧的Manage Jenkins页面，单击系统设置。

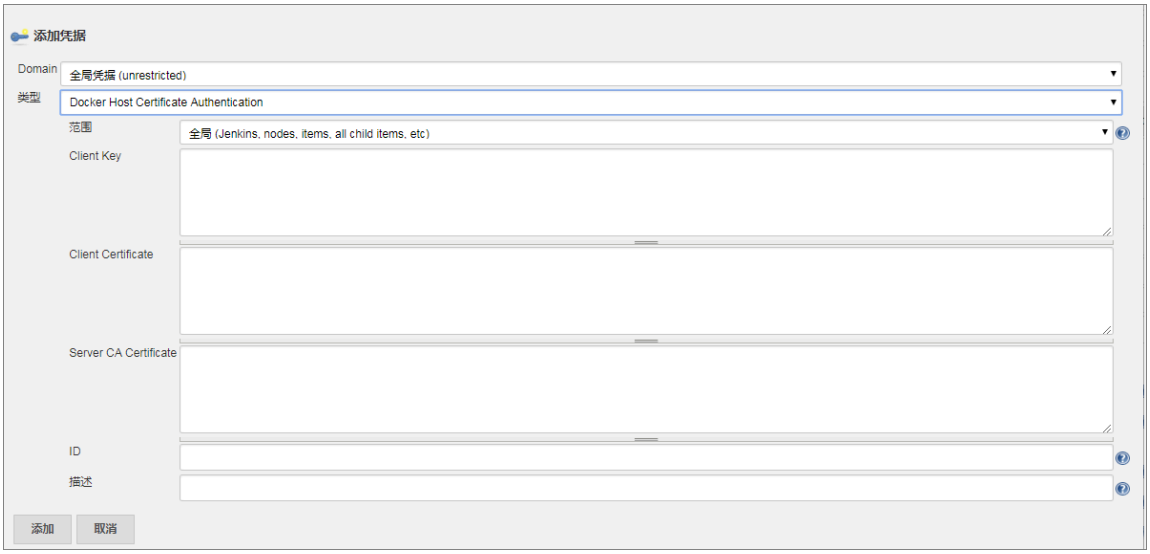
   c. 在Cloud区域中，单击Credentials右侧的Add。



   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：

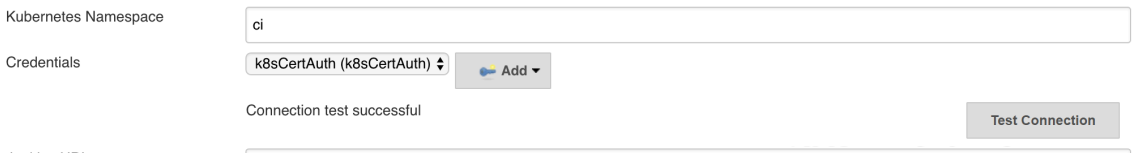


- 类型：选择Docker Host Certificate Authentication。
- Client Key: 填写KubeConfig中client-key-data对应的内容。
- Client Certificate: 填写KubeConfig中client-certificate-data对应的内容。
- ID：填写证书ID。本示例中填写k8sCertAuth。
- 描述：填写描述。

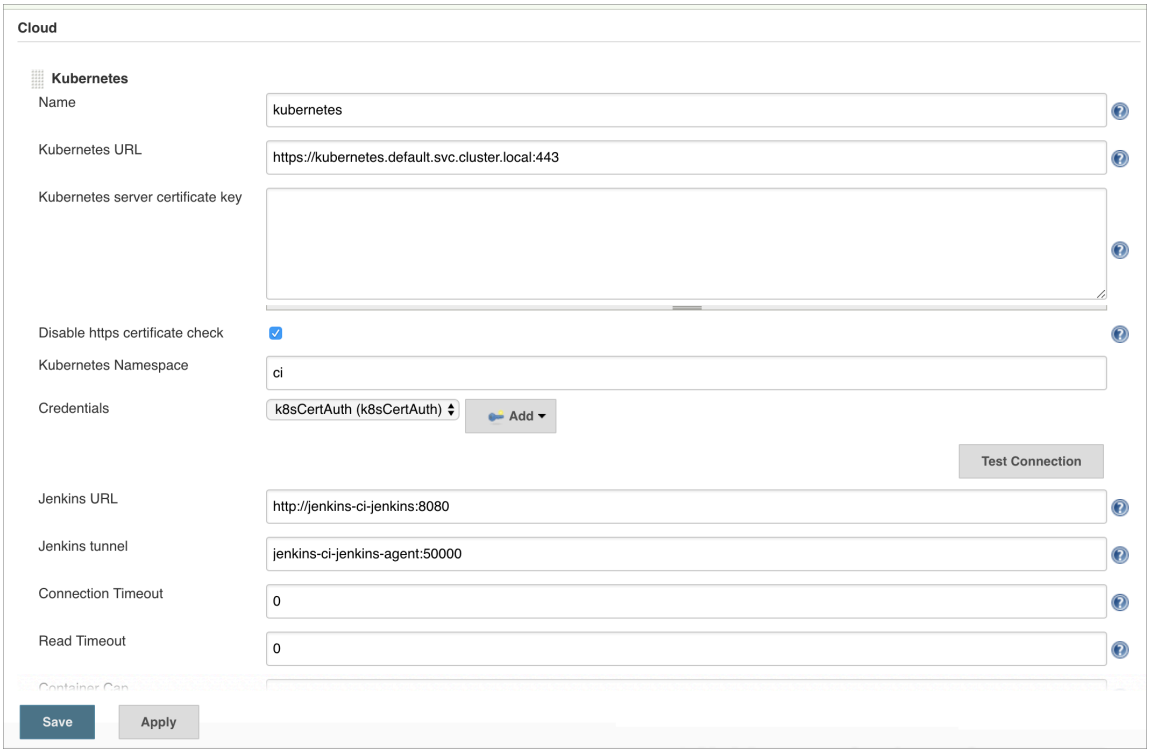
e. 单击添加。

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。



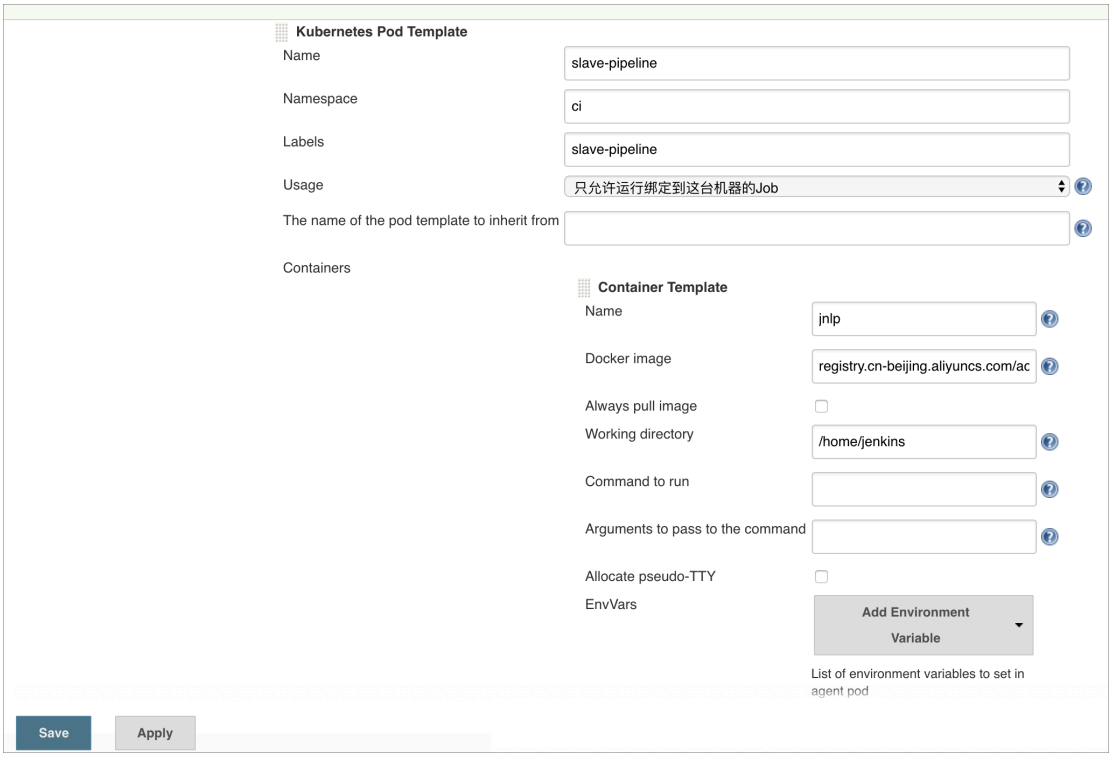
g. Kubernetes集群动态分配构建pod，其配置如下图所示。



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。

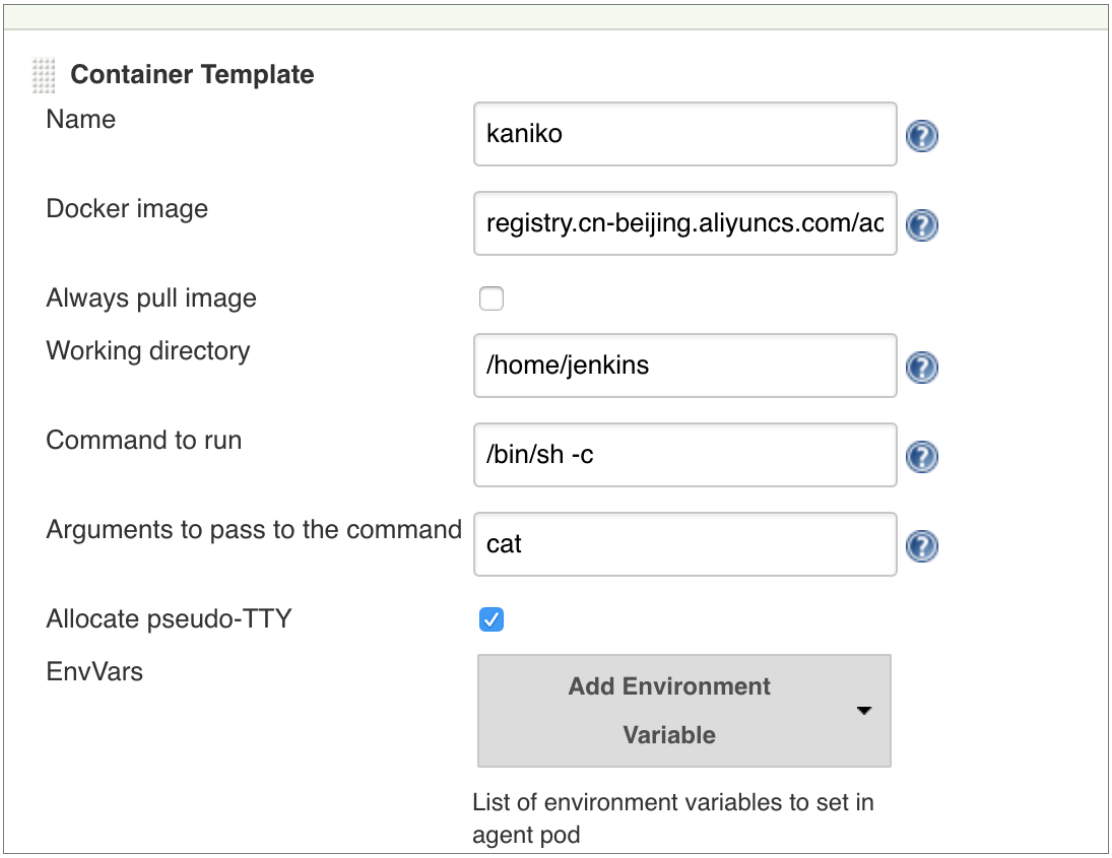
- Container jnlp的配置如下图所示。



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
```

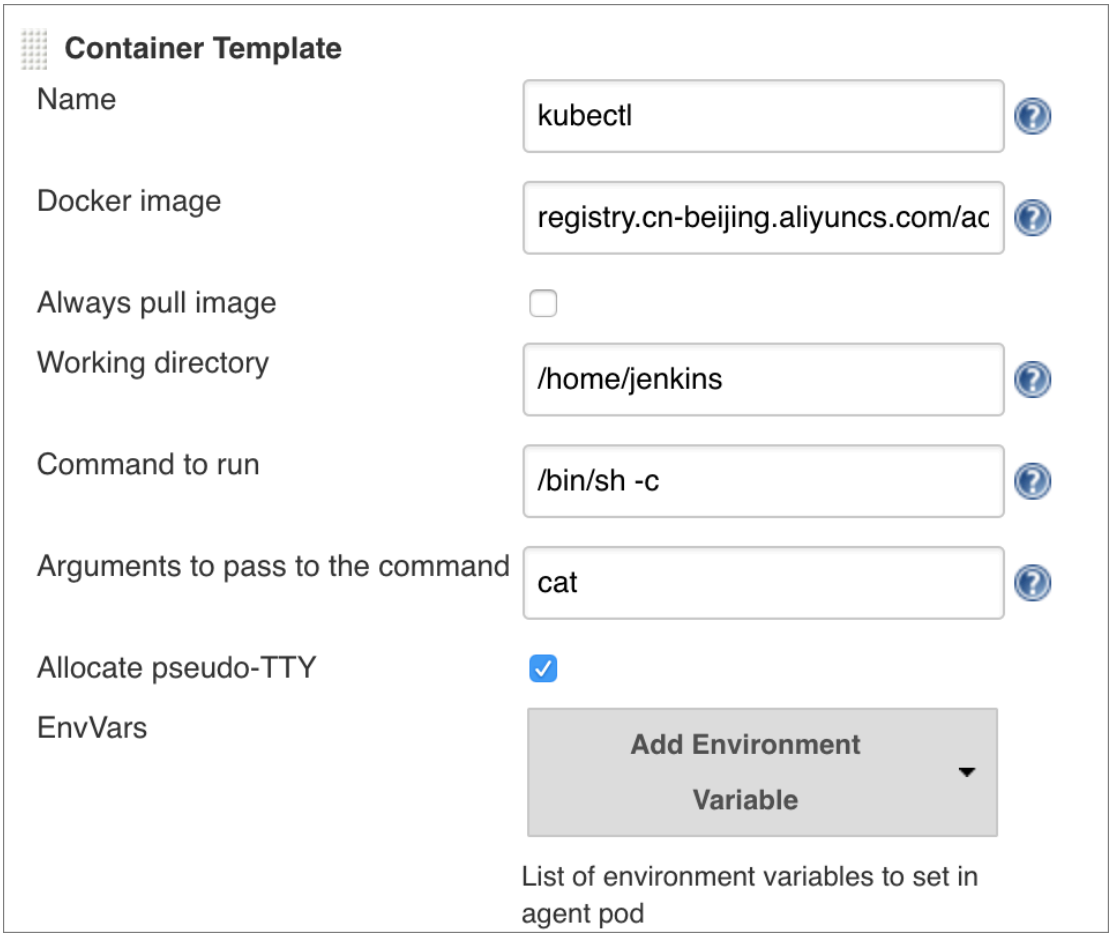
- Container kaniko的配置如下图所示。



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
```

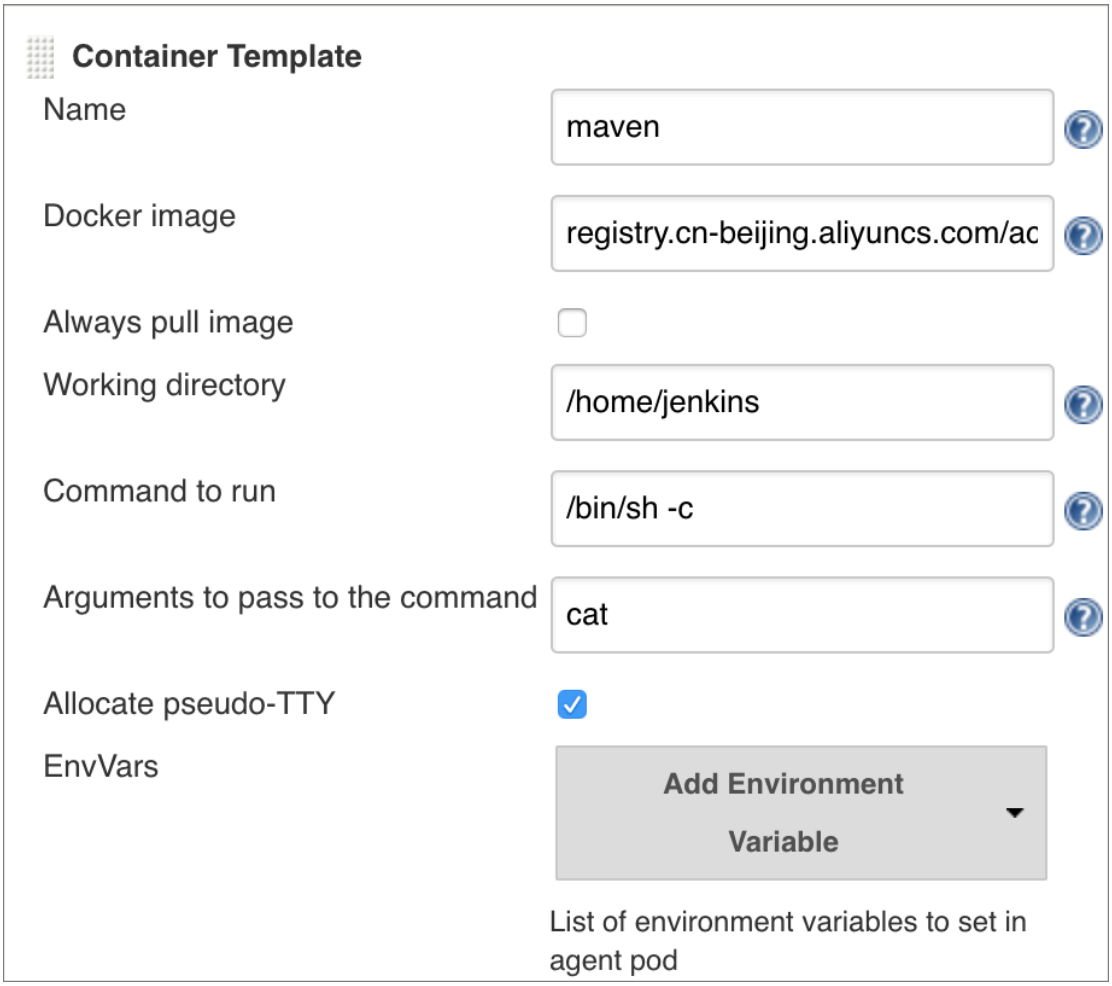
- Container kubectl的配置如下图所示。



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
```

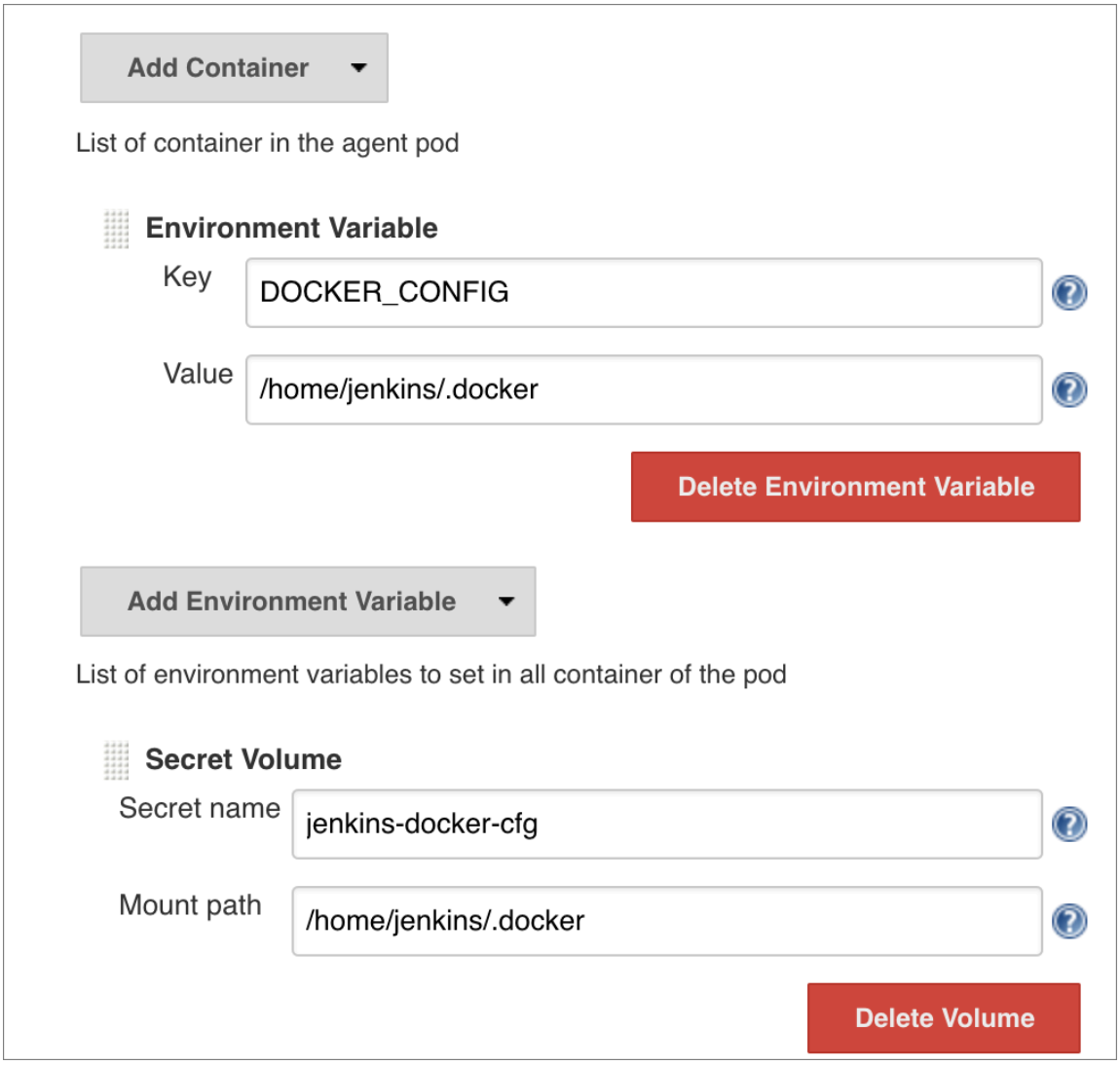
- Container maven的配置如下图所示。



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。

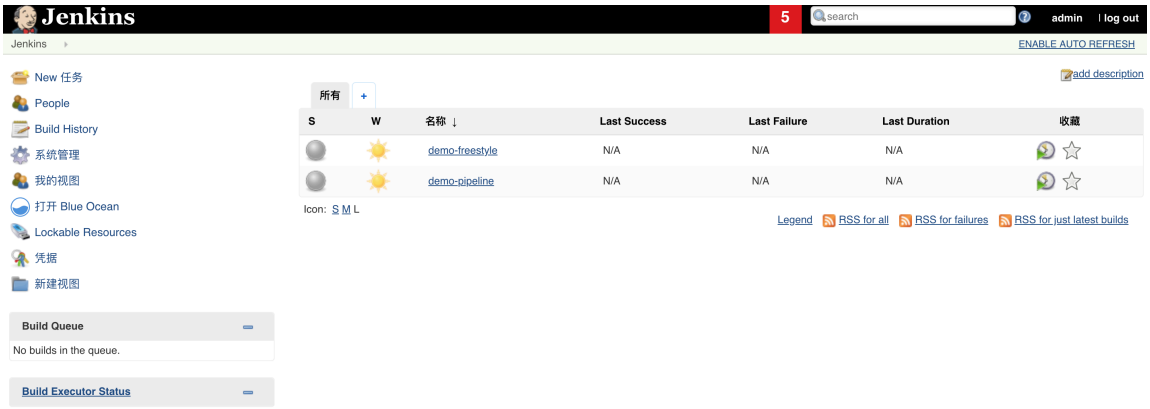
   本示例中使用阿里云镜像服务提供的北京区域镜像仓库：

```
$ 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
```

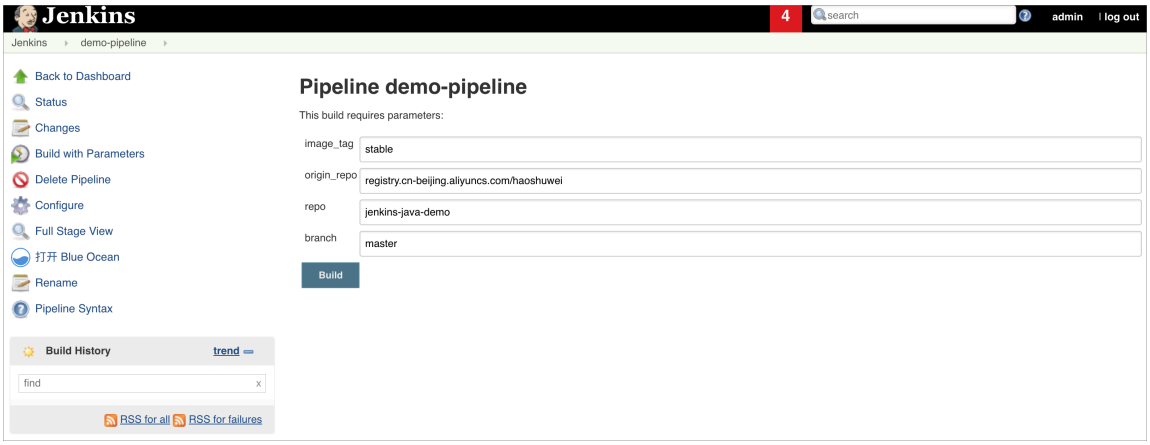
3. 构建demo-pipeline并访问应用服务。

   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。

9.3 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。

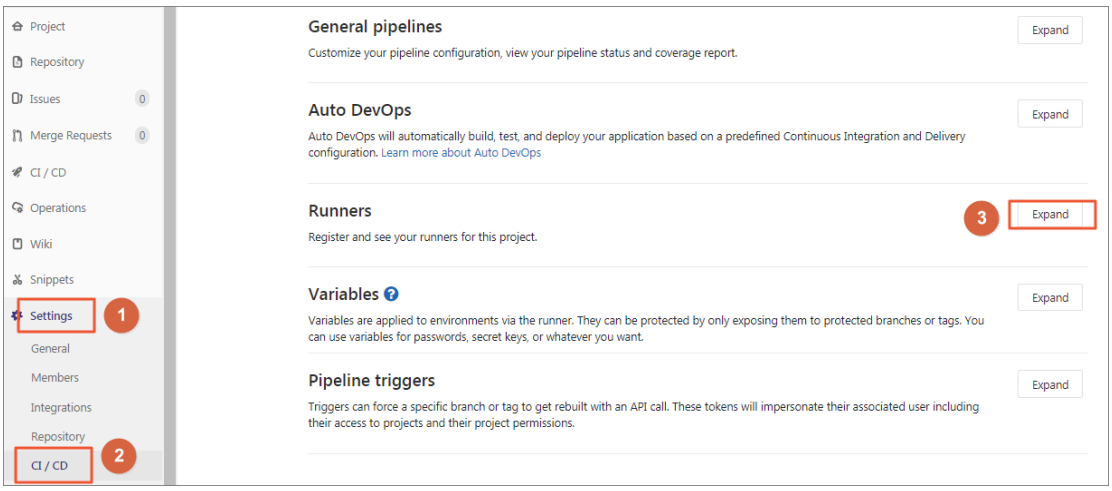
      A. 登录GitLab。

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

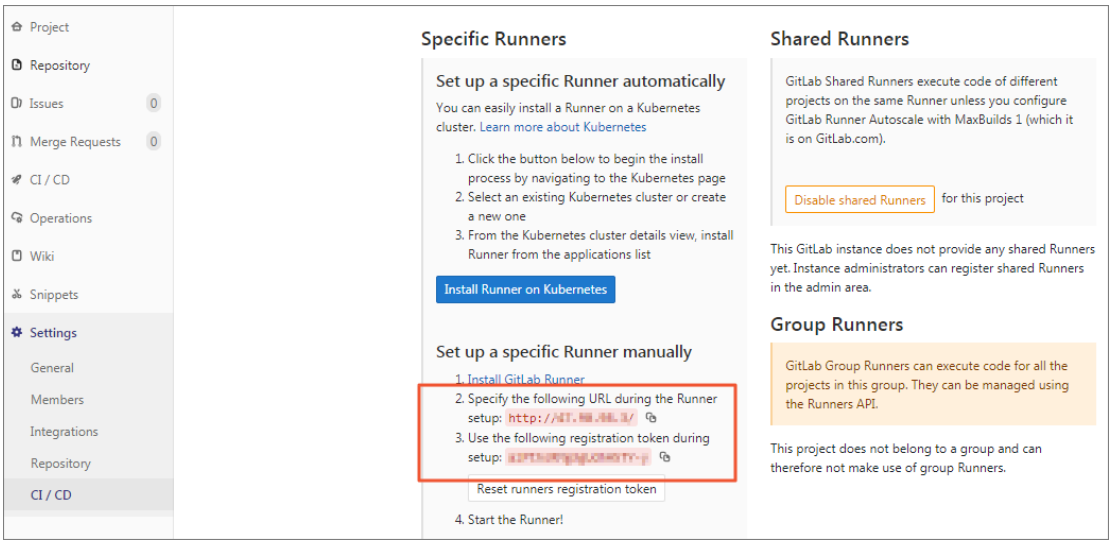
      C. 在Your projects页签下，选择相应的project。

      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。



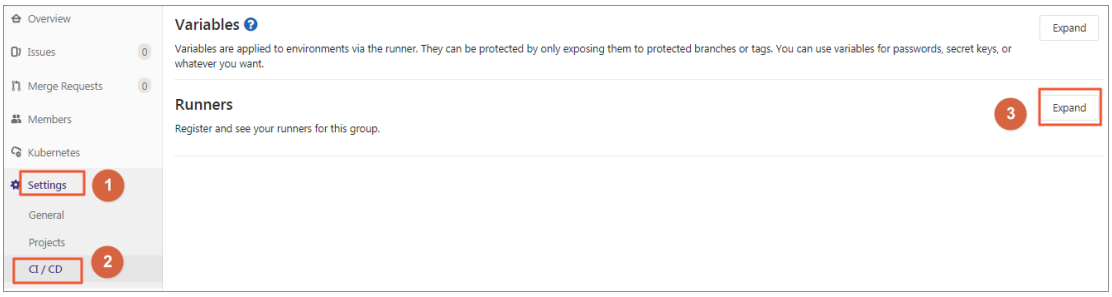
   b) 获取Group Runners的注册信息。

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

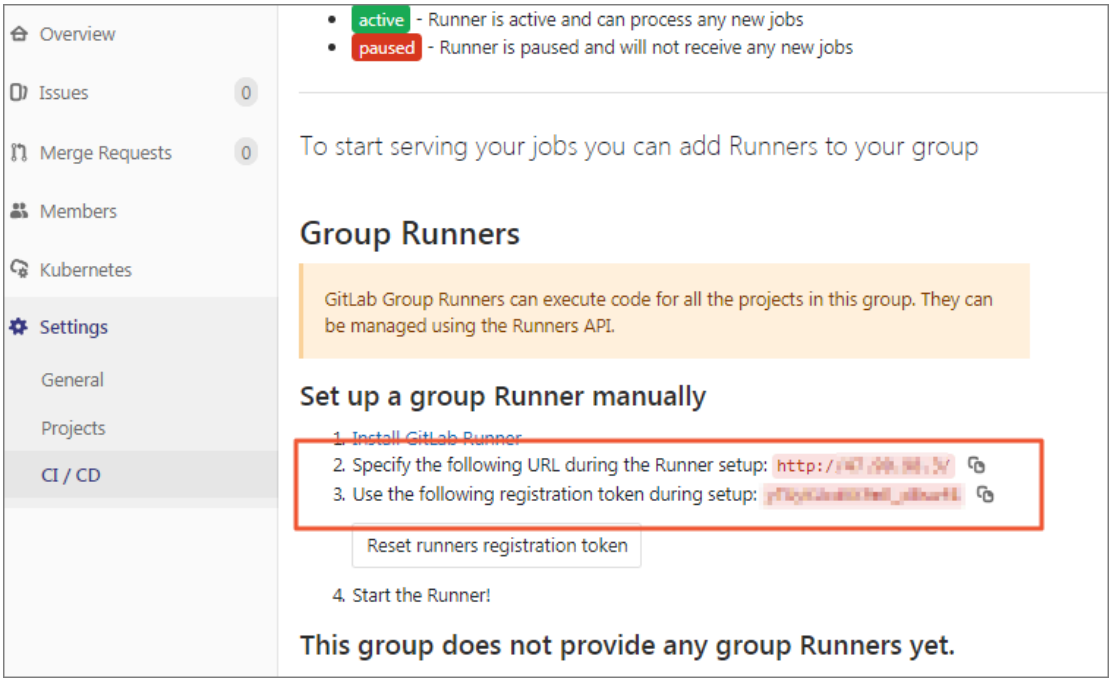
      B. 在Your groups页签下，选择相应的group。

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://github.com/haoshuwei/ack-gitlab-runner.git
```

修改*values.yaml*文件，示例如下：

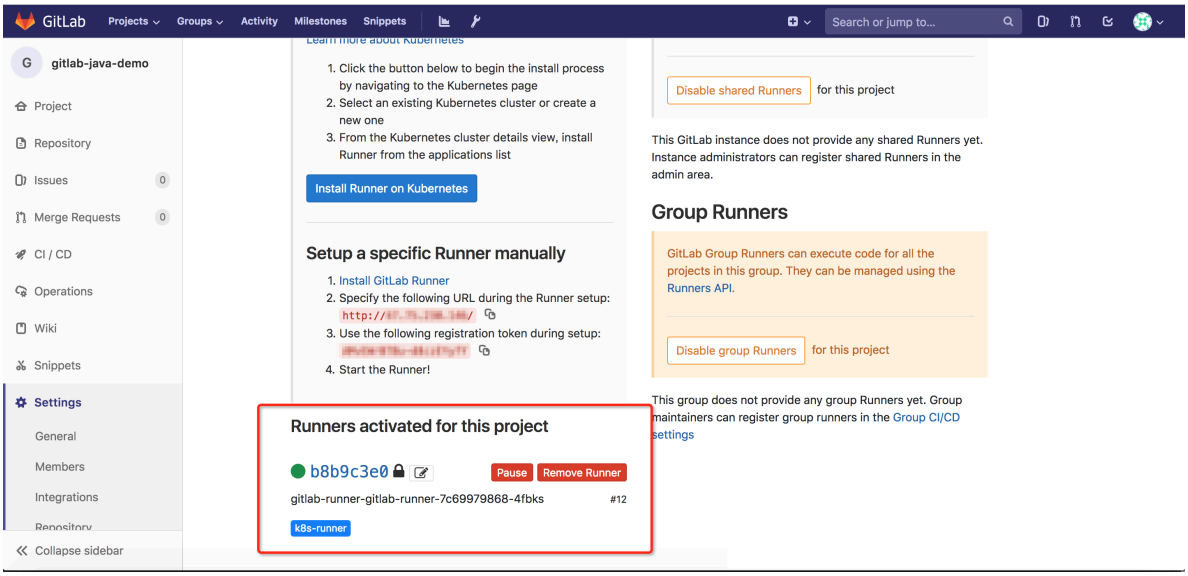
```
## GitLab Runner Image
##
image: gitlab/gitlab-runner:alpine-v12.1.0

## 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：

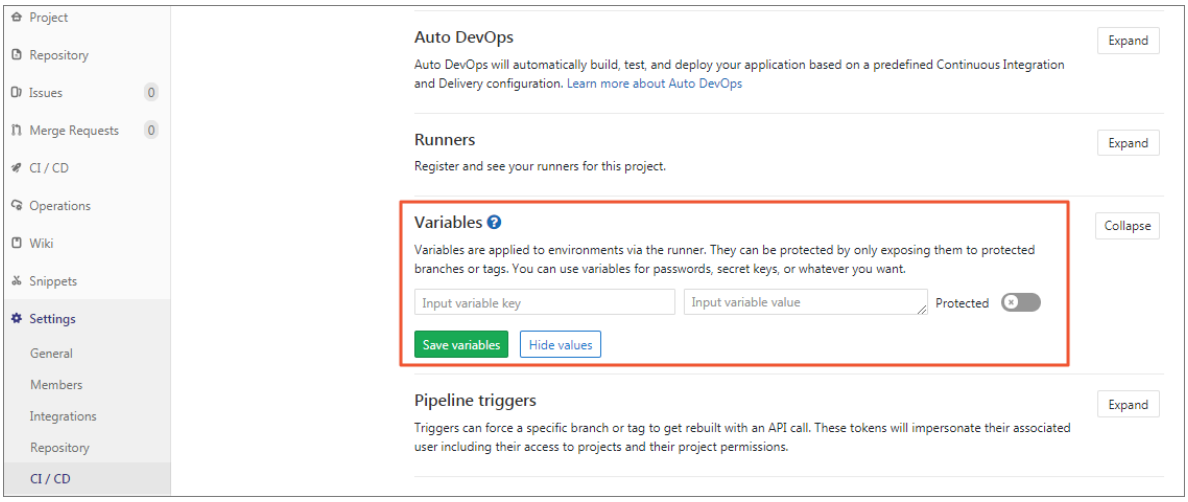
```
cat >>/home/gitlab-runner/.gitlab-runner/config.toml <<EOF
            [[runners.kubernetes.volumes.pvc]]
              name = "gitlab-runner-cache"
              mount_path = "{{ .Values.runners.cachePath }}"
EOF
```

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-runner
  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
  tags:
```

```
    - k8s-runner
  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-runner
  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-runner
  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
  tags:                      # GitLab Runner的tag
    - k8s-runner
  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
```

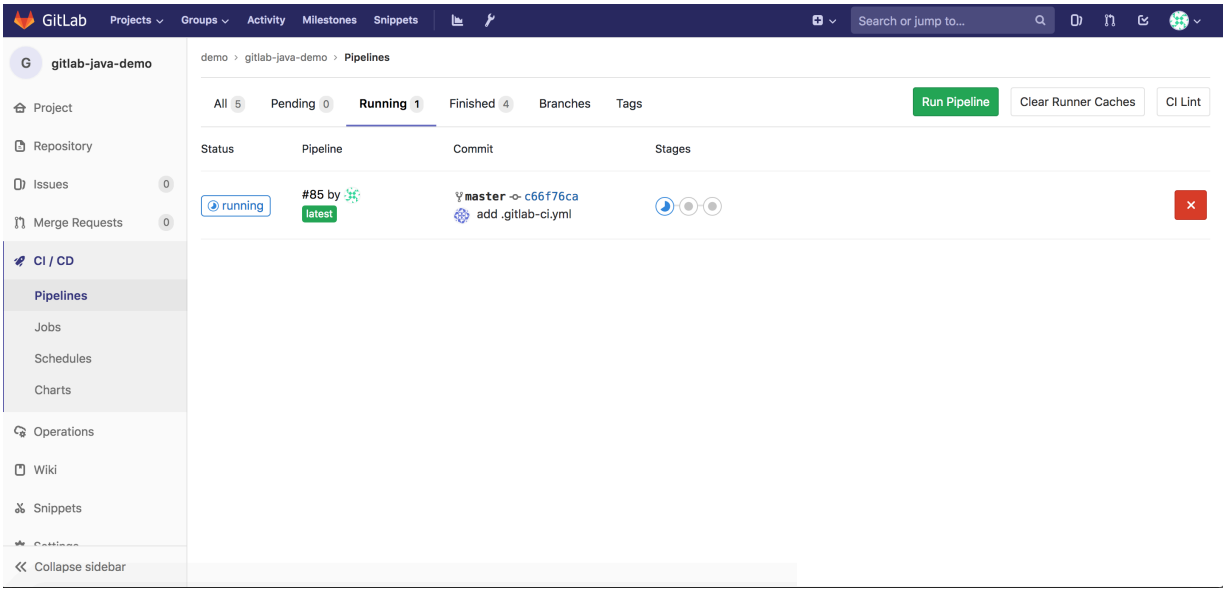
```
    - docker build -t registry.cn-beijing.aliyuncs.com/gitlab-demo
/java-demo:$CI_PIPELINE_ID .      # 打包Docker镜像，使用的tag为本次
Pipeline的ID
    - docker push registry.cn-beijing.aliyuncs.com/gitlab-demo/java-
demo:$CI_PIPELINE_ID       # 推送Docker镜像
```

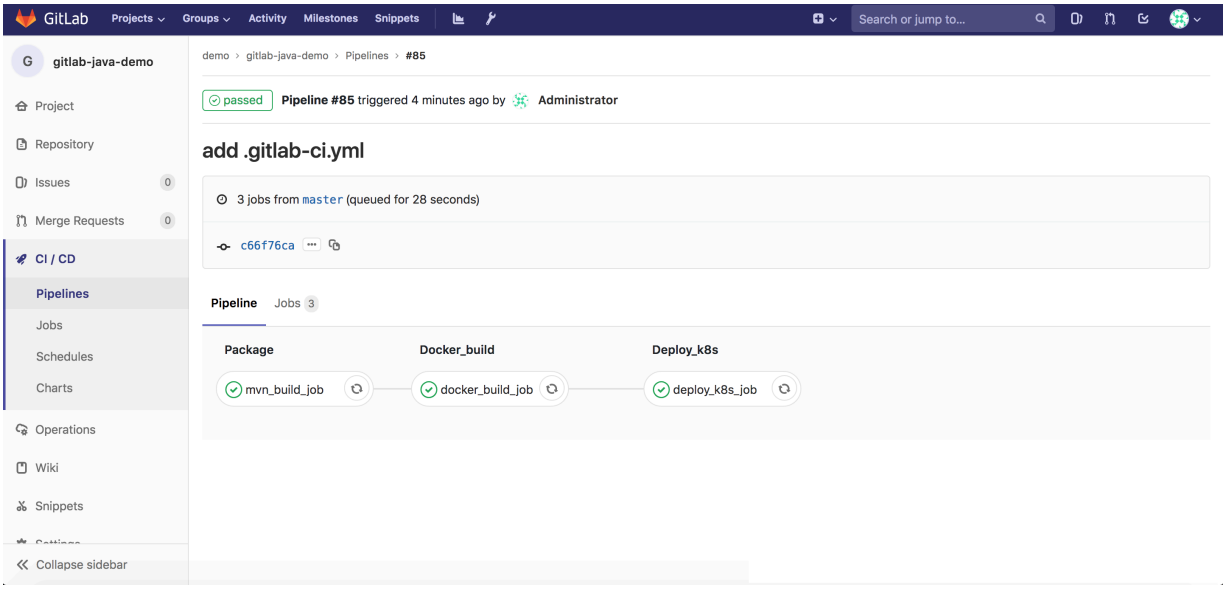
- 应用部署阶段：

```
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-runner
  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。

9.4 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。

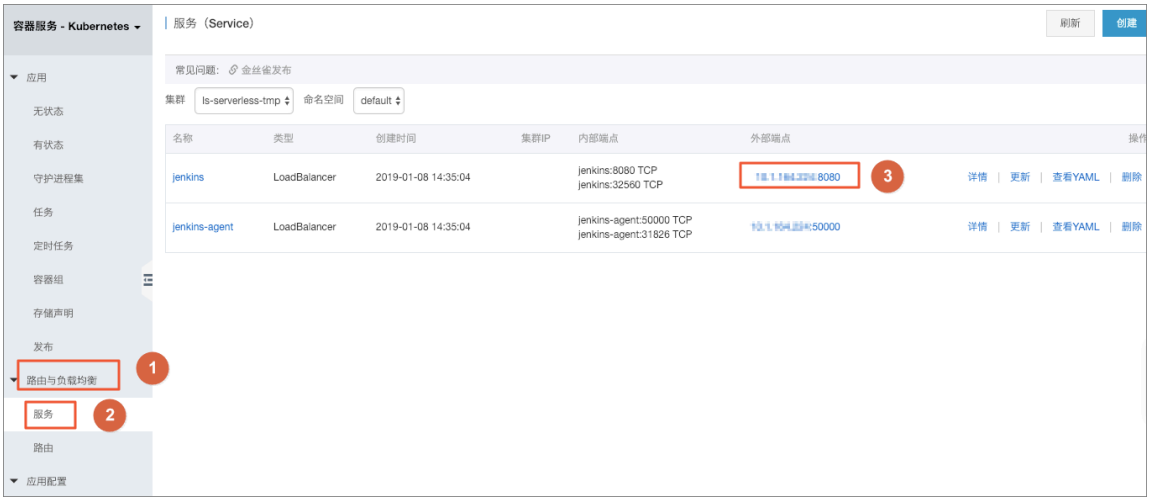
```
$ kubectl apply -f serverless-k8s-jenkins-deploy.yaml
```

4. 登录Jenkins。

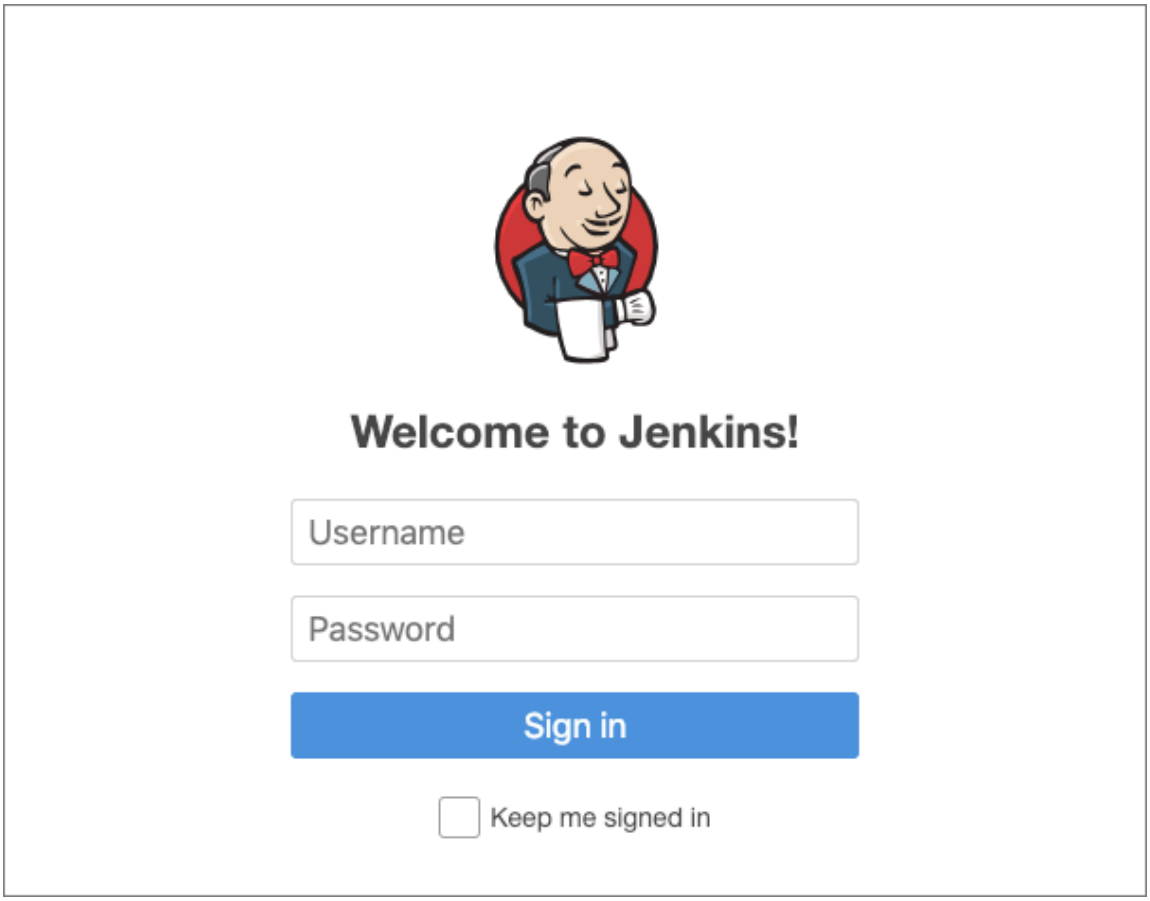
   a. 登录容器服务管理控制台。

   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

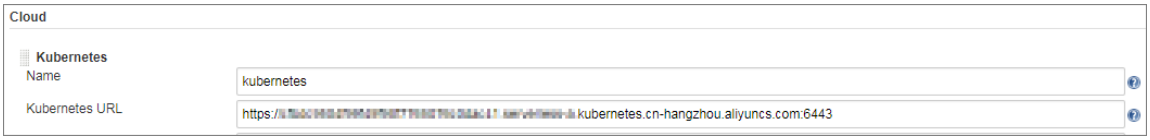
创建集群证书和镜像仓库证书，构建和部署示例应用

1. 配置Kubernetes Cloud，动态创建slave pod。

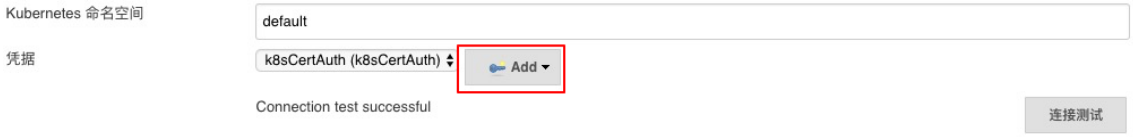
   a. 在左侧导航栏中，选择系统管理。

   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。

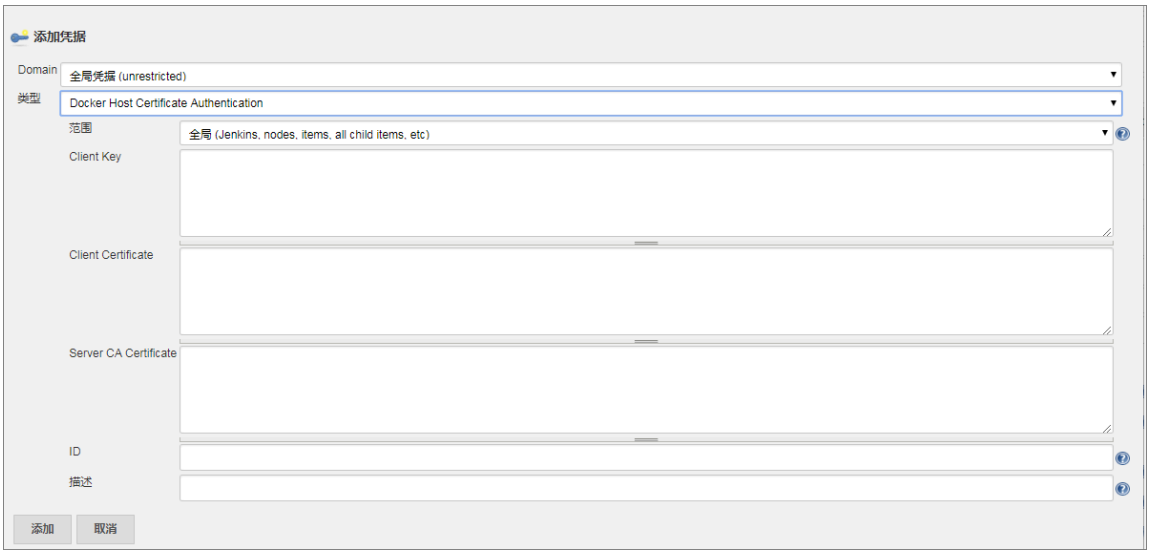


   d. 单击凭证右侧的Add。



   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：

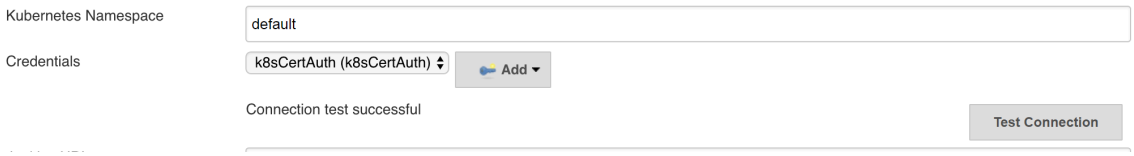


- 类型：选择Docker Host Certificate Authentication。
- Client Key: 填写KubeConfig中client-key-data对应的内容。
- Client Certificate: 填写KubeConfig中client-certificate-data对应的内容。
- ID：填写证书ID。本示例中填写k8sCertAuth。
- 描述：填写描述。

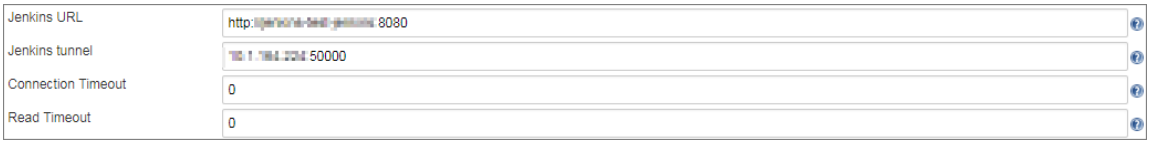
e. 单击添加。

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
```

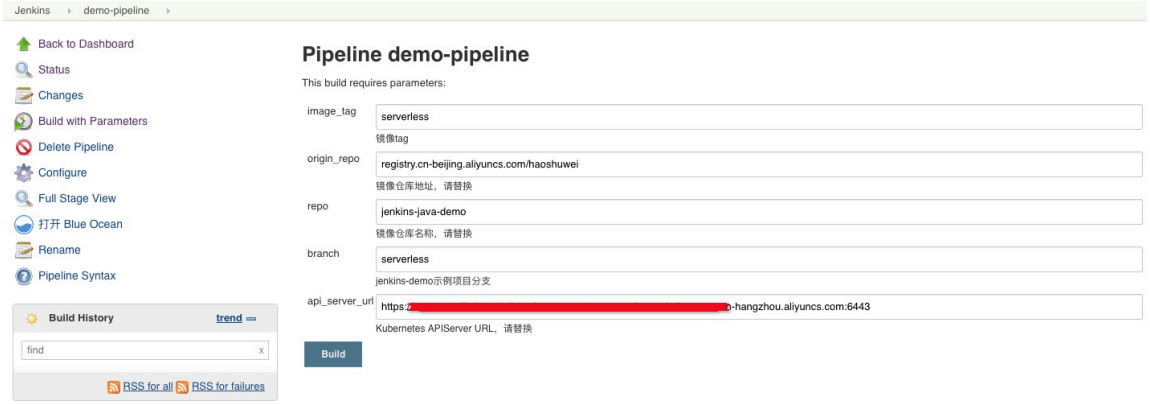
3. 构建demo-pipeline并访问应用服务。

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/jenkins-java-demo:serverless，请根据提示信息进行替换。



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。

9.5 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

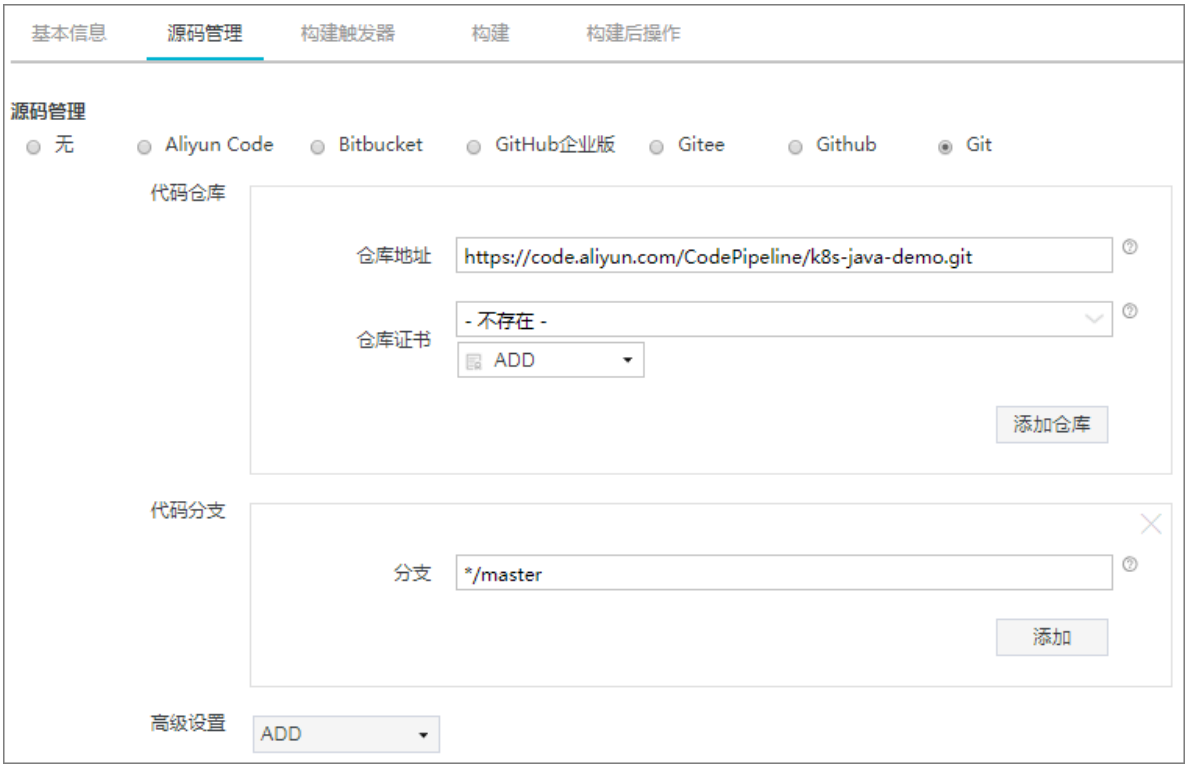
构建java-demo项目并部署到Kubernetes集群。

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`。



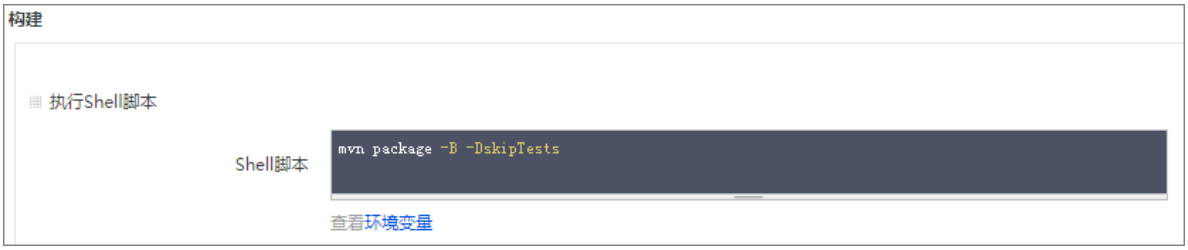
说明：

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`
```

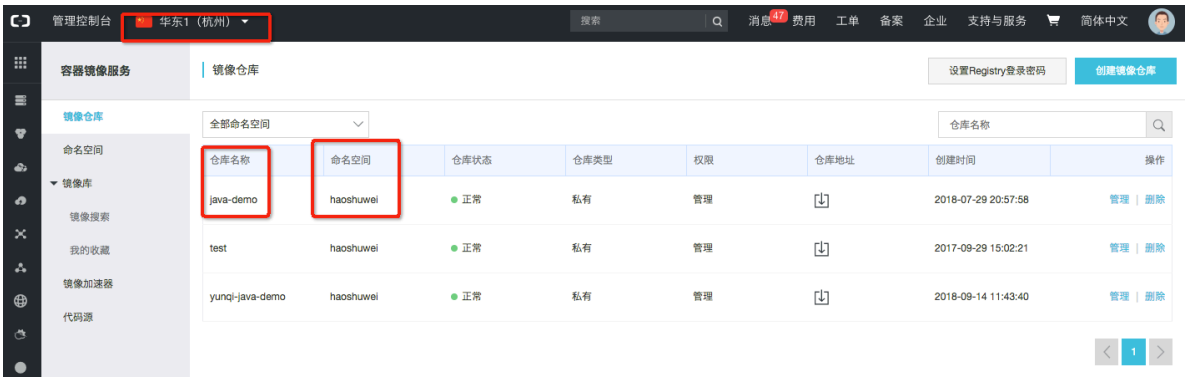
```
echo IMAGE_TAG=$TIME >> env.properties
```

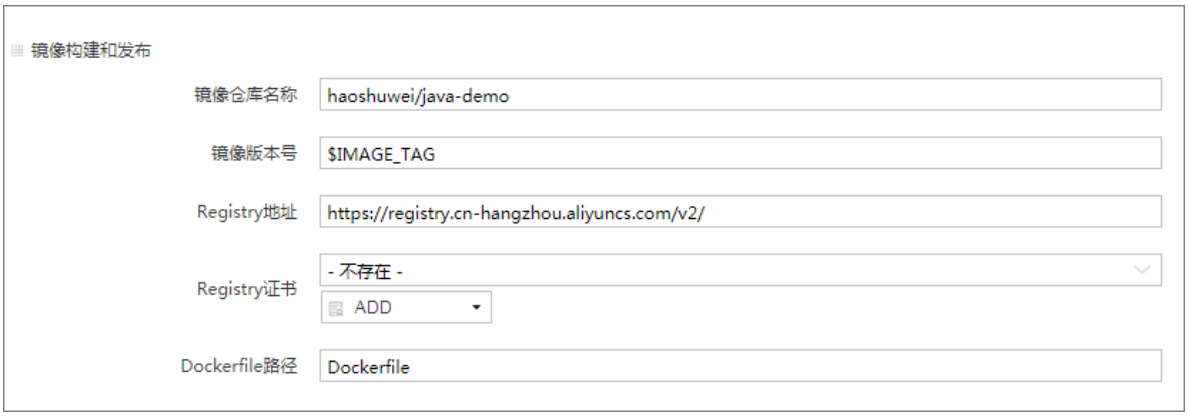
b. 在增加构建步骤下拉框中选择注入环境变量，并填写变量文件路径。

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。





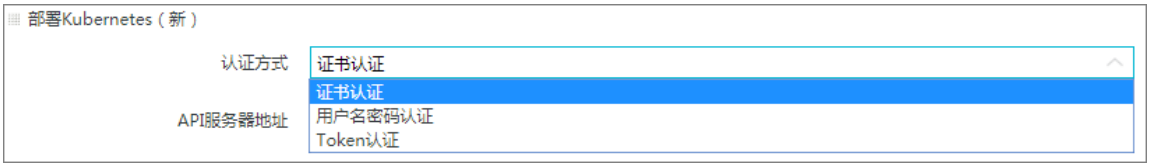
- 镜像仓库名称：填写镜像仓库名称，格式为命名空间/仓库名称。本示例中，填写haoshuwei/java-demo。
- 镜像版本号：填写镜像版本号，例如$IMAGE_TAG。
- Registry地址：填写Registry地址，例如杭州区域的Registry地址`https://registry.cn-hangzhou.aliyuncs.com/v2/`。
- Registry证书：创建Registry类型的证书。
- Dockerfile路径：填写Dockerfile路径。该路径必须位于源码项目的根目录下。

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。

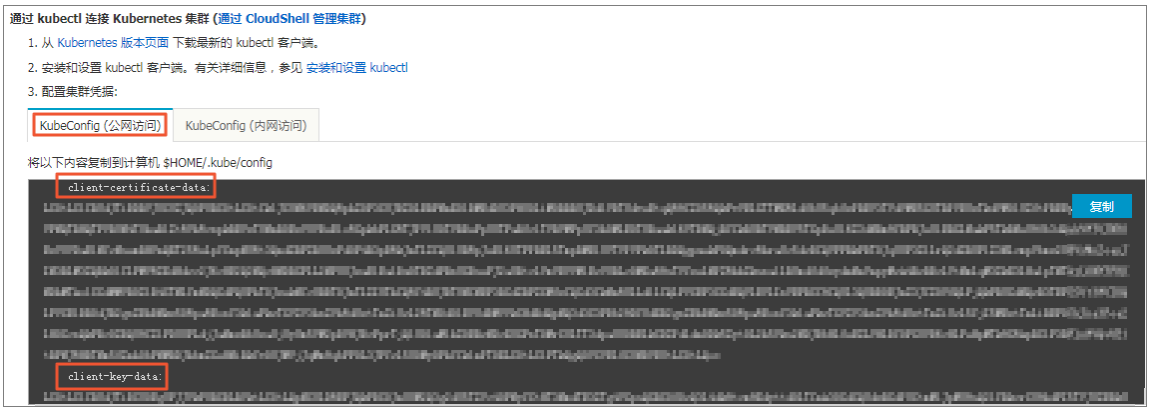


b. 填写API服务器地址。

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。

状态检查配置	namespace:default deployments:java-demo

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
```

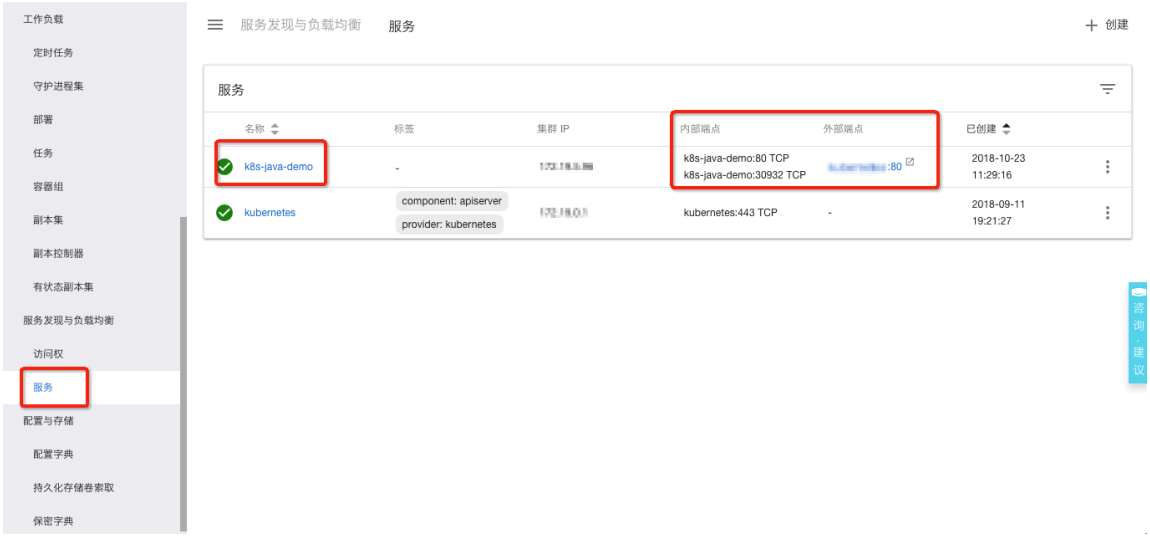
12.在Kubernetes集群控制台查看并访问服务。

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。

9.6 Kubernetes基于云效的DevOps实践

9.6.1 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。

9.6.2 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#unique_11。

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。

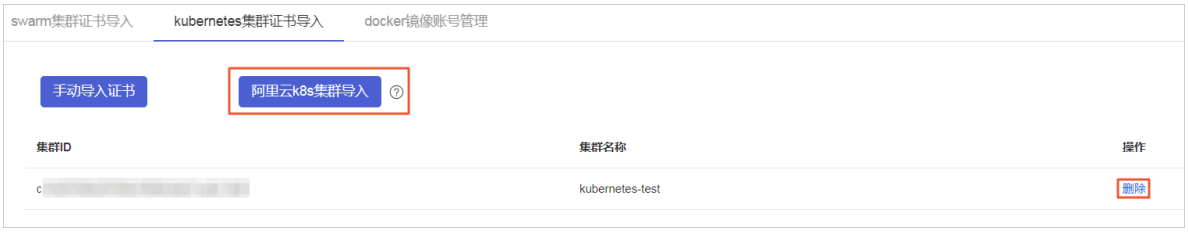
手动导入证书

- 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。



9.6.3 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。

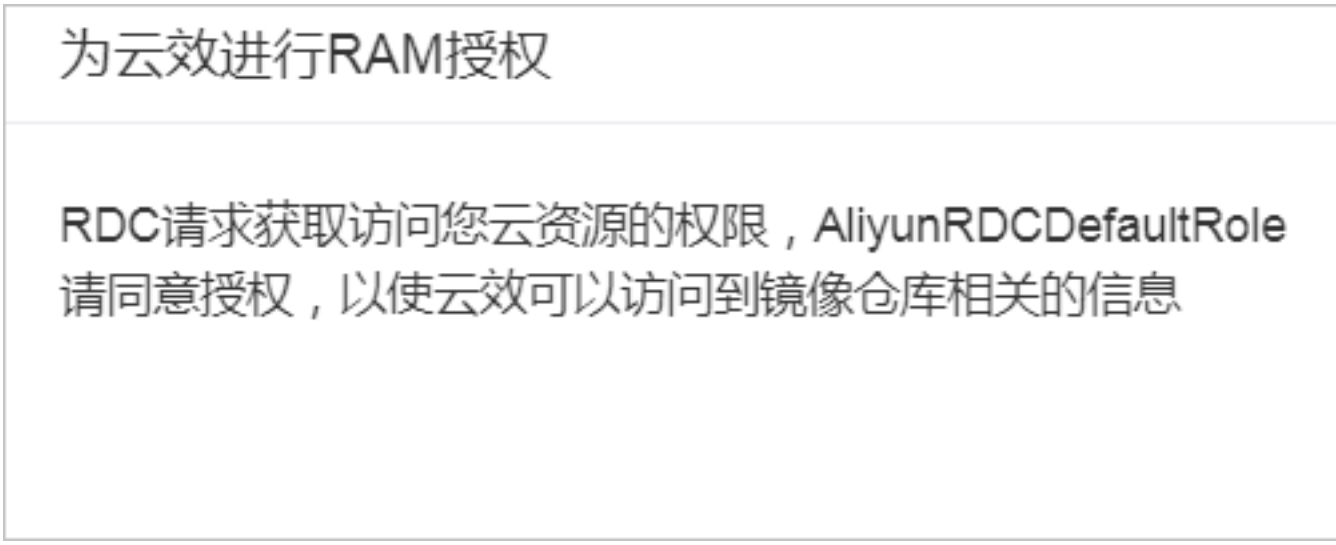
操作步骤

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。



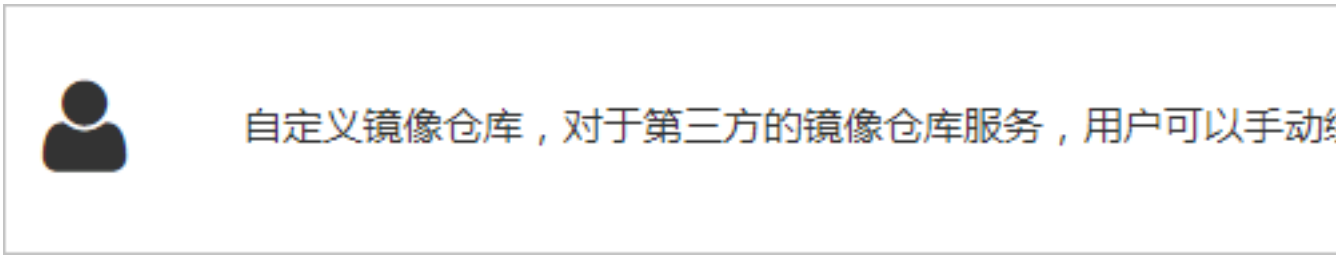
5. 单击RAM账号授权区域的已授权，点击完成绑定，进入绑定账号页面。

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。

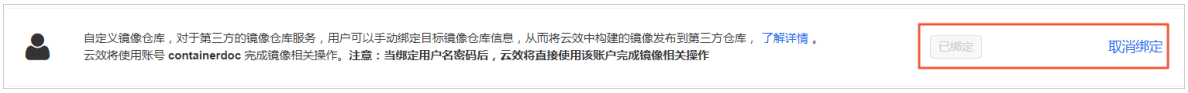
说明:

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。



9.6.4 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。

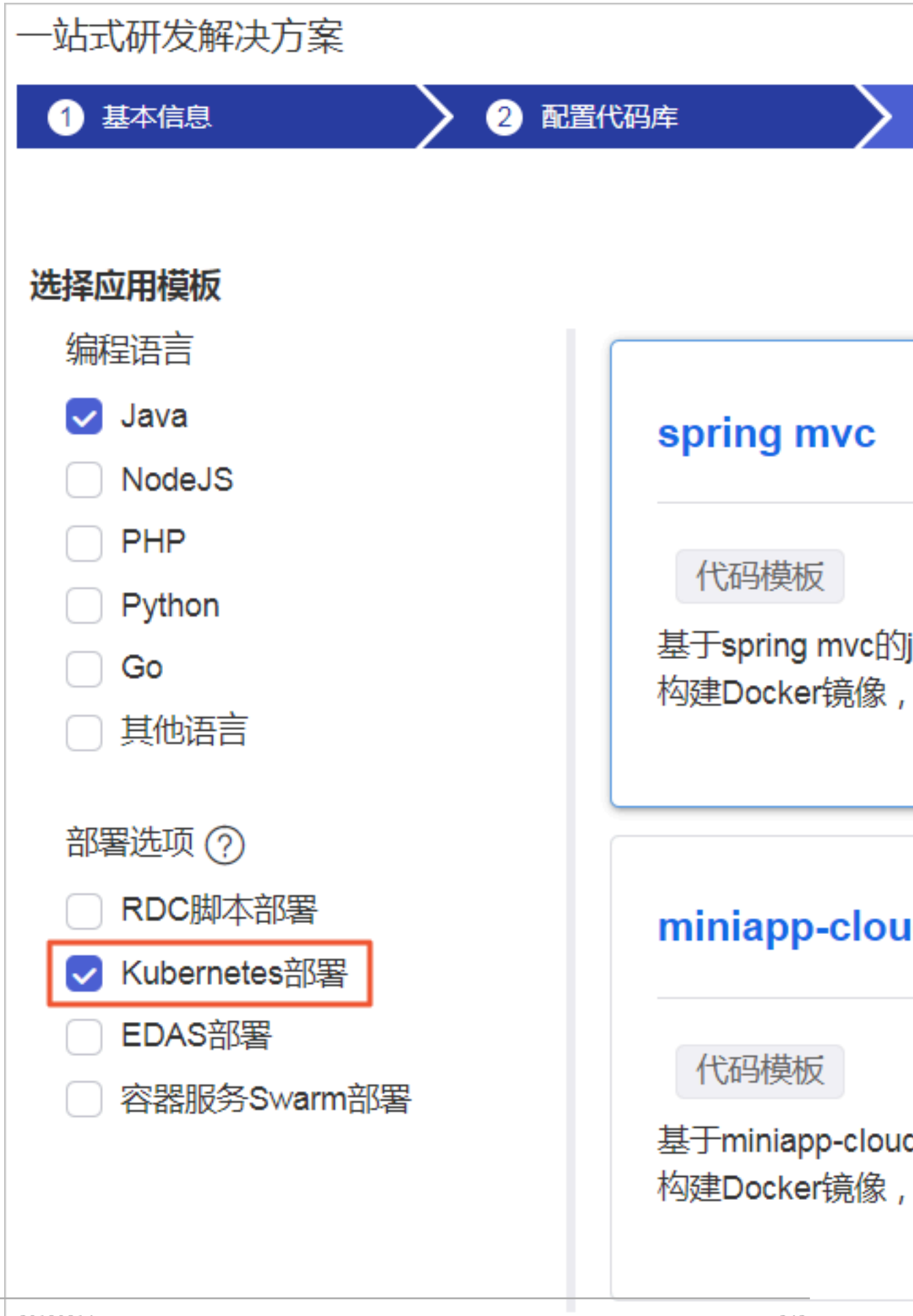
说明:

> 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
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。

9.6.5 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

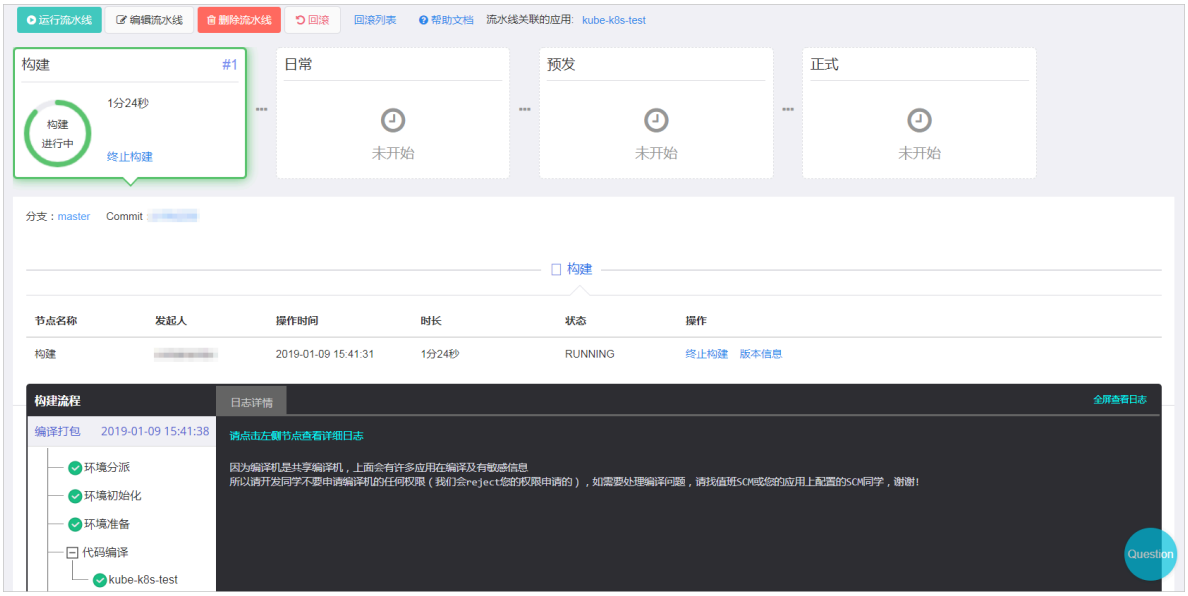
云效管理控制台会为应用创建一个持续交付的流水线。

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。

9.6.6 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。

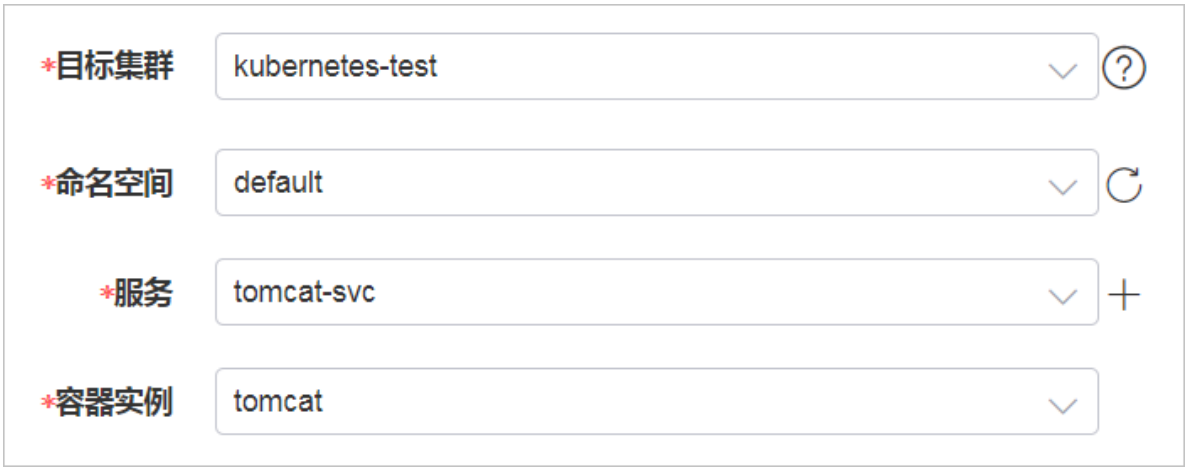
4. 在环境列表页面，单击日常环境的部署配置。

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。

9.6.7 运行流水线

本文主要介绍如何运行流水线。

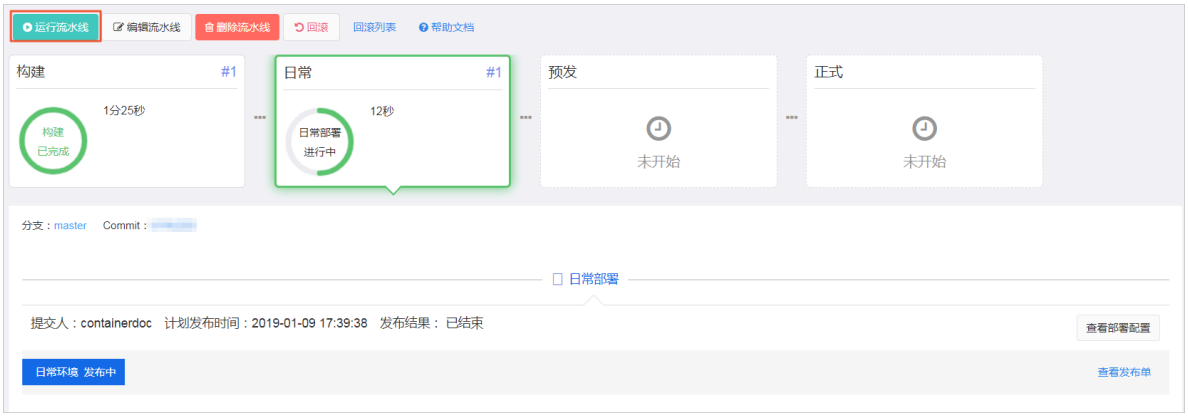
背景信息

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。

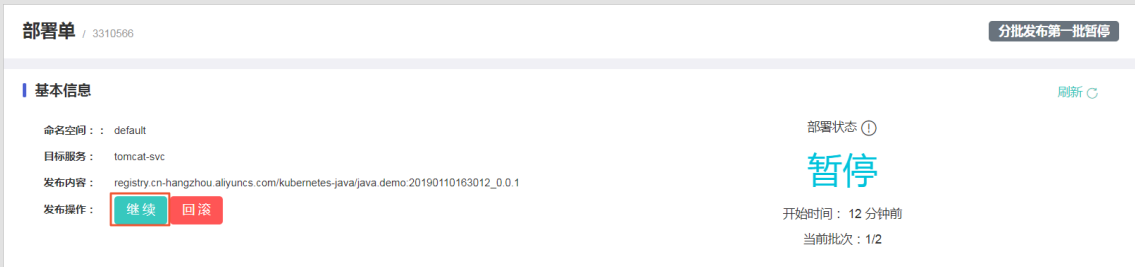
当环境关联的Service存在关联的Deployment资源时：

- 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。

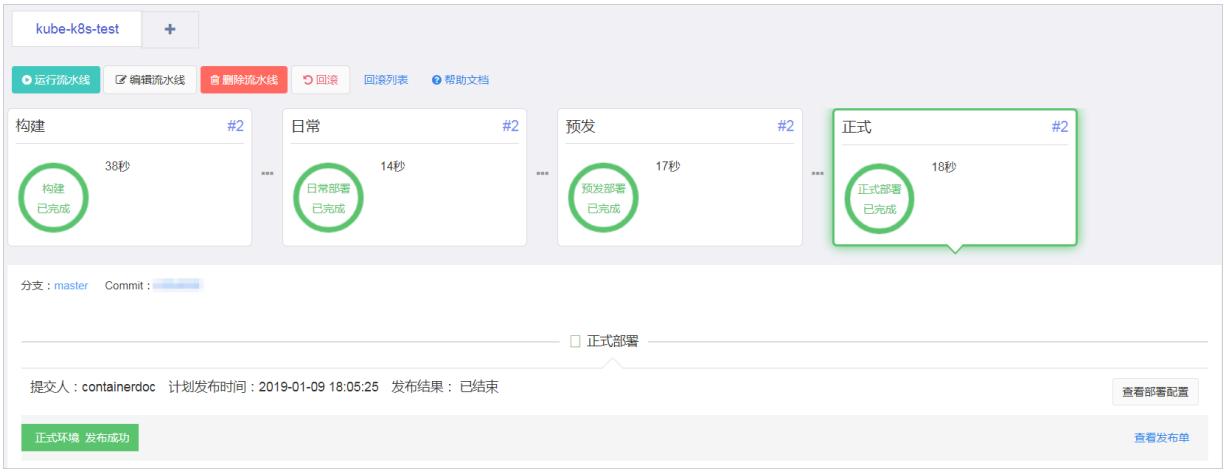
  在部署单页面，单击基本信息区域的继续。



- 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。



9.7 使用Bamboo在阿里云Kubernetes集群中部署Bamboo Agent并执行构建任务

本文档以构建一个 Java 软件项目并部署到阿里云容器服务的 Kubernetes 集群为例，为您介绍如何使用 Bamboo在阿里云Kubernetes服务上运行Remote Agents并在agents上运行Build Plans。

前提条件

- 您已经成功创建一个 Kubernetes 集群，参见创建Kubernetes集群。
- 您已经创建好一个Bomboo Server。

源码项目

本示例中创建的GitHub源码项目地址为如下：

```
https://github.com/AliyunContainerService/jenkins-demo.git
```

分支为`bamboo`。

在Kubernetes中部署Remote Agent

1. 创建kaniko-docker-cfg secret。

   在Remote Agent上构建任务并且使用kaniko推送容器镜像时，kaniko-docker-cfg secret用于的配置镜像仓库访问权限。

   a. 在Linux服务器上使用root用户执行如下命令，生成`/root/.docker/config.json`。

   ```
   docker login registry.cn-hangzhou.aliyuncs.com
   ```

   b. 通过CloudShell 连接 Kubernetes 集群，并执行如下命令，创建kaniko-docker-cfg secret。

   ```
   kubectl -n bamboo create secret generic kaniko-docker-cfg  --from-file=/root/.docker/config.json
   ```

2. 创建Bamboo Agent 应用。

   创建serviceaccount bamboo以及clusterrolebinding，用于kubectl部署应用到kubernetes集群的权限设置。

   a. 创建并拷贝以下内容到*bamboo-agent.yaml*，并执行`kubectl -n bamboo apply -f bamboo-agent.yaml`命令，创建Bamboo Agent 应用。

   ```
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     namespace: bamboo
     name: bamboo

   ---
   apiVersion: rbac.authorization.k8s.io/v1
   kind: ClusterRoleBinding
   metadata:
     name: bamboo-cluster-admin
   subjects:
     - kind: ServiceAccount
       name: bamboo
       namespace: bamboo
   roleRef:
     kind: ClusterRole
     name: cluster-admin
     apiGroup: rbac.authorization.k8s.io

   ---
   apiVersion: apps/v1beta2
   kind: Deployment
   metadata:
     name: bamboo-agent
   spec:
     replicas: 1
     selector:
       matchLabels:
         app: bamboo-agent
     template:
   ```

```
    metadata:
      labels:
        app: bamboo-agent
    spec:
      serviceAccountName: bamboo
      containers:
      - name: bamboo-agent
        env:
        - name: BAMBOO_SERVER_URL
          value: http://xx.xx.xx.xx:8085
        image: registry.cn-hangzhou.aliyuncs.com/haoshuwei/docker-
bamboo-agent:v1
        imagePullPolicy: Always
        volumeMounts:
          - mountPath: /root/.docker/
            name: kaniko-docker-cfg
      volumes:
        - name: kaniko-docker-cfg
          secret:
            secretName: kaniko-docker-cfg
```

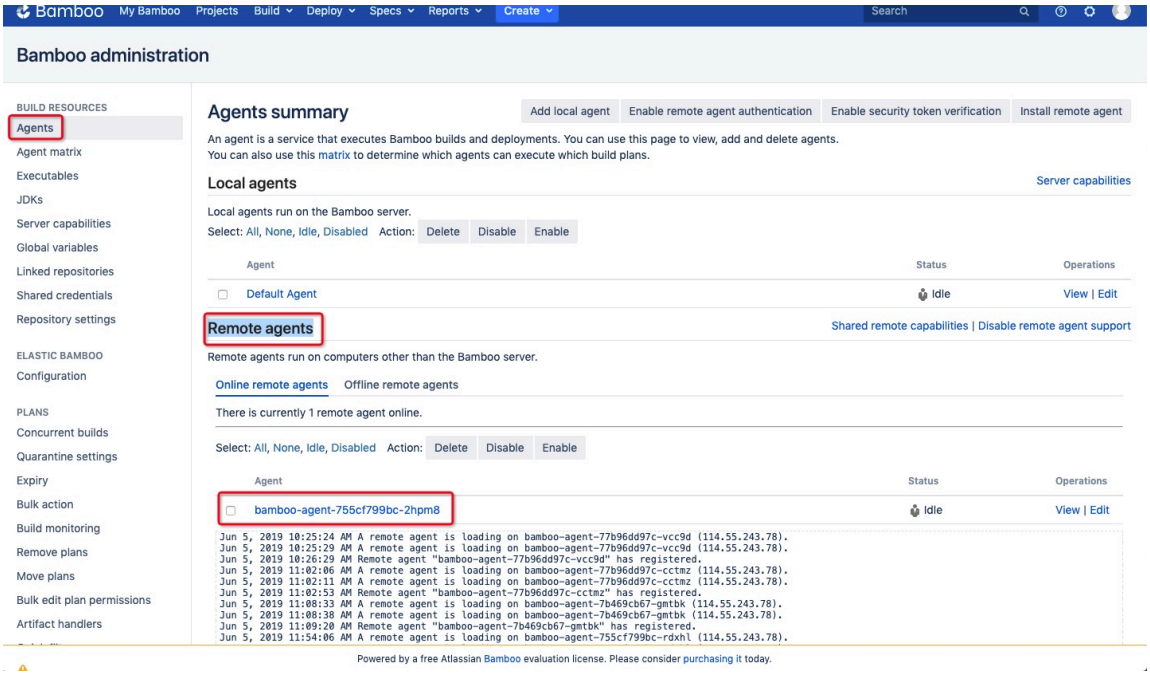
b. 完成后，执行如下命令查看日志。

```
kubectl -n bamboo logs -f <bamboo agent pod name>
```

说明:

示例中*bamboo agent pod name*需要替换成实际的文件名。

c. Remote Agent注册成功后，登录Bamboo。单击右上角 ，在下拉框中选择Agent，在Bamboo administration页面，可以看到部署好的Agent。



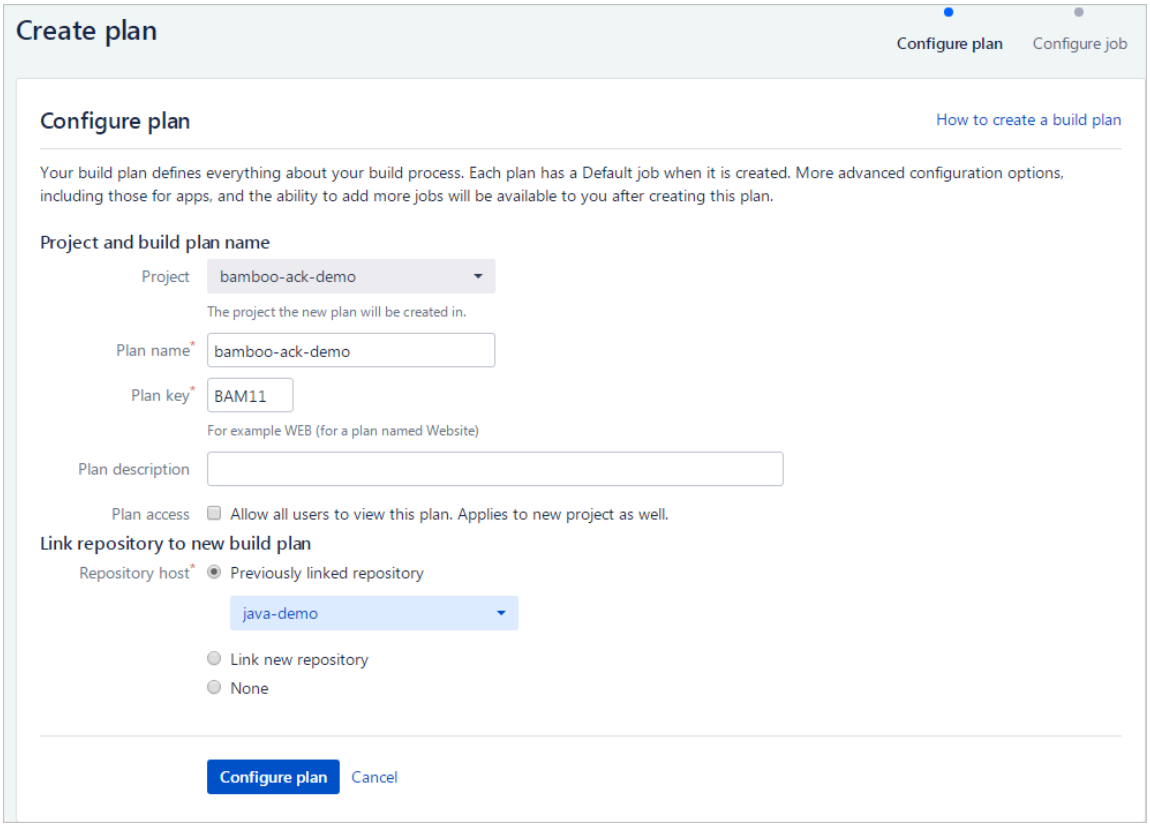
配置一个Build plan

完成应用源码拉取、编译打包、容器镜像打包和推送、应用部署的过程。

1. 创建一个Build plan。

   a. 登录Bamboo，选择Create > Create plan，进入Configure plan页面。

   b. Project选择`bamboo-ack-demo`，填写`Plan name`、`Plan key`和`Plan description`，`Repository host`选择`java-demo`，完成后单击Configure plan。

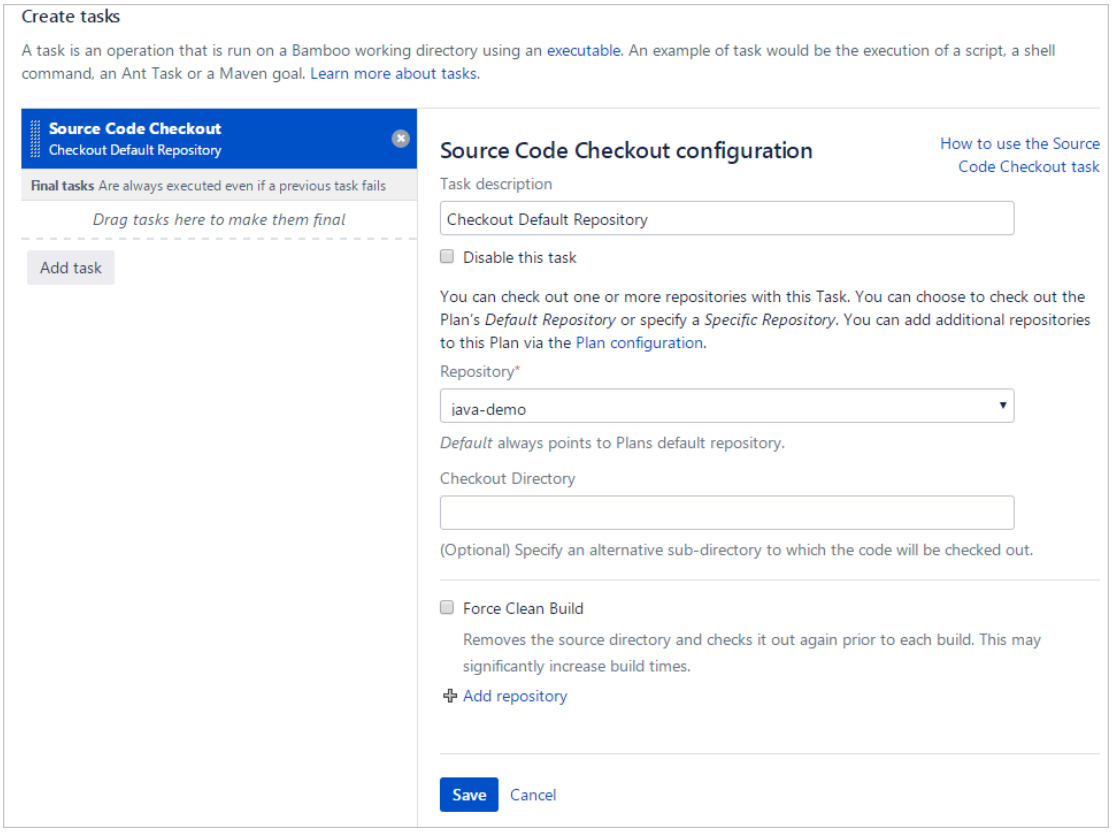


2. 进入Configure Job页面，配置Stages并添加Job。

a. 源码拉取

在创建Bulid plan时，参数`Repository host`选择`java-demo`，即已完成源码拉取，您可以通过如下操作，修改源码拉取位置。

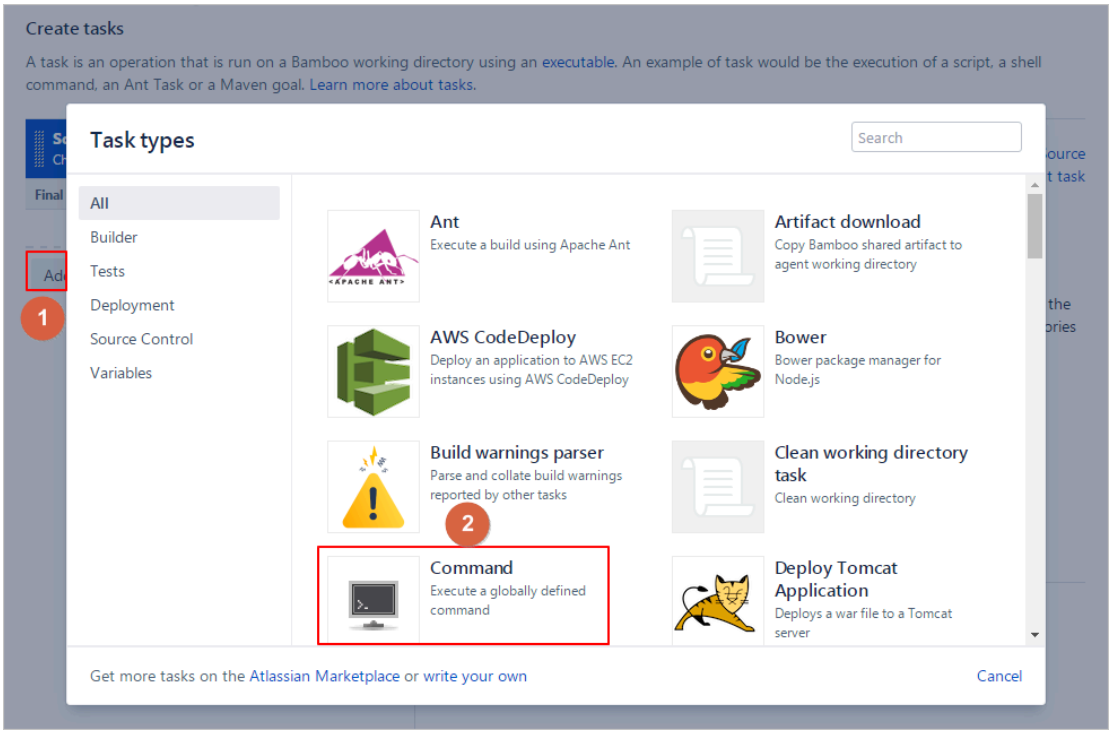
A. 在Create tasks区域，单击Source Code Checkout。



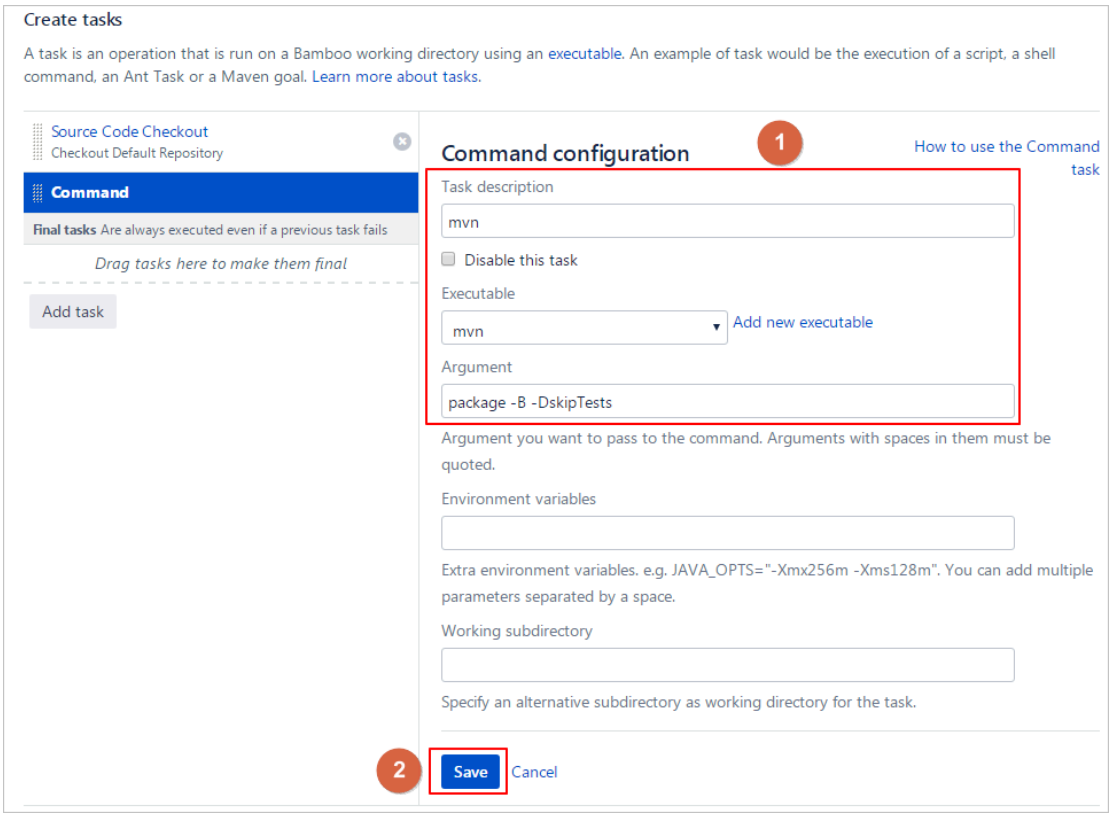
B. 在右侧的Source Code Checkout configuration页面，修改参数`Repository host`的取值，单击Save。

b. mvn打包

A. 在Create tasks区域，单击Add task，在弹出的Task types页面选择Command。

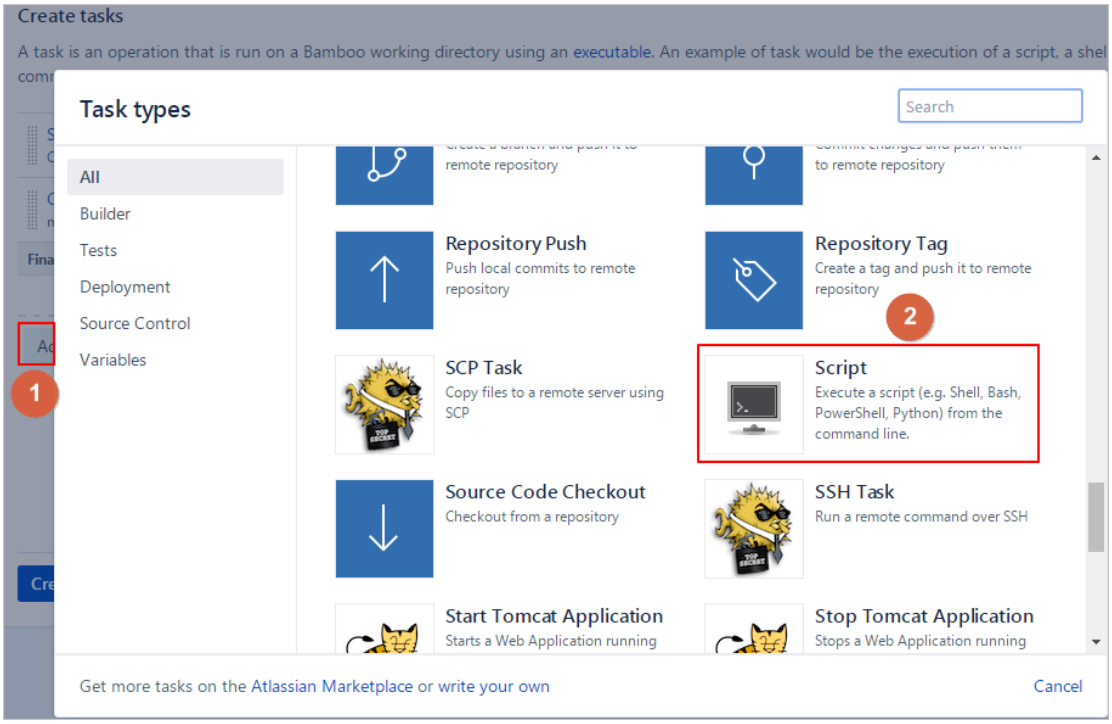


B. 在右侧弹出Command configuration页面，填写Task description、Executable和Argument后，单击Save。

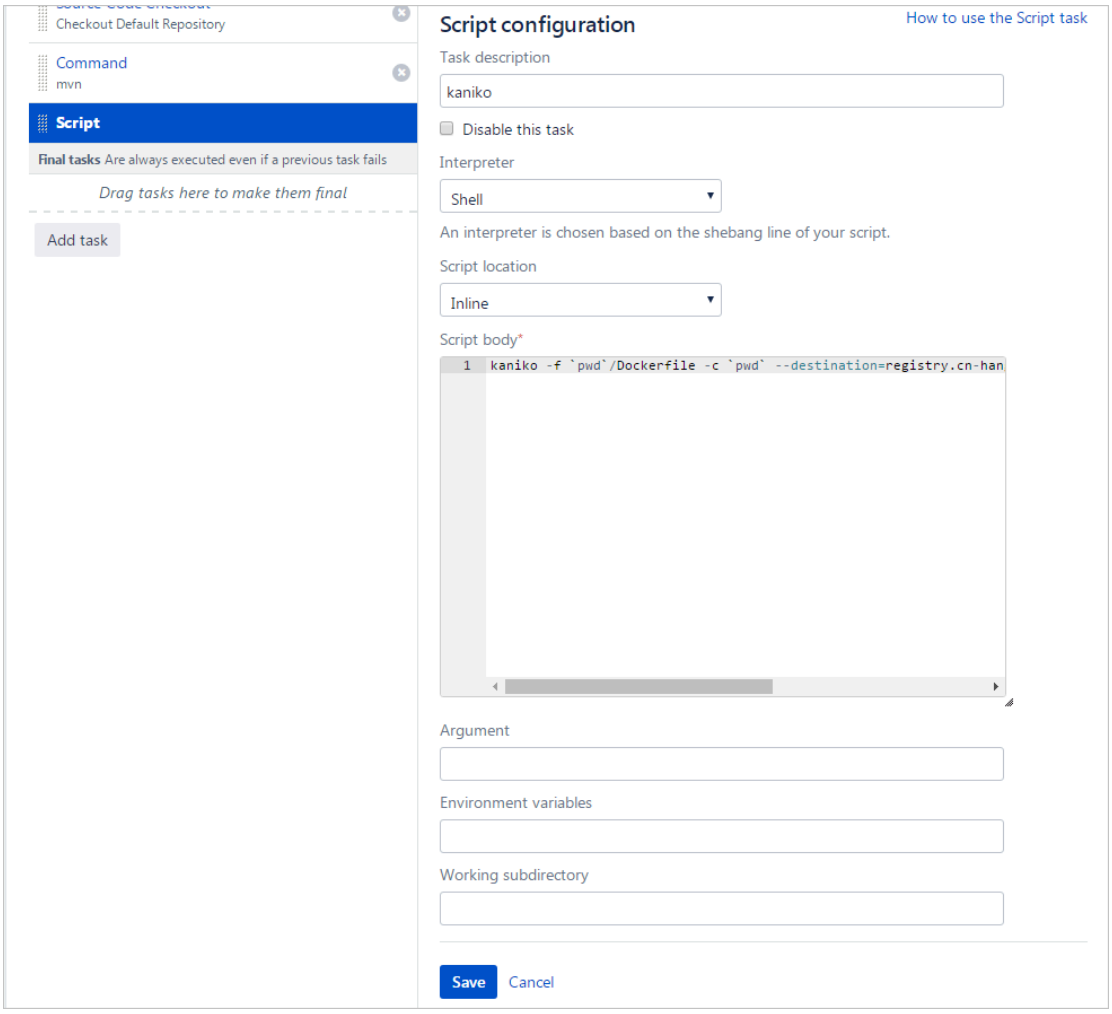


c. kaniko打包和推送容器镜像

A. 在Create tasks区域，单击Add task，在弹出的Task types页面选择Script。



B. 在右侧弹出Script configuration页面，填写Task description、Script location后（其他参数保持默认值即可），单击Save。

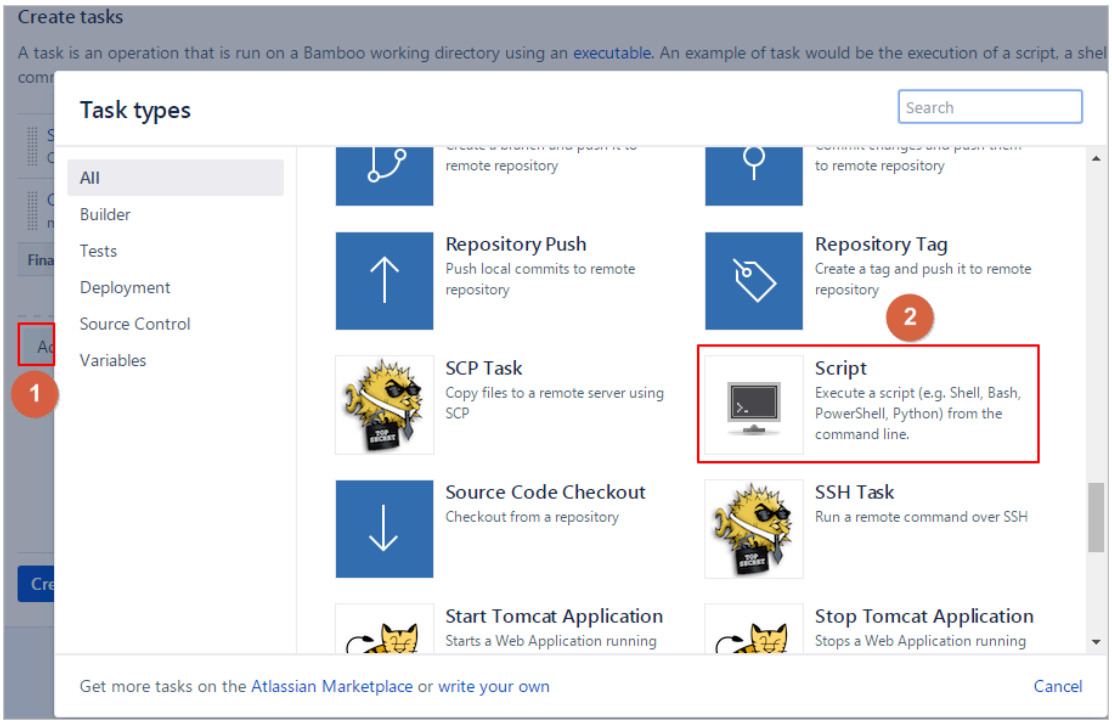


本示例中，Script location如下：

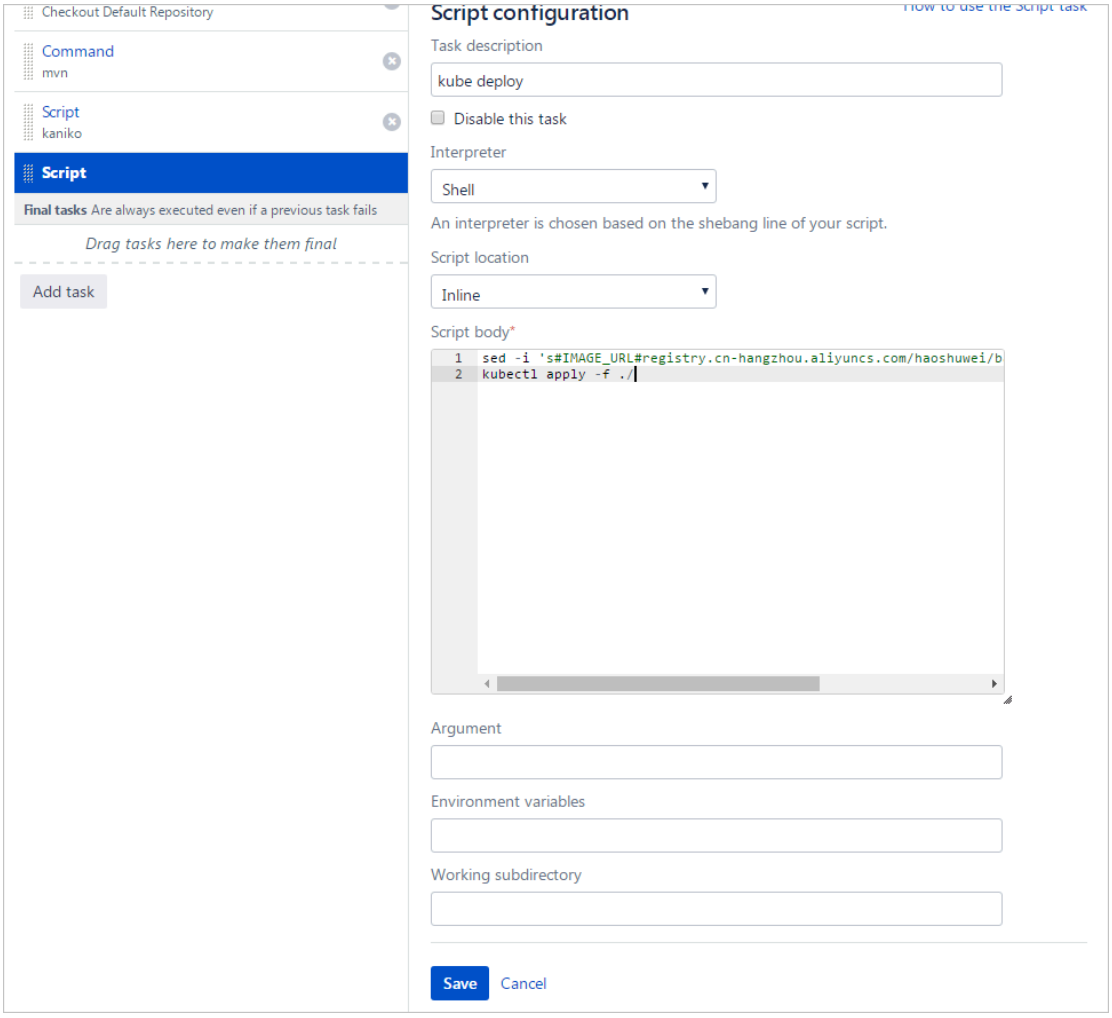
```
kaniko -f `pwd`/Dockerfile -c `pwd` --destination=registry.cn-hangzhou.aliyuncs.com/haoshuwei/bamboo-java-demo:latest
```

d. Kubectl 部署应用到 Kubernetes

A. 在Create tasks区域，单击Add task，在弹出的Task types页面选择Script。



B. 在右侧弹出Script configuration页面，填写Task description、Script location后（其他参数保持默认值即可），单击Save。



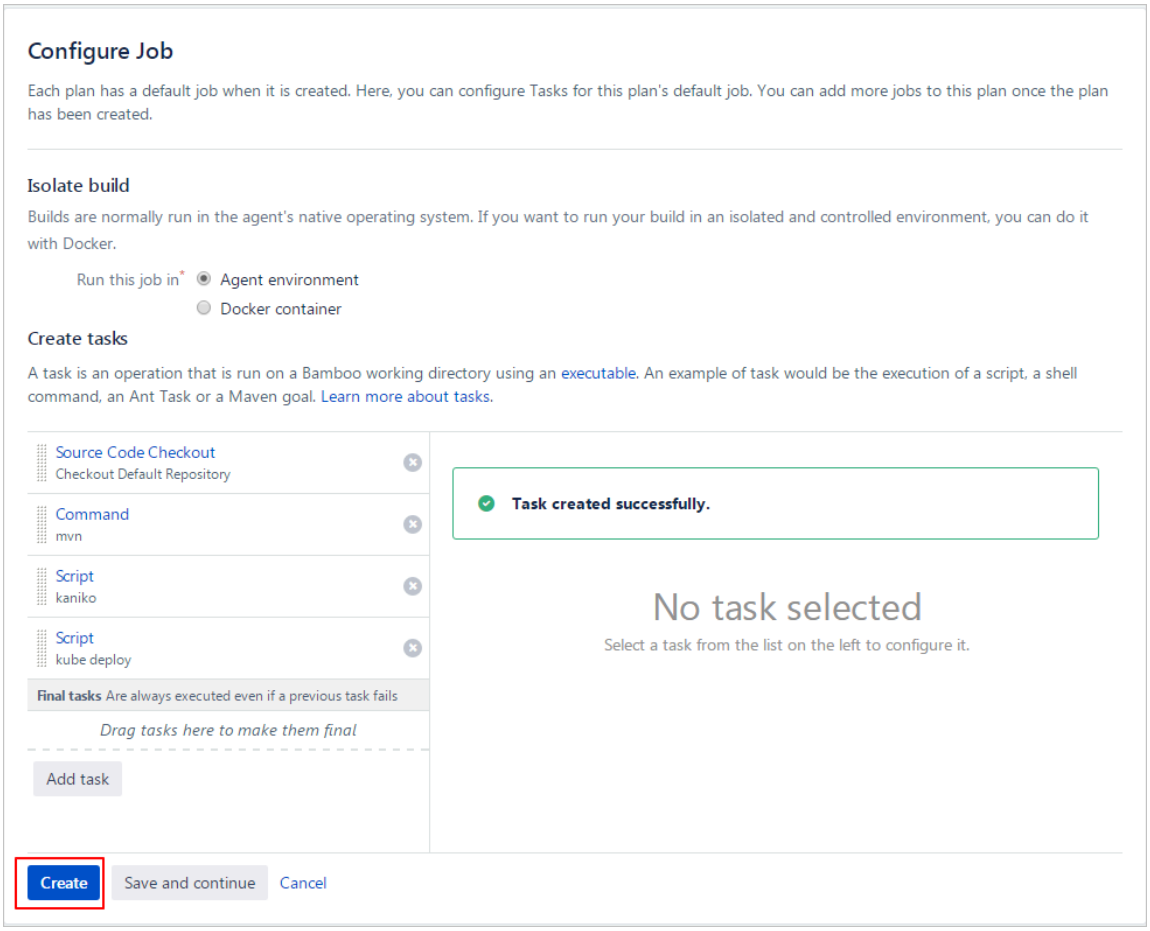
本示例中，Script location如下：

```
sed -i 's#IMAGE_URL#registry.cn-hangzhou.aliyuncs.com/haoshuwei
/bamboo-java-demo:latest#' ./*.yaml
```

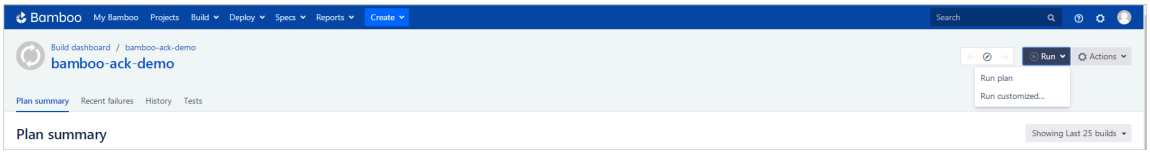
```
kubectl apply -f ./
```

3. 运行 Build。

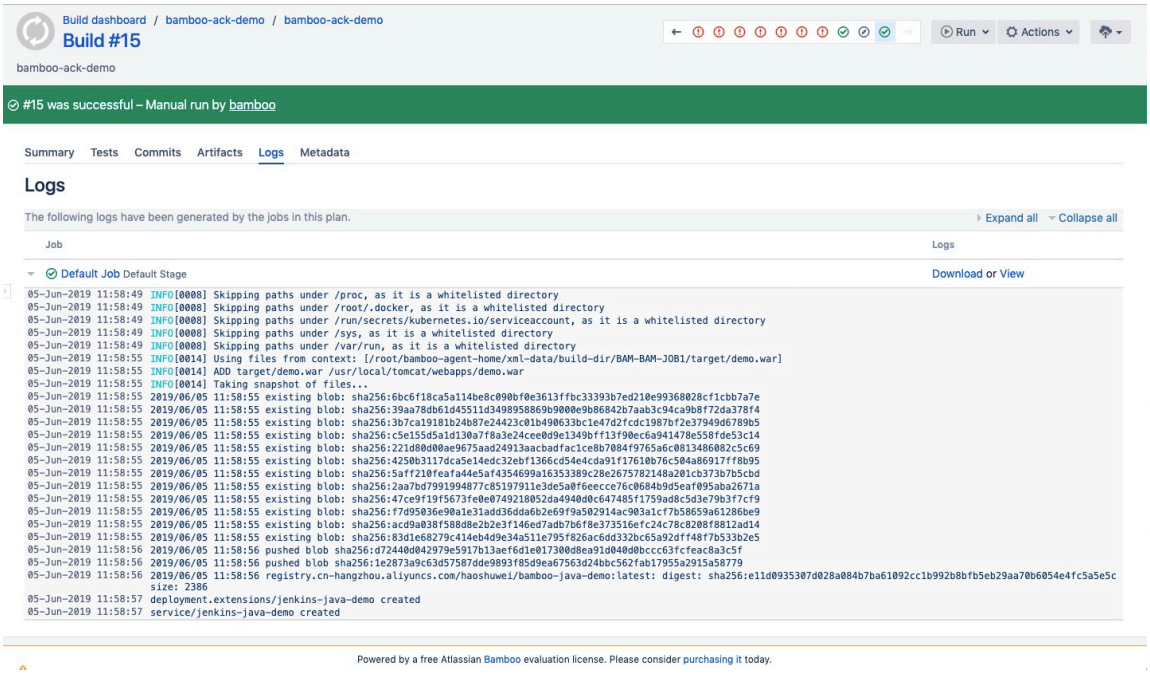
   a. 上述配置完成后，单击Create，进入bamboo-ack-demo页面。



   b. 在右上角选择Run > Run plan，运行Build。



   您可以通过单击Logs页签，可以运行日志。

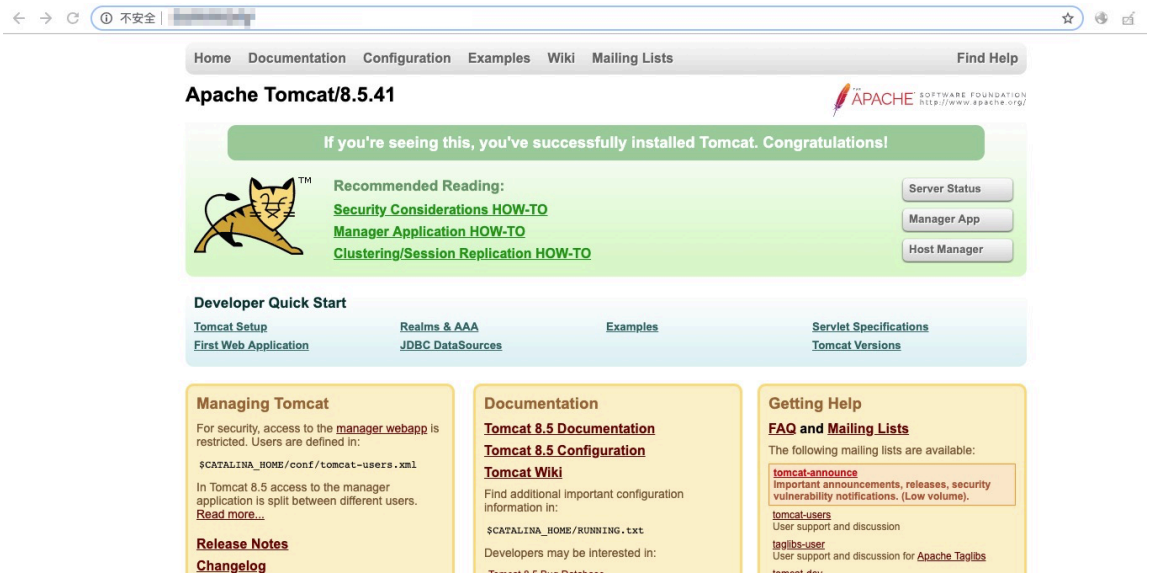


4. 访问应用服务。

   a. 执行`kubectl -n bamboo get svc`命令，查看应用服务。

```
[root@iZbp12i73koztp1cz75skaZ bamboo]# kubectl -n bamboo get svc
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP
   PORT(S)        AGE
jenkins-java-demo   LoadBalancer   xx.xx.xx.xx     xx.xx.xx.xx
   80:32668/TCP   39m
```

   b. 在浏览器中使用*http://EXTERNAL-IP*，访问应用。



相关参考

单击这里查看本示例中镜像registry.cn-hangzhou.aliyuncs.com/haoshuwei/docker-bamboo-agent:v1的制作。

了解更多Bamboo的相关内容，请参见Bamboo。

10 Swarm迁移Kubernetes

10.1 容器服务swarm集群与Kubernetes集群的主要功能比对

10.1.1 概述

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

前提条件

您已经成功创建一个 Kubernetes 集群，参见#unique_11。

说明:

。

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

使用限制

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

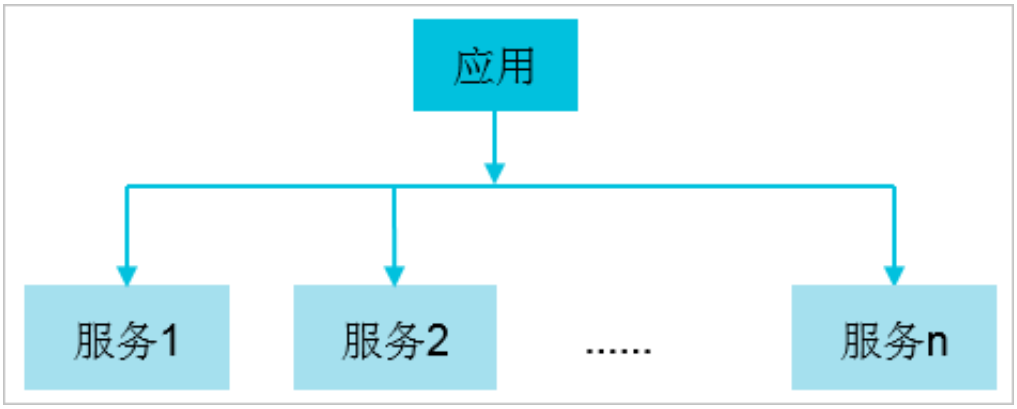
10.1.2 概念比对

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

应用

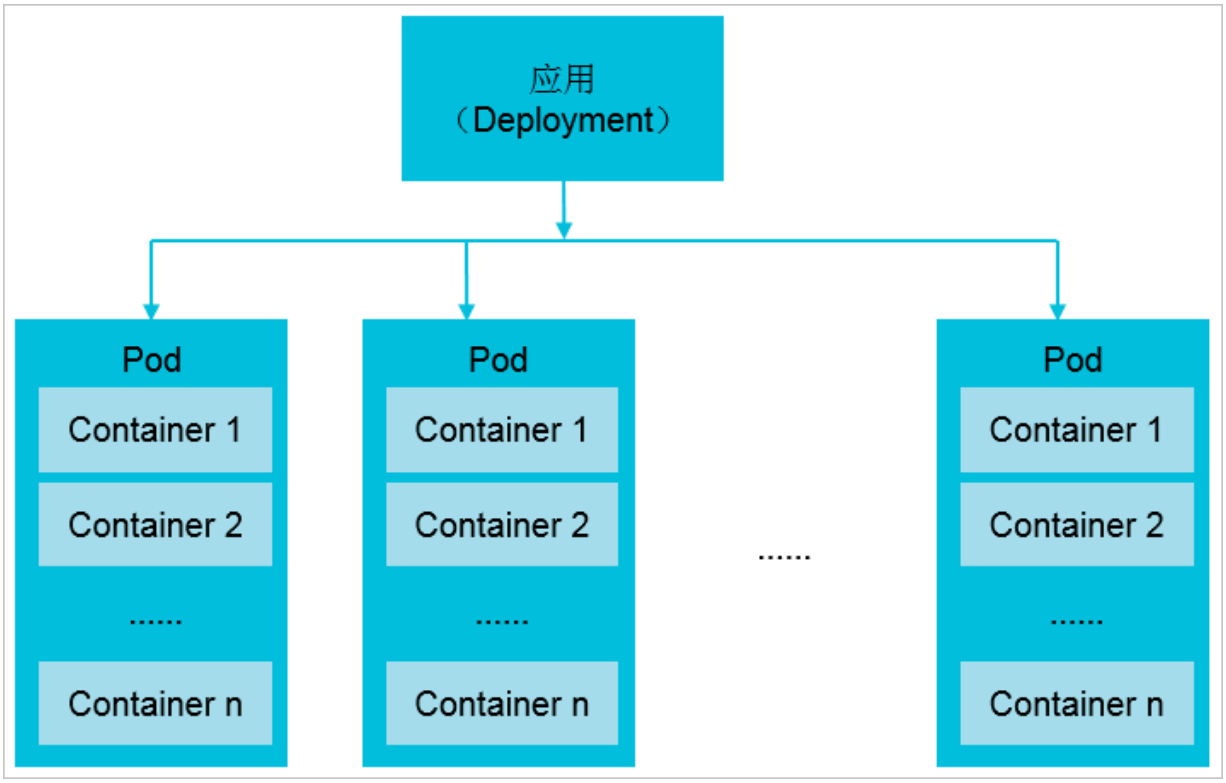
容器服务Swarm集群

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



容器服务Kubernetes集群

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



说明:

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

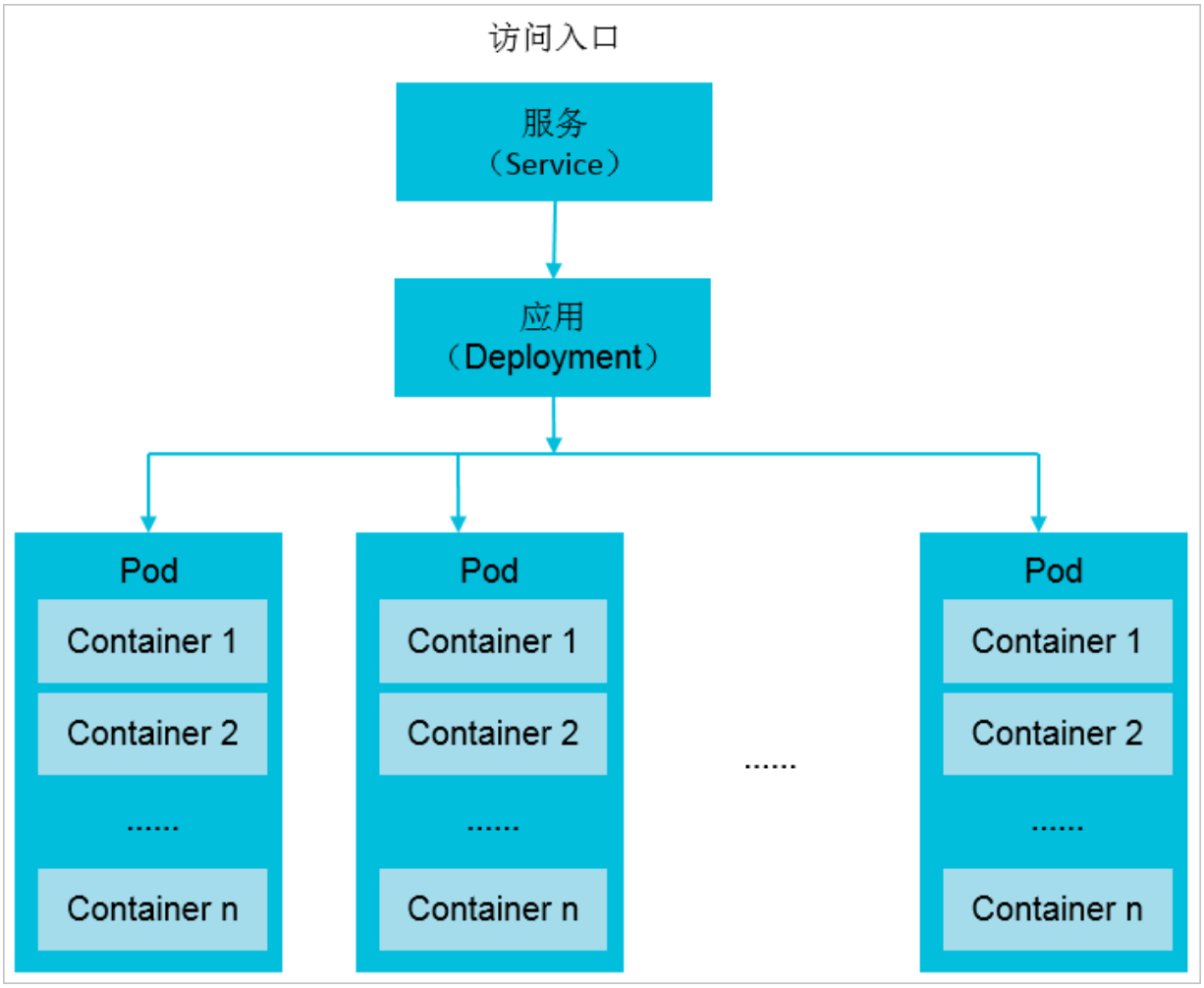
服务

容器服务Swarm集群

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

容器服务Kubernetes集群

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



应用访问

容器服务Swarm集群

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

- <HostIP>:<port>
- 简单路由
- 负载均衡

容器服务Kubernetes集群

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

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

10.1.3 使用镜像创建应用-基本配置比对

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

使用镜像创建应用

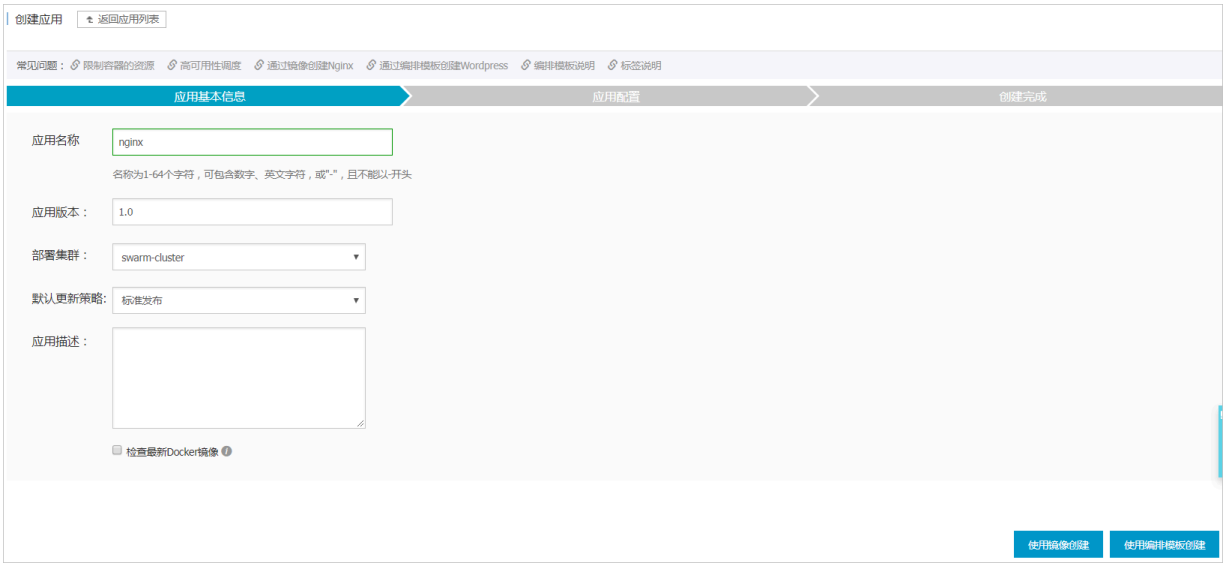
容器服务Swarm与Kubernetes集群在使用镜像创建应用时，部署界面差异较大。

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

应用基本信息

容器服务Swarm集群

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



容器服务Kubernetes集群

与Swarm集群不同的是：需要配置命名空间、副本数量及类型。

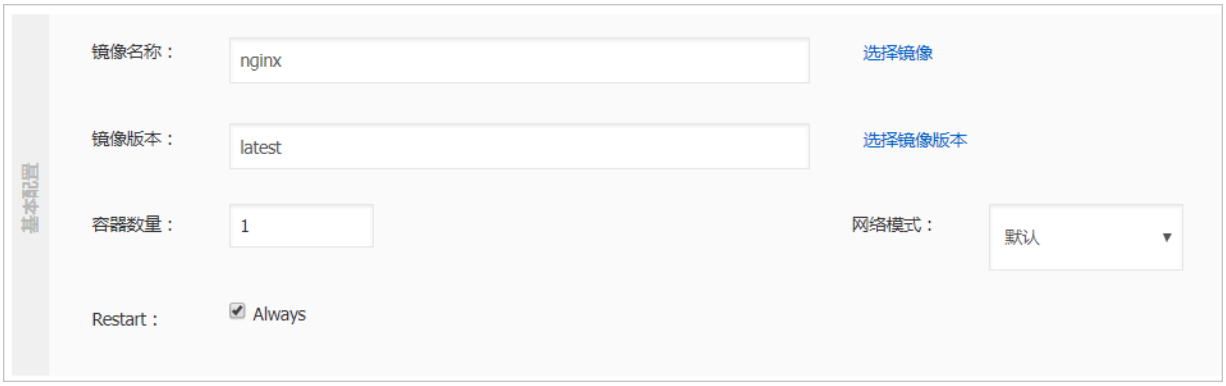
命名空间是容器服务Kubernetes集群特有的概念，Kubernetes通过命名空间（namespace）进行资源的隔离，如CPU等。同时也可以区分不同的使用环境，比如测试环境、开发环境。如果涉及生产环境，建议通过集群隔离。Kubernetes的命名空间，可参考基本概念。

基本配置

基本配置，主要是选择镜像和镜像版本。

容器服务Swarm集群

网络模式目前支持默认和host两种。



容器服务Kubernetes集群

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

10.1.4 使用镜像创建应用-网络配置比对

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

使用镜像创建应用

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

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

网络配置

容器服务Swarm集群的网络配置主要完成应用对外访问方式的暴露。

端口映射

容器服务Swarm集群

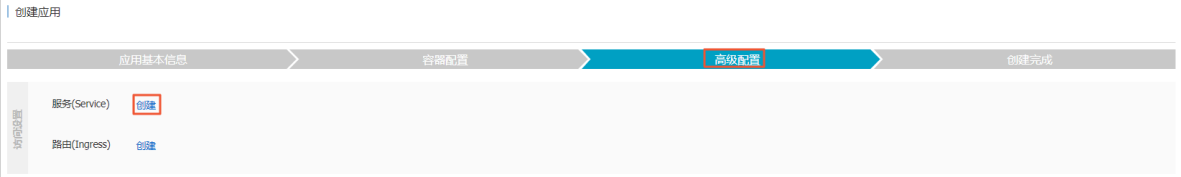
容器服务Swarm集群的端口映射是将应用端口映射到宿主机，在每个宿主机上都会启用相同的端口，这样访问应用的时候只需要`<HostIP>:<Port>`即可访问。

容器服务Kubernetes集群

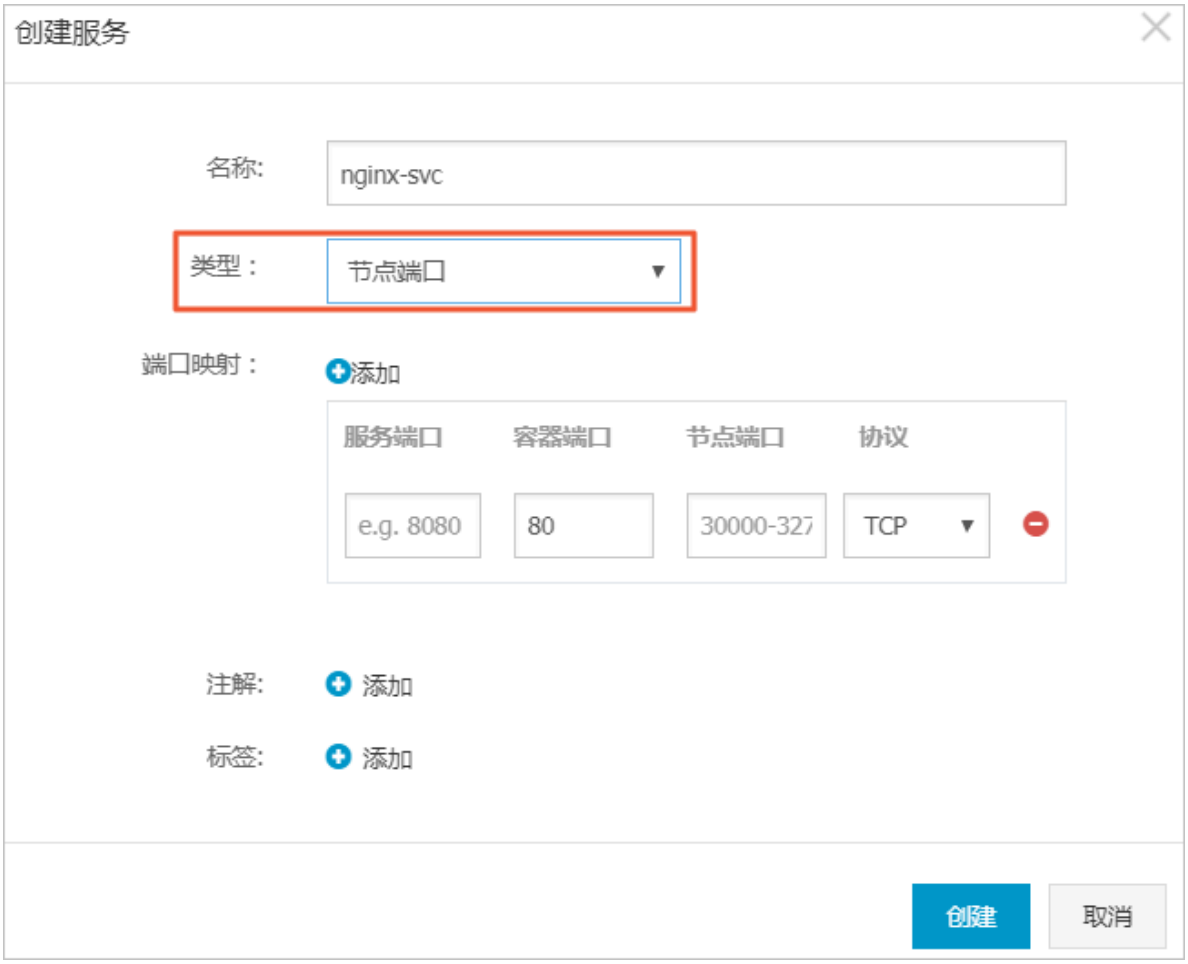
在容器服务Kubernetes集群可以通过NodePort类型的Service来实现，有以下两种方法：

方法一：创建应用时配置

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



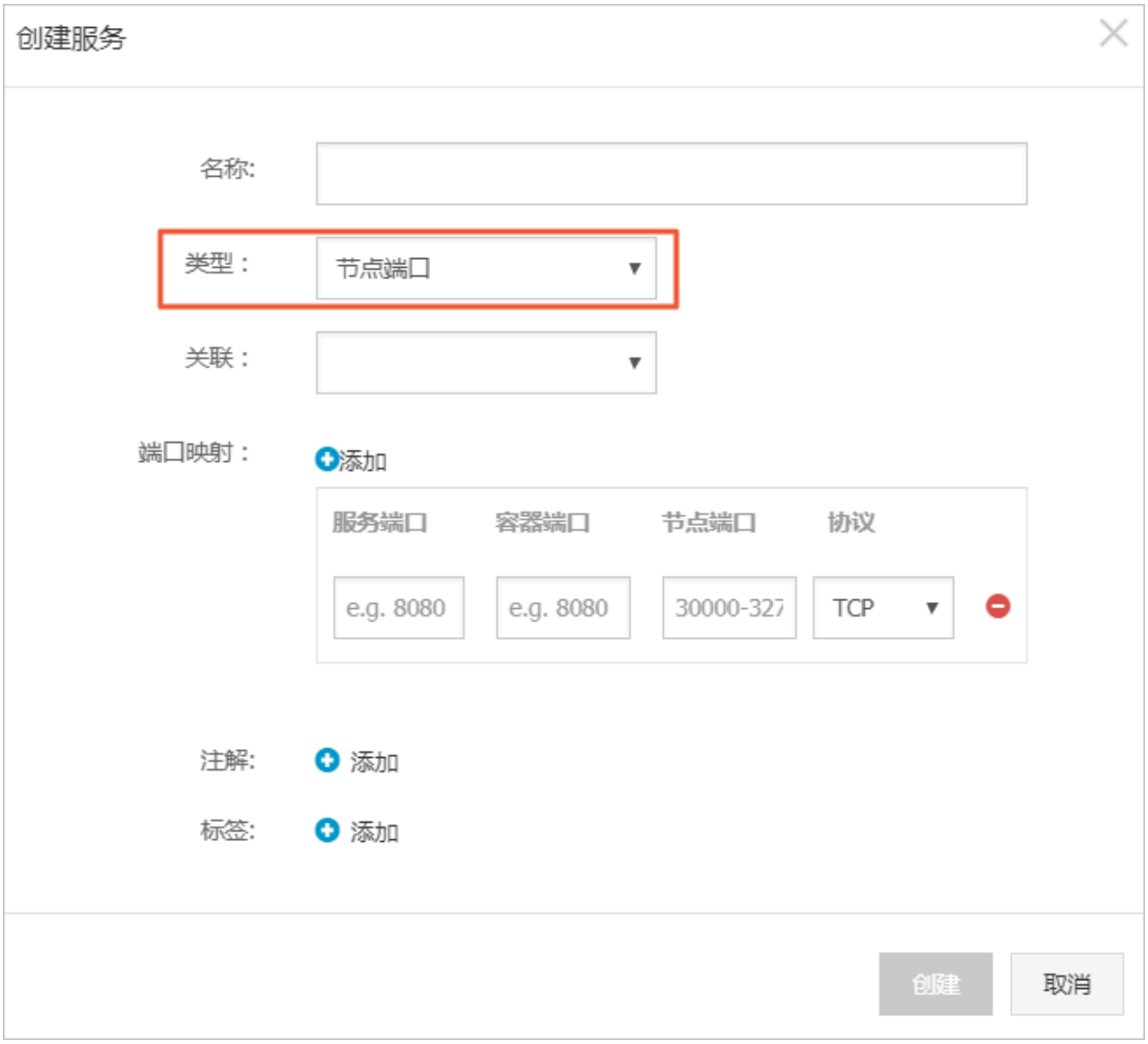
2. 类型请选择节点端口。详细信息可参考镜像创建无状态Deployment应用。



方法二：通过创建服务配置

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

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



简单路由配置

容器服务Swarm集群

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

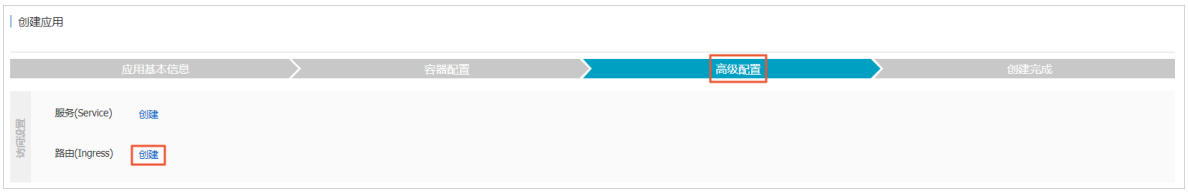
容器服务Kubernetes集群

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

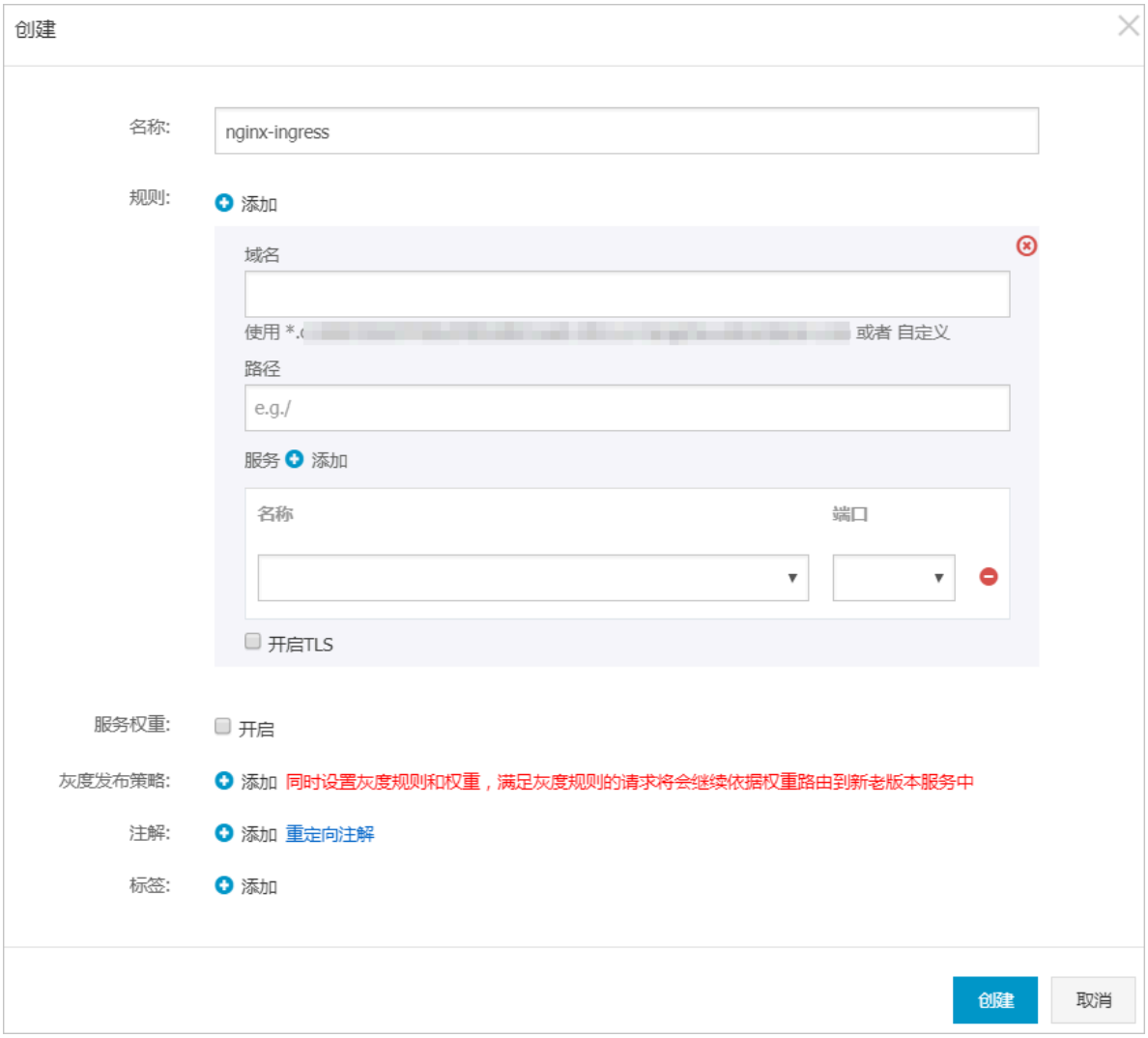
容器服务Kubernetes集群实现路由（Ingress）功能，有两种方法：

方法一：创建应用时配置

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



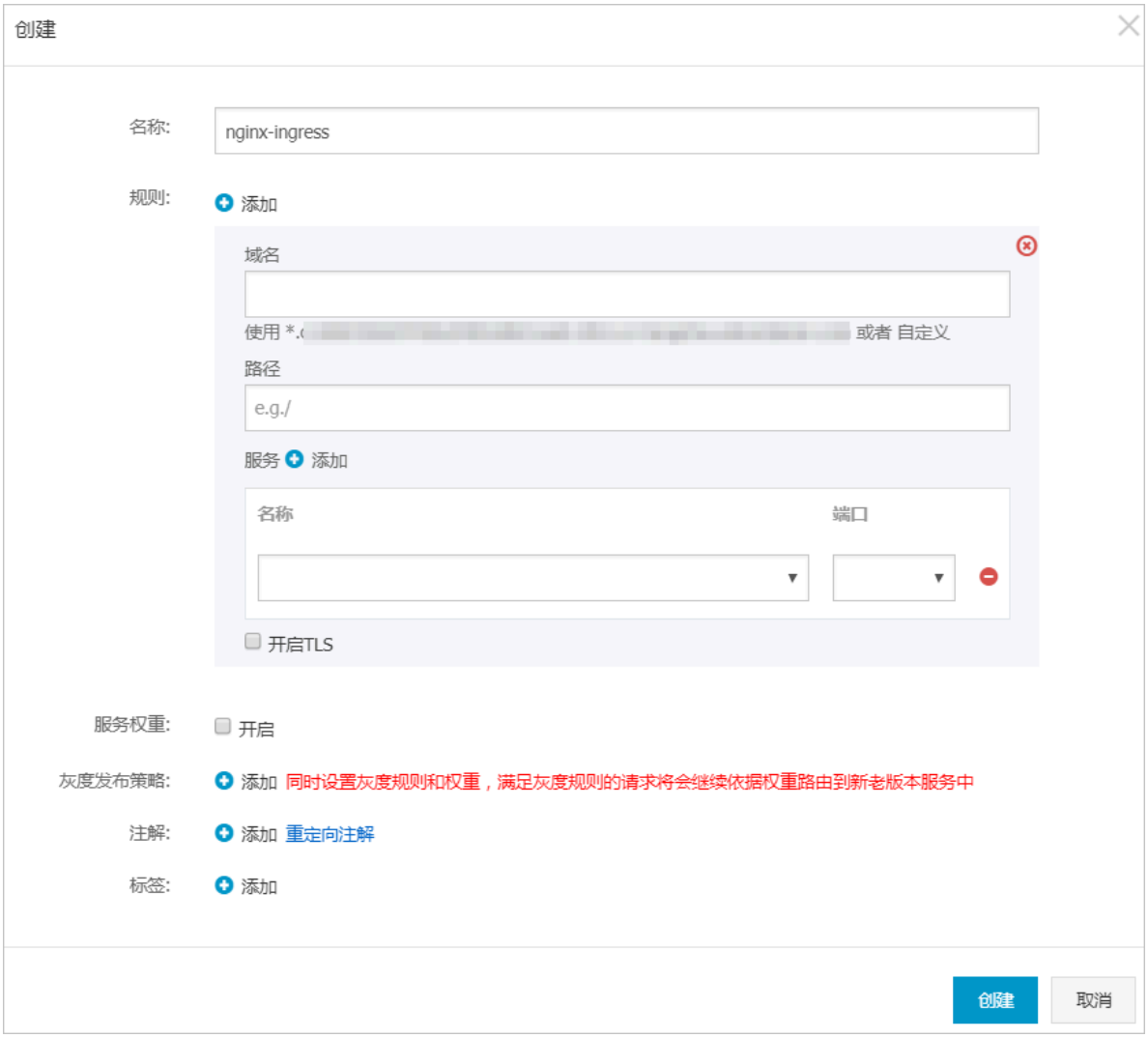
2. 详细信息可参考镜像创建无状态Deployment应用。



方法二：通过创建路由配置

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

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



负载均衡路由配置

容器服务Swarm集群

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

容器服务Kubernetes集群

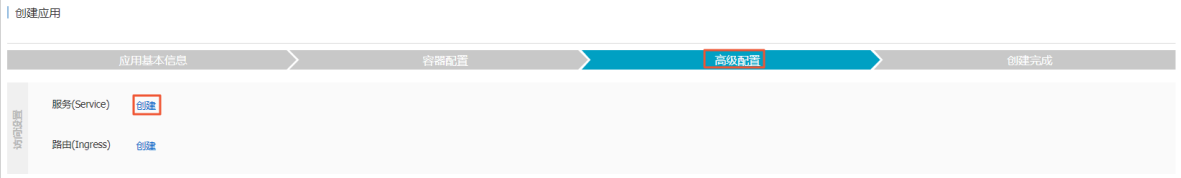
容器服务Kubernetes集群同样支持通过绑定SLB的方式进行应用访问方式的暴露。容器服务Kubernetes集群上创建SLB是通过LoadBalancer类型的Service进行自动创建，不需要手工

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

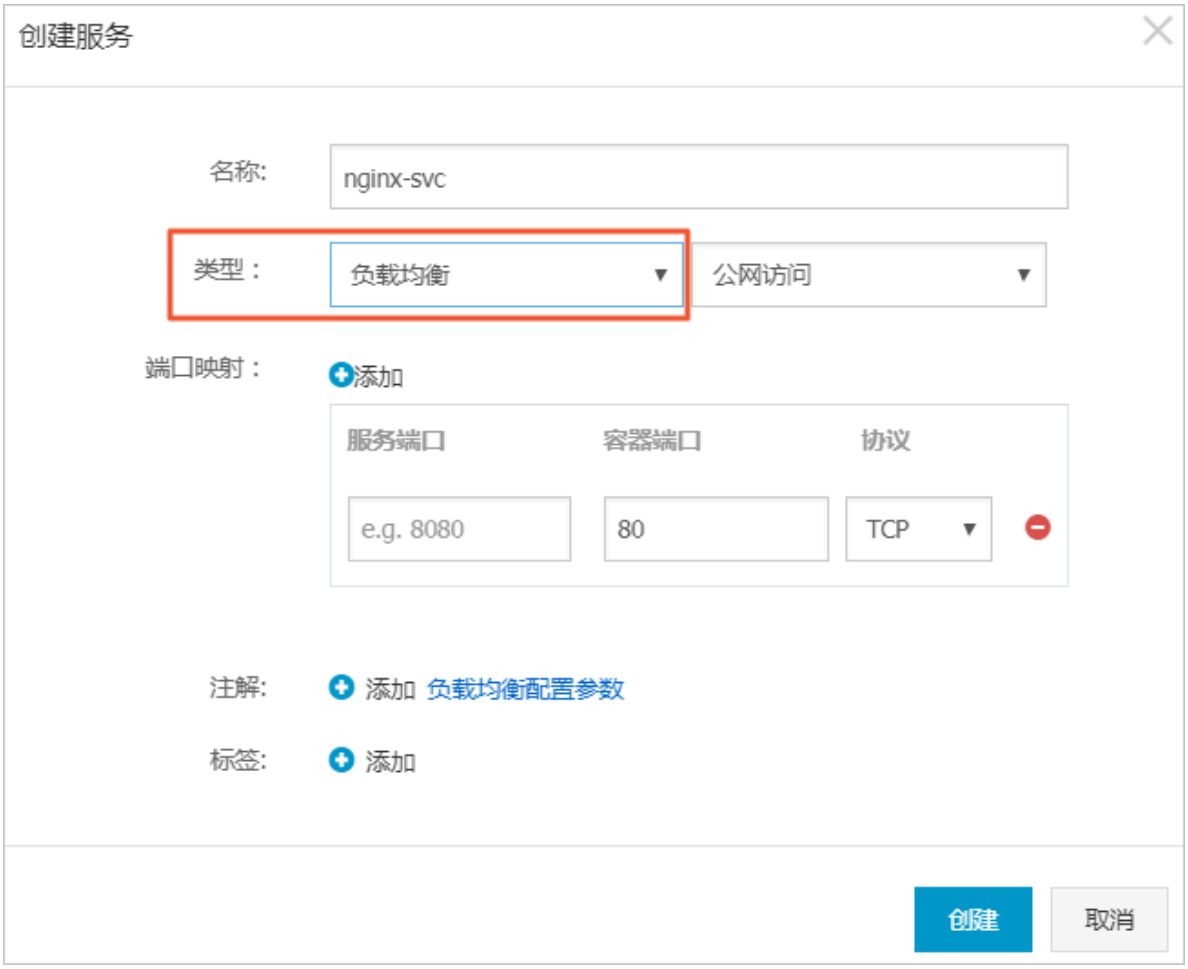
容器服务Kubernetes集群上创建Loadbalancer类型的Service有两种方式：

方法一：创建应用时配置

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



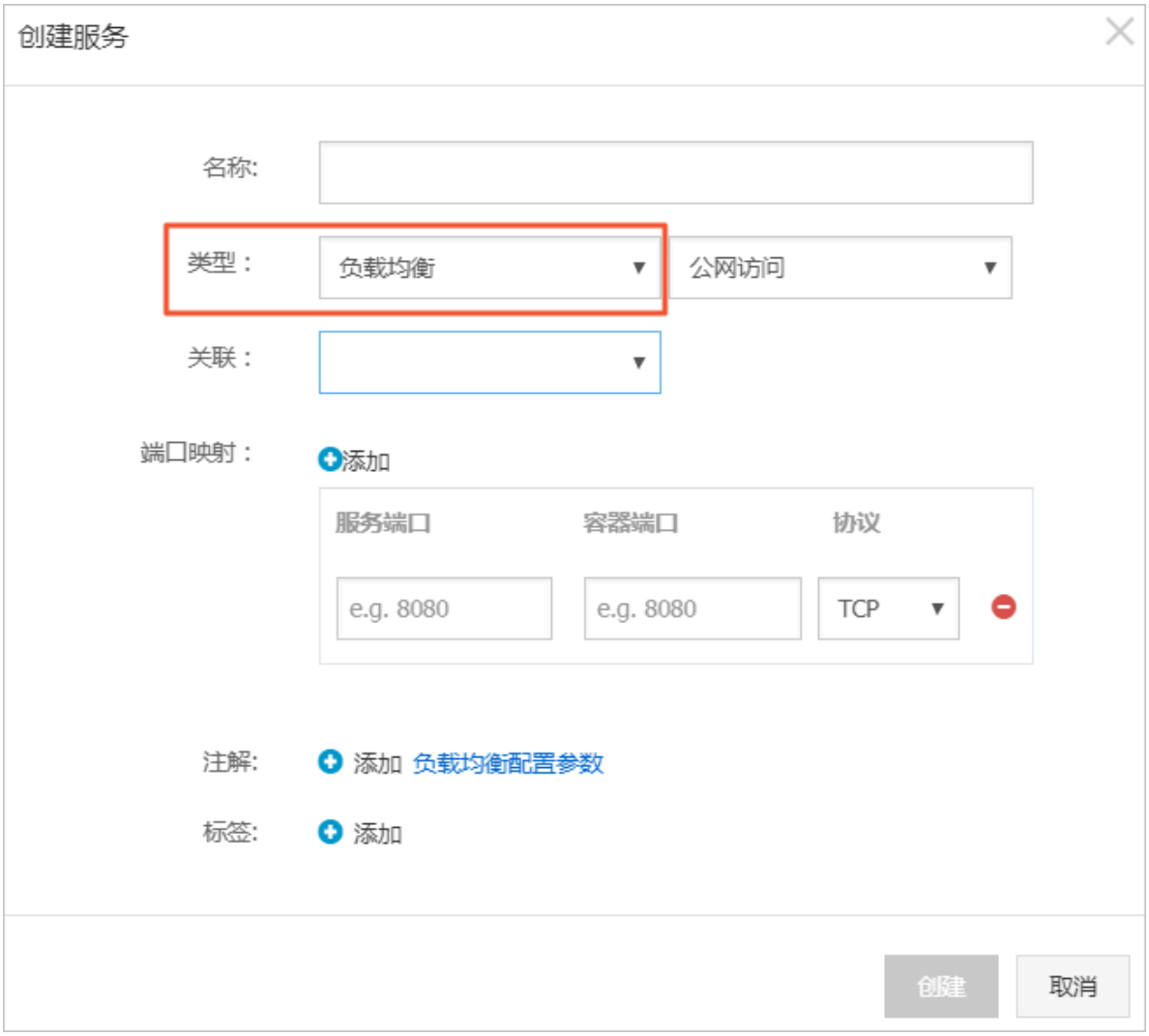
2. 类型请选择负载均衡。详细信息可参考镜像创建无状态Deployment应用。



方法二：通过创建服务配置

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

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



10.1.5 使用镜像创建应用-数据卷及环境变量配置比对

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

使用镜像创建应用

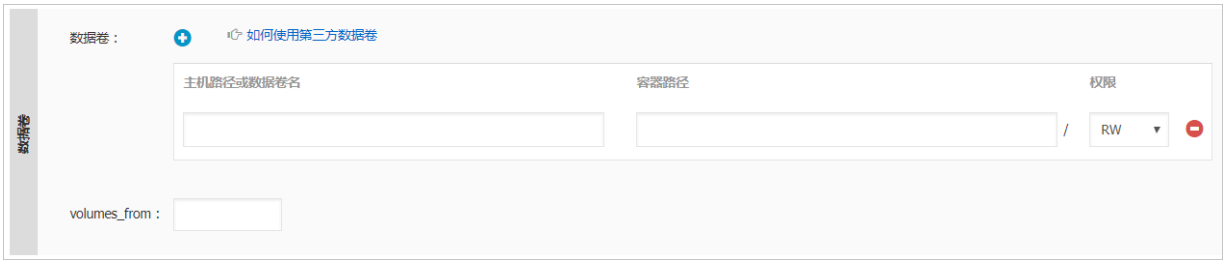
容器服务Swarm与Kubernetes集群在使用镜像创建应用时，部署界面差异较大。

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

数据卷

容器服务Swarm集群

填写用户申请的云存储或者本地存储路径。



容器服务Kubernetes集群

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

用户可以选择本地存储，也可以选择云存储：

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

环境变量

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

10.1.6 使用镜像创建应用-容器配置及标签比对

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

使用镜像创建应用

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

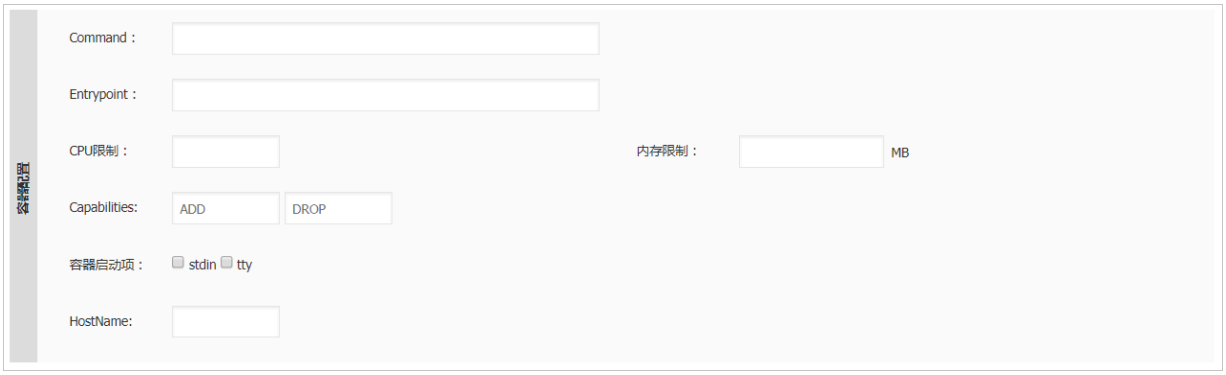
- **容器服务Swarm集群，请参考创建应用。**

- 容器服务Kubernetes集群，请参考镜像创建无状态Deployment应用。

容器配置

容器服务Swarm集群

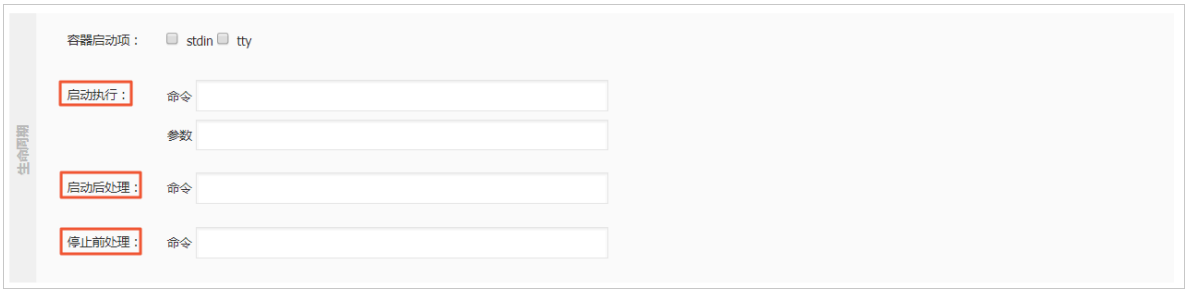
设置容器的启动命令（Command和Entrypoint）、资源限制（CPU限制和内存限制）及启动项等配置。



容器服务Kubernetes集群

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

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



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

标签

容器服务Swarm集群

能够完成健康检查、设置访问域名、日志等功能。

容器服务Kubernetes集群

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

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

10.1.7 使用镜像创建应用-健康检查及自动伸缩比对

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

使用镜像创建应用

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

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

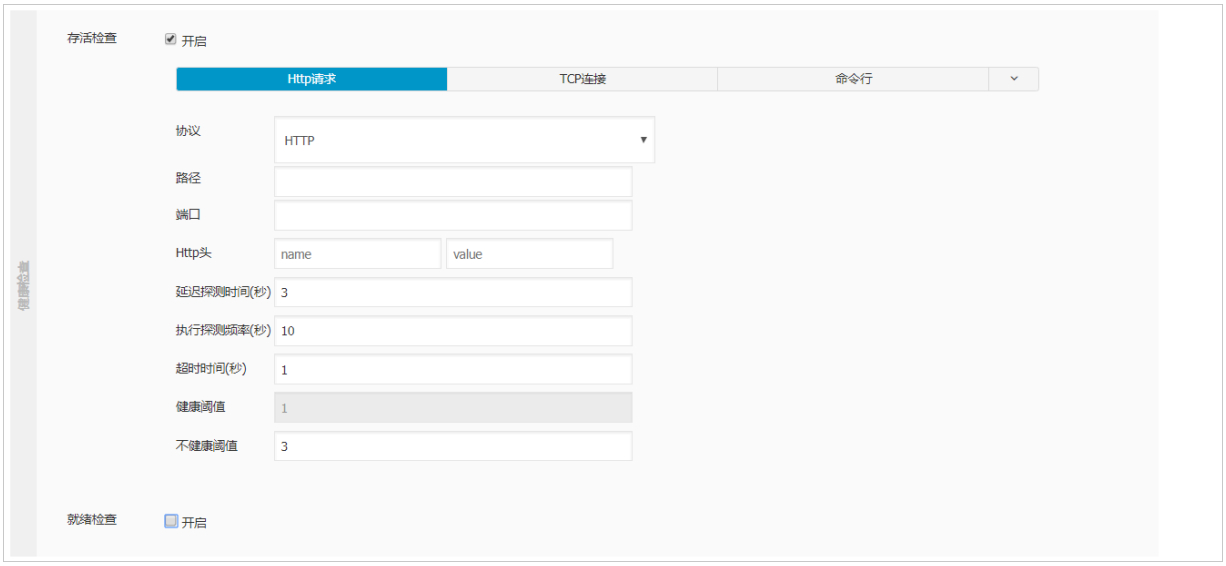
健康检查

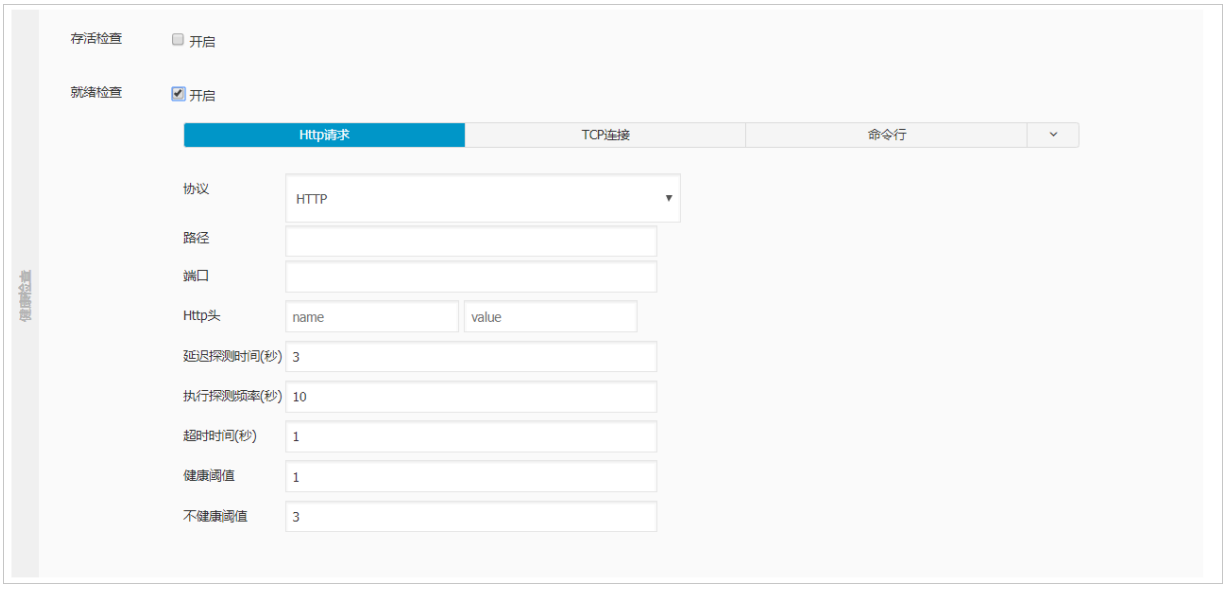
容器服务Swarm集群

健康检查是通过标签的方式实现。

容器服务Kubernetes集群

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





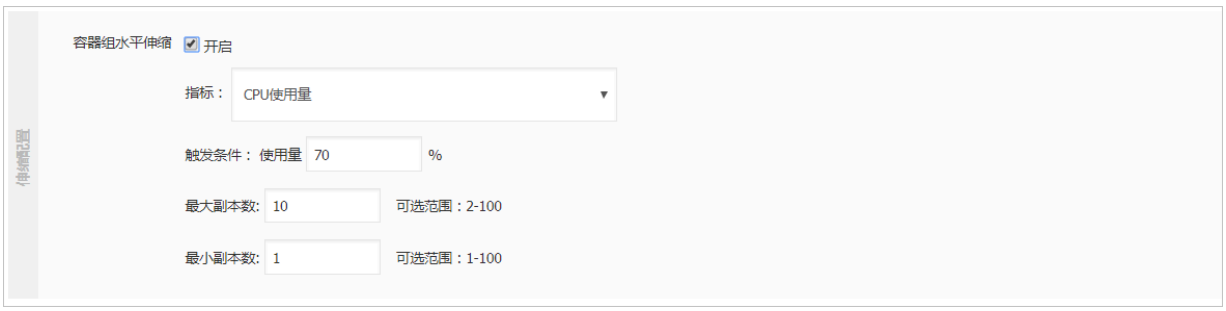
自动伸缩

容器服务Swarm集群

提供基于CPU和内存两个维度的自动伸缩。

容器服务Kubernetes集群

可配置容器组的水平伸缩，同样提供CPU和内存两个维度的自动伸缩。



10.1.8 使用yaml文件创建应用比对

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

背景信息

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

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

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

  Kompose工具下载地址：

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

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

  表 10-1: Kompose工具不支持的标签

标签	参考
external	external
dns_options	dns_options
oom_kill_disable	oom_kill_disable
affinity:service	服务部署约束（affinity:service）

- 您也可以手动改写yaml文件。

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

容器服务Swarm与Kubernetes集群的yaml文件比对

容器服务Swarm集群

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

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

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

容器服务Kubernetes集群

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

10.1.9 网络比对

本文介绍容器服务Swarm集群与Kubernetes集群的网络比对。

容器服务Swarm集群

容器服务Swarm集群使用的网络有两种：

- VPC网络
- 经典网络

容器服务Kubernetes集群

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

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

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

10.1.10 日志及监控比对

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

日志

容器服务Swarm集群

通过标签实现日志等功能。

容器服务Kubernetes集群

Kubernetes集群的日志功能，可在以下情况配置及使用：

- 创建Kubernetes集群：

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

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

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

监控

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

容器服务Swarm集群

默认情况，不勾选。

容器服务Kubernetes集群

默认情况，勾选。

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

10.1.11 应用访问比对

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

集群内部应用间访问

容器服务Swarm集群

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

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

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

容器服务Kubernetes集群

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

在创建应用时，可以将需要被访问服务的服务名称以环境变量的方式使用。

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

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

从集群外部访问应用

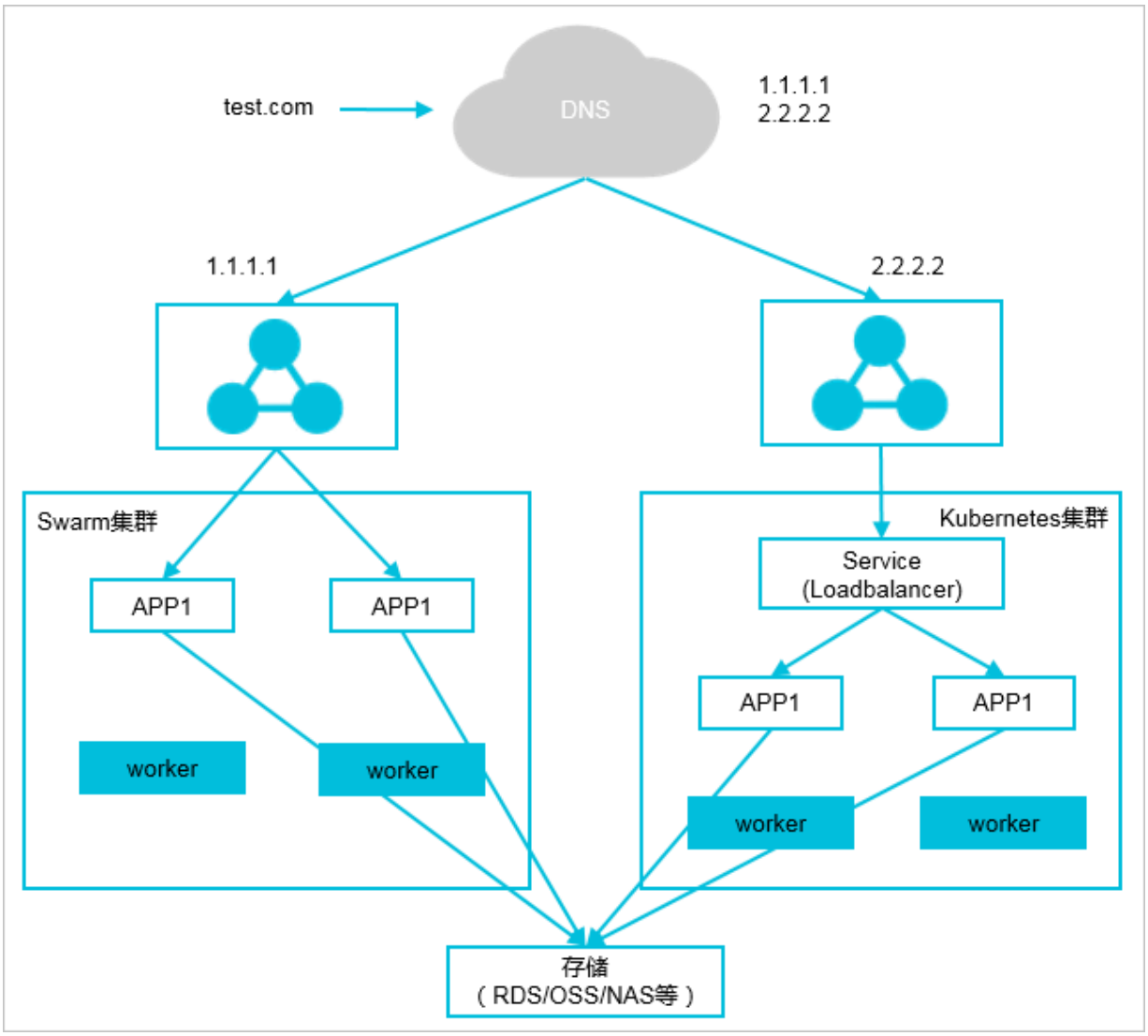
通过域名访问应用

说明：

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

简单路由（域名绑定到容器服务Swarm集群默认的SLB上）

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



迁移方法

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

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

  将DNS中`test.com`域名的后端IP地址1.1.1.1删除。

后续流量经过DNS后全部流向Kubernetes集群上的应用。

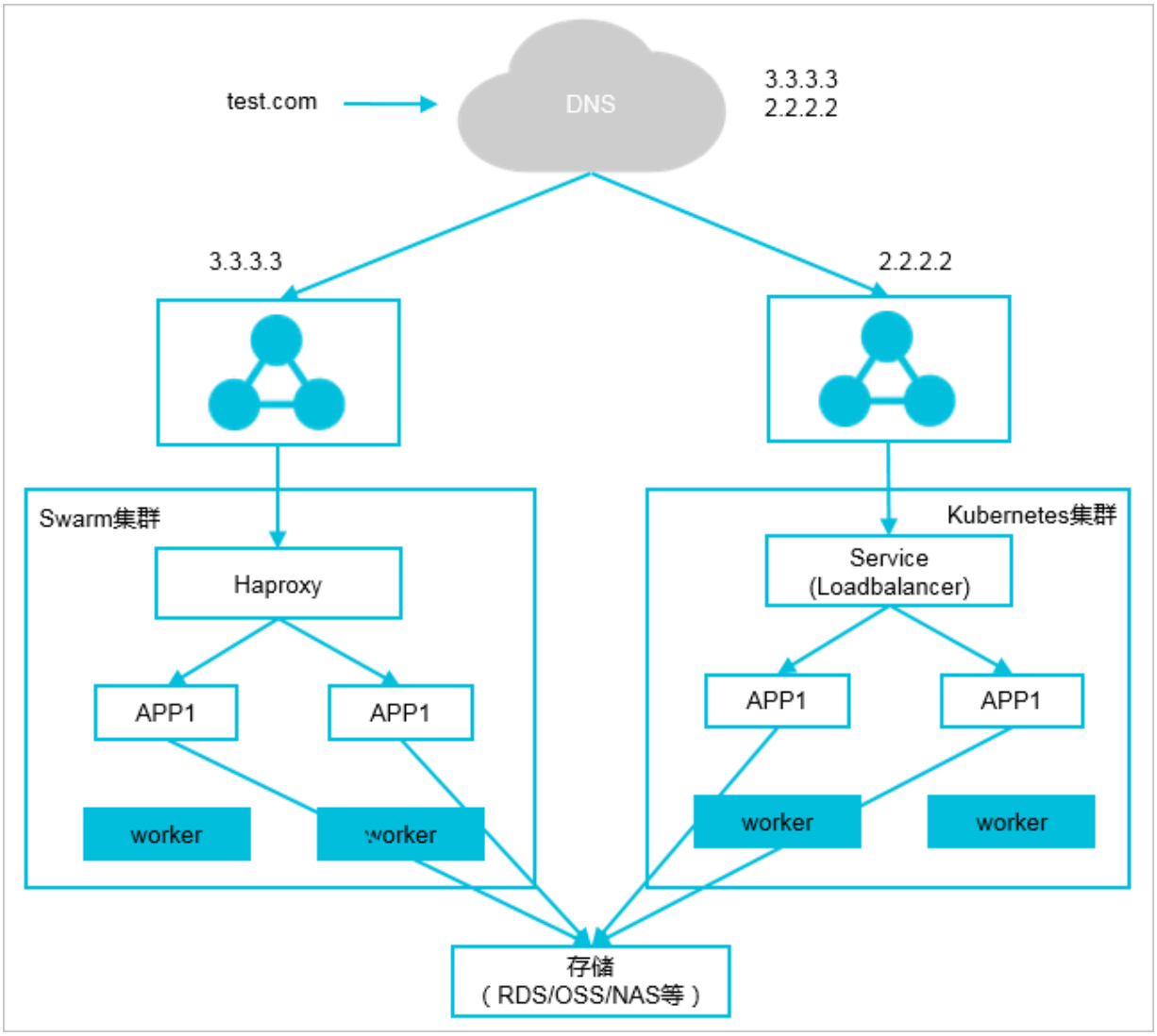
简单路由（域名绑定到应用在容器服务Swarm集群上自建的SLB）

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

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

迁移方法

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



通过<HostIP>:<port>访问应用

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

迁移方法

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

> **说明:**
> 此步骤需要逐个停止并修改应用实例。

3. 将Kubernetes集群内的worker节点，挂载到Swarm集群的SLB下。
4. 当有流量时，会有部分流量进入Kubernetes集群的应用，待Kubernetes集群的应用验证可用后，将SLB中Swarm集群的节点删除，完成集群的迁移。

通过负载均衡访问应用

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

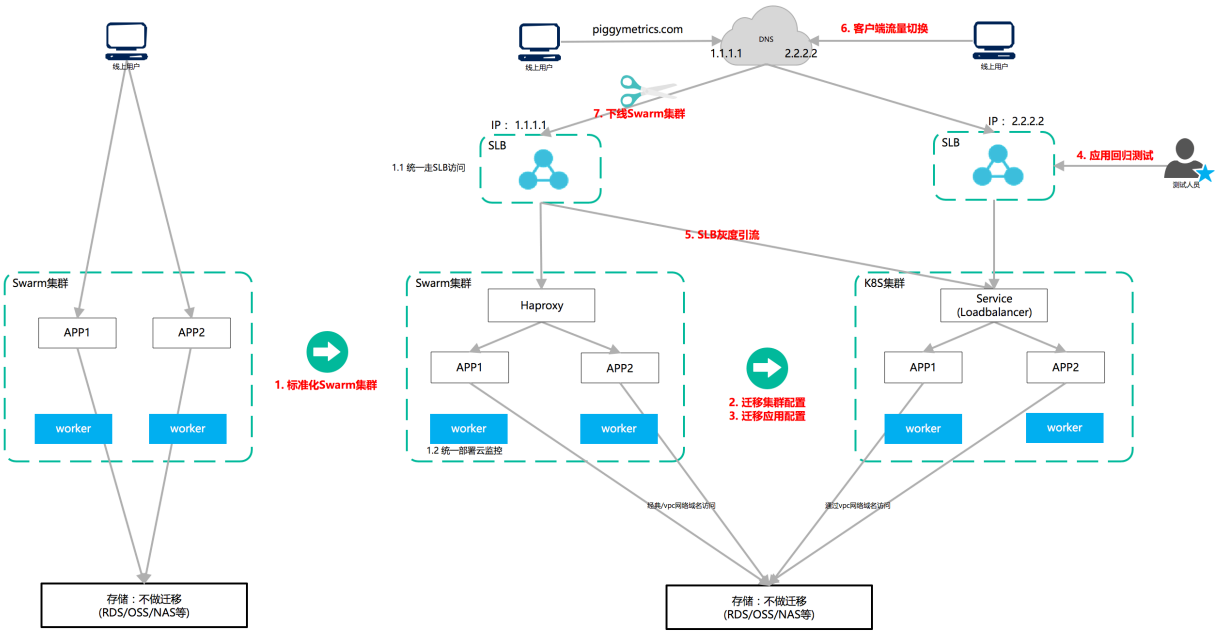
迁移方法

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

10.2 迁移方案概述

本文整体简单介绍下如何通过7个步骤，将容器服务 Swarm 集群平滑迁移到 Kubernetes 集群，并尽量确保迁移期间对业务无影响。

迁移方案



迁移步骤

1. 标准化Swarm集群

   由运维同学对已有 Swarm 集群做少量运维操作，降低后续 Kubernetes 集群迁移成本和风险

   a. 客户端统一通过SLB访问集群应用，保证后续SLB灰度引流实时生效

      - 如果通过SLB 访问应用，能够快速引流验证或出问题时能实时回滚（SLB控制台变更实时生效）
      - 如果不通过SLB而通过其他方式访问应用，则通过客户端发布或 DNS 回滚（时间区间为：0~48h）无法实时生效，导致对业务产生影响

   b. 统一部署云监控，方便后续监控各ECS运行情况，保证灰度引流生效

2. 迁移 Swarm 集群配置

   由运维同学完成 Kubernetes 集群创建及集群维度资源配置，降低开发人员应用迁移复杂度

   a. 集群机器资源及网络迁移
   b. Node 标签迁移
   c. VPC 网络互联互通验证
   d. 数据卷迁移
   e. 配置项迁移

3. 迁移应用配置

   由研发同学基于kompose转换工具完成应用配置迁移

   a. 准备迁移环境
   b. 预处理 Swarm 编排文件
   c. 转换 Swarm 编排文件
   d. 部署 Kubernetes 资源文件
   e. 手动迁移应用配置
   f. 应用启动调试
   g. 迁移应用日志配置

4. 应用回归测试

   应用在不影响线上流量情况下，由测试人员完成新集群业务功能回归测试

   a. 配置应用测试域名
   b. 测试业务功能
   c. 确认应用日志采集
   d. 确认应用监控

5. SLB灰度引流

   由运维同学逐步线上引流灰度验证，有问题能够快速回滚

   a. 配置NodePort引流
   b. 灰度引流快速回滚

6. 客户端流量切换

   由运维人员做客户端或DNS切换，将流量切换至新集群

   a. DNS流量切换：调整DNS解析配置实现流量切换
   b. 客户端流量切换：升级客户端代码或配置实现流量切换

7. 下线Swarm集群

   由运维人员确认新集群服务访问正常，下线老Swarm集群资源

   a. 确认Swarm集群流量
   b. 下线Swarm集群资源

10.3 标准化Swarm集群

从集群外部访问应用时，如果您通过`<HostIP>:<port>`访问应用或基于备案域名但直接解析到NodeIP场景，则在迁移Swarm集群时，无法做到不停机迁移业务和实时回滚，所以先调整成统一通过SLB访问应用，再通过部署监控插件的方式，完成标准化Swarm集群，确保集群迁移时不停机迁移业务和实时回滚。

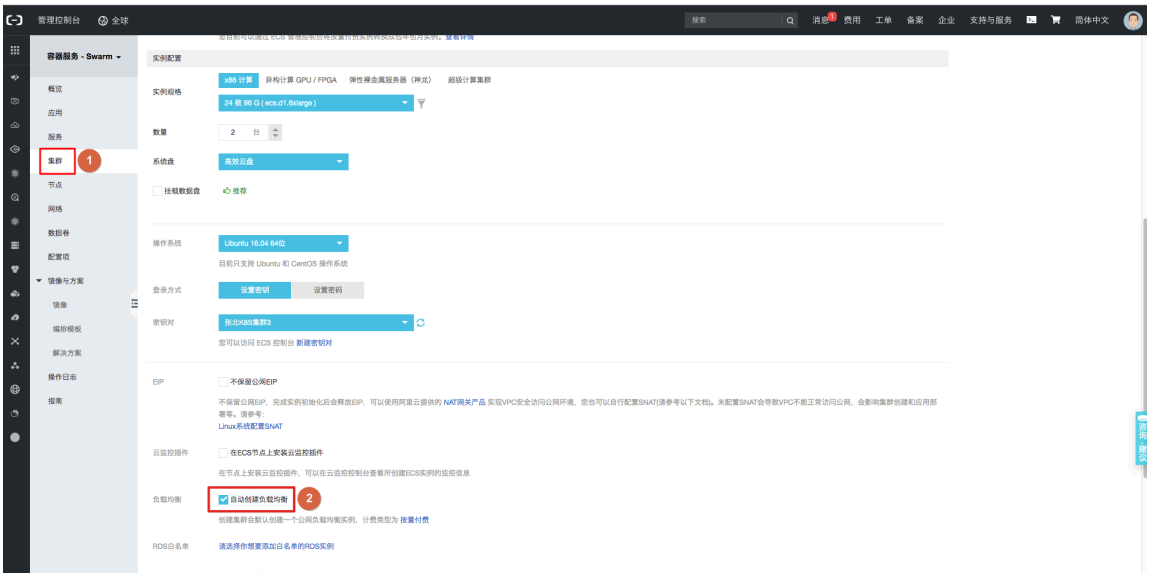
统一通过SLB访问Swarm集群

1. 创建SLB实例

- 自动创建 SLB

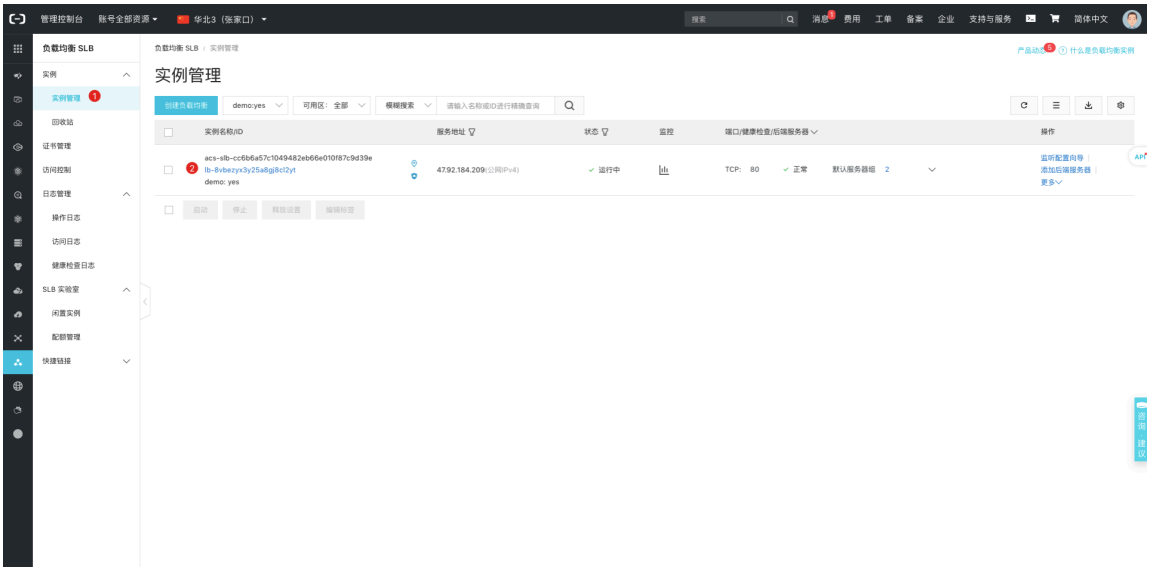
  如果您在创建Swarm集群时，已勾选自动创建负载均衡，如图 10-1: 设置SLB所示。表示您已自动创建一个按量付费的负载均衡实例。

  图 10-1: 设置SLB



  您可以登录负载均衡管理控制台，查看已创建的SLB，确认该SLB已配置了监听后端服务器的 TCP 9080 端口，如图 10-2: 查看服务端口所示。

  图 10-2: 查看服务端口



- 手动创建SLB

如果您创建Swarm集群时，未勾选自动创建负载均衡，则只能在集群创建完成后，手动创建SLB并做绑定，具体步骤如下：

a. 创建SLB。请参见创建负载均衡实例。

b. 为Swarm集群绑定SLB，保证后续该集群配置会自动更新SLB配置。请参见为集群绑定和解绑负载均衡。

2. 配置简单路由

原来通过`<HostIP>:<port>`访问应用，因将Host IP绑定在客户端，如后续出现Host 替换或扩容，只能通过升级客户端方式来访问新Host 服务；因此，建议您通过域名访问服务的方式。

> **说明:**
> 如果没有申请自定义域名，阿里云容器服务免费支持为每个应用或服务配置独立域名，格式为`XXX.$cluster_id.$region_id.alicontainer.com`，其中XXX域名前缀可以根据每个应用或服务自行定义。

通过这种方式，可以将服务访问切换成通过域名访问（用户自定义域名或容器服务提供的测试域名均可）的形式，而不再依赖于ECS或SLB的`<HostIP>:<port>`访问应用这种方式；详细操作请参见简单路由（支持 HTTP/HTTPS)。

3. SLB访问业务功能测试

您可以使用上个步骤配置的域名访问业务，测试业务功能是否正常，并观察SLB流量是否有流量进入。

a. 使用`http://<HostIP>:<port>`访问业务。

图 10-3: 容器服务提供的域名

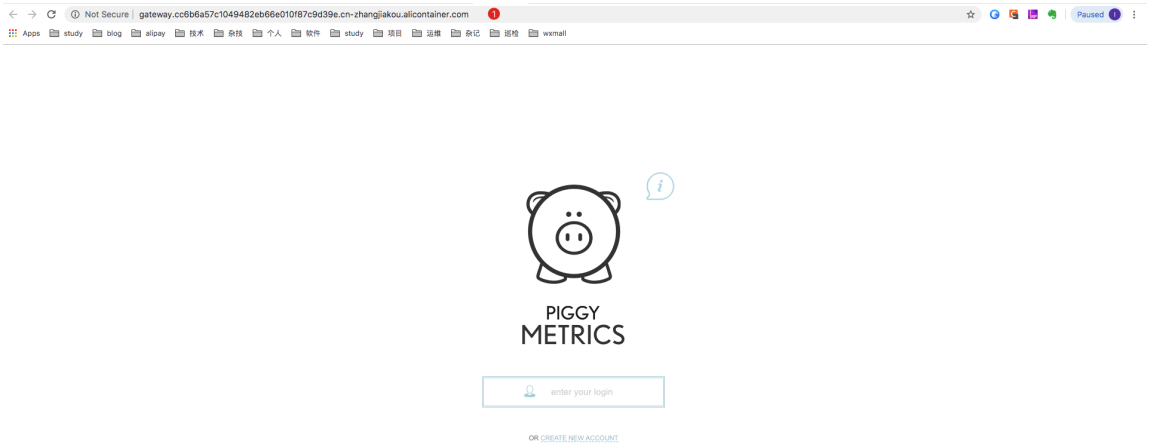
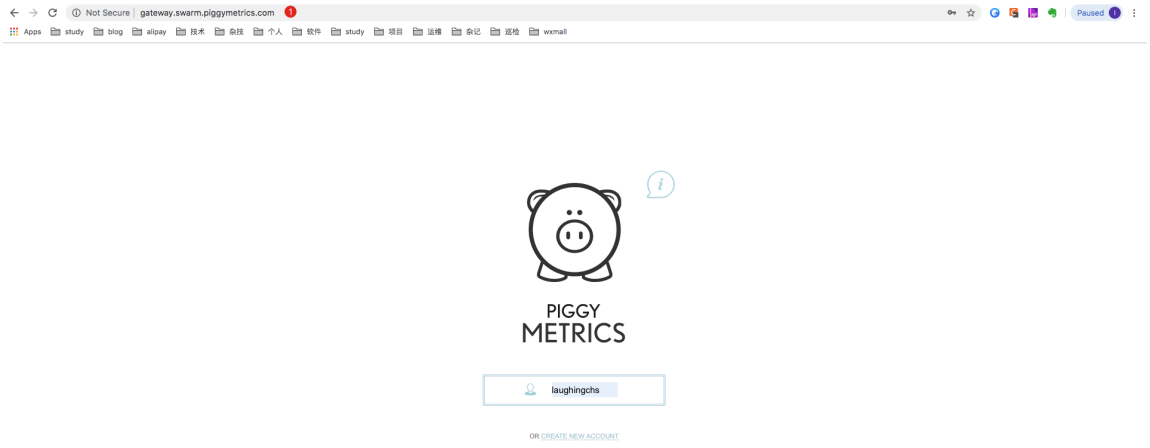
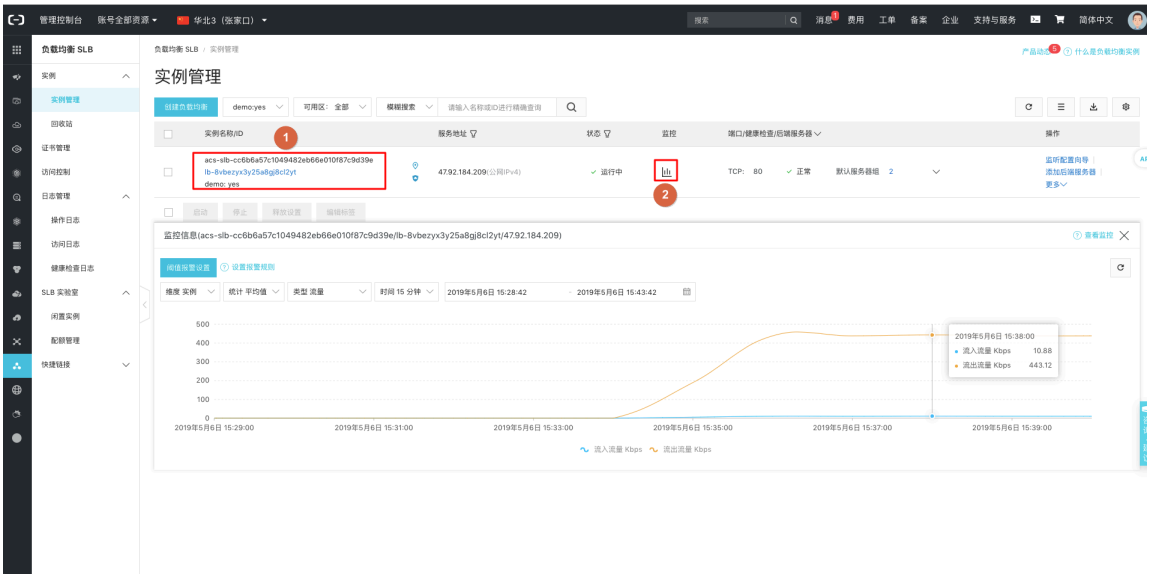


图 10-4: 用户自定义的域名



b. 查看SLB监控是否有流量进入。



> **说明：**
> 请完成业务功能测试，确认业务流量通过SLB访问域名的方式对业务无影响。

4. 修改流量解析入口。

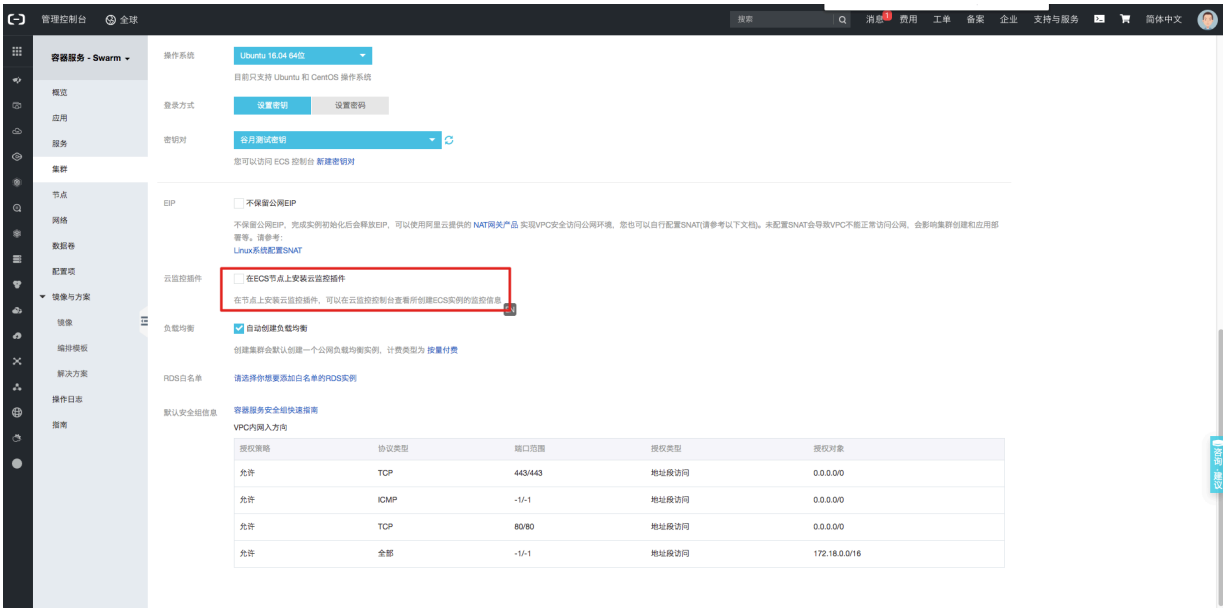
   - 修改DNS解析地址：若原先客户端是通过域名方式访问服务的，请联系DNS服务商修改DNS解析，将原来Node Host IP修改成SLB IP。

     > **说明：**
     > 修改DNS解析地址后，各DNS服务商都有缓存，需要观察SLB监控看流量是否已切换完成。

   - 修改客户端访问地址：若原先客户端是通过Node Host IP访问的，请升级客户端，修改成基于域名访问的方式。

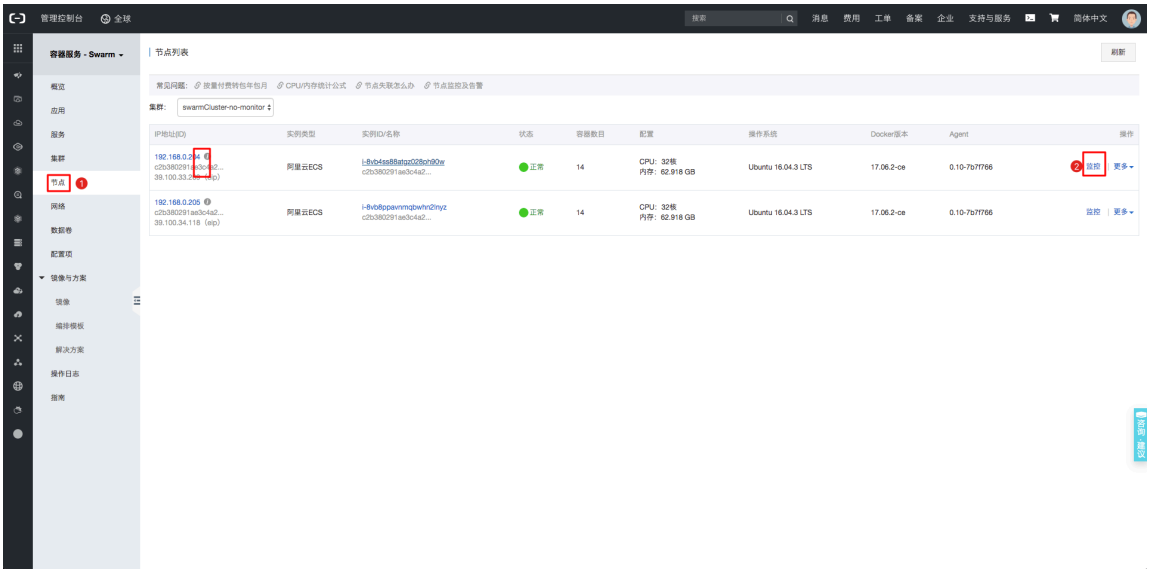
统一部署云监控（可选）

如果在创建集群时，没有勾选在ECS节点上安装云监控插件，则无法在云监控查看ECS实例的监控信息，不利于后续迁移时观察集群情况。例如下线Swarm集群时，无法通过云监控确认对应ECS机器是否还有流量。您可以通过如下操作，部署云监控。

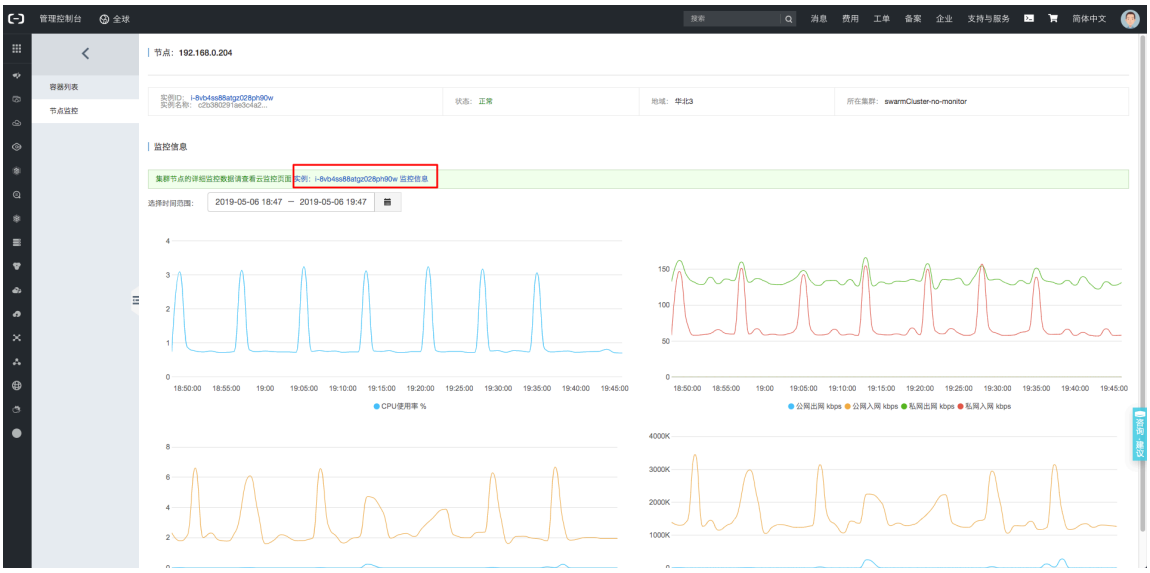

管理控制台
全球
搜索
消息
费用
工单
备案
企业
支持与服务
简体中文
容器服务 - Swarm
概览
应用
服务
集群
节点
网络
数据卷
配置项
镜像与方案
镜像
编排模板
解决方案
操作日志
指南
操作系统
Ubuntu 16.04 64位
目前只支持 Ubuntu 和 CentOS 操作系统
登录方式
设置密钥
设置密码
密钥对
谷月测试密钥
您可以访问 ECS 控制台 新建密钥对
EIP
不保留公网EIP
不保留公网EIP，完成实例初始化后会释放EIP，可以使用阿里云提供的 NAT网关产品 实现VPC安全访问公网环境。您也可以自行配置SNAT(请参考以下文档)。未配置SNAT会导致VPC不能正常访问公网，会影响集群创建和应用部署等。请参考：
Linux系统配置SNAT
云监控插件
在ECS节点上安装云监控插件
在节点上安装云监控插件，可以在云监控控制台查看所创建ECS实例的监控信息
负载均衡
自动创建负载均衡
创建集群会默认创建一个公网负载均衡实例，计费类型为 按量付费
RDS白名单
请选择你想要添加白名单的RDS实例
默认安全组信息
容器服务安全组快速指南
VPC内网入方向
授权策略
协议类型
端口范围
授权类型
授权对象
允许
TCP
443/443
地址段访问
0.0.0.0/0
允许
ICMP
-1/-1
地址段访问
0.0.0.0/0
允许
TCP
80/80
地址段访问
0.0.0.0/0
允许
全部
-1/-1
地址段访问
172.18.0.0/16

1. 查看Swarm集群节点ECS监控信息。

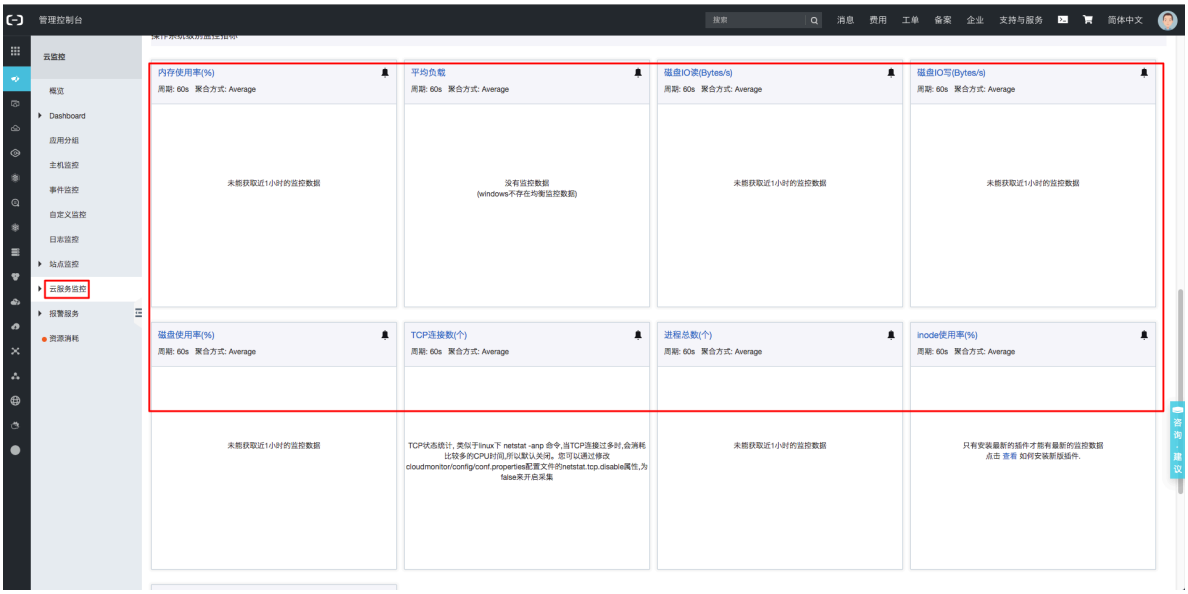
   a. 您可以登录容器服务管理控制台，在左侧菜单栏单击节点，在目标节点右侧单击监控，进入云监控入口链接页面。



   b. 单击节点监控，在监控信息区域可以查看到该节点在某个时间区间内的详细监控信息。

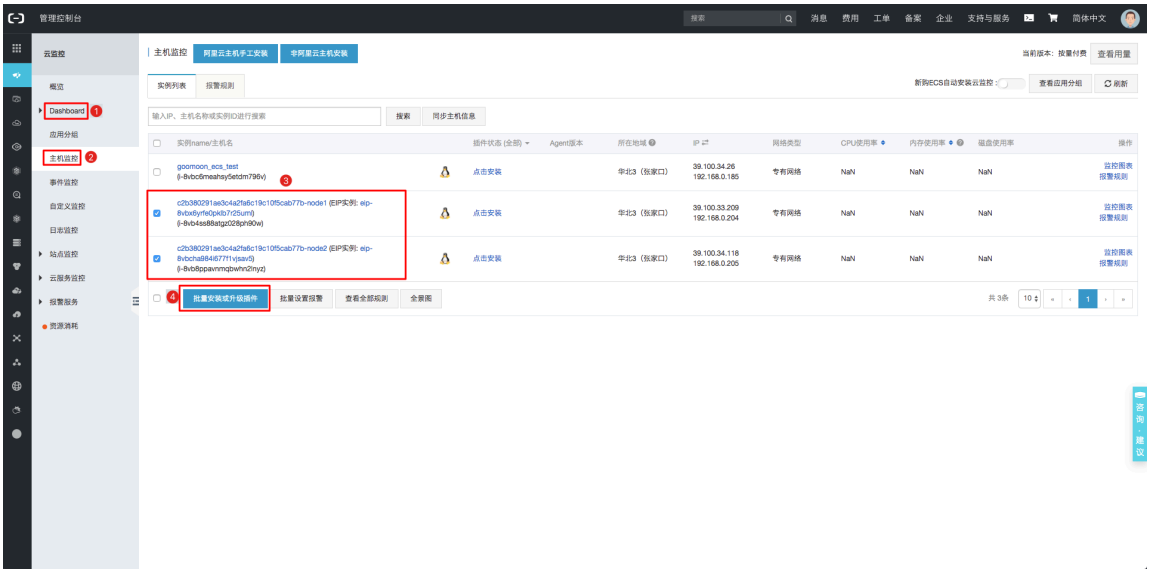


如果未安装云监控插件，则无法查看ECS详细监控信息如内存、磁盘、TCP连接数等。您可以按照如下操作，ECS云监控插件。

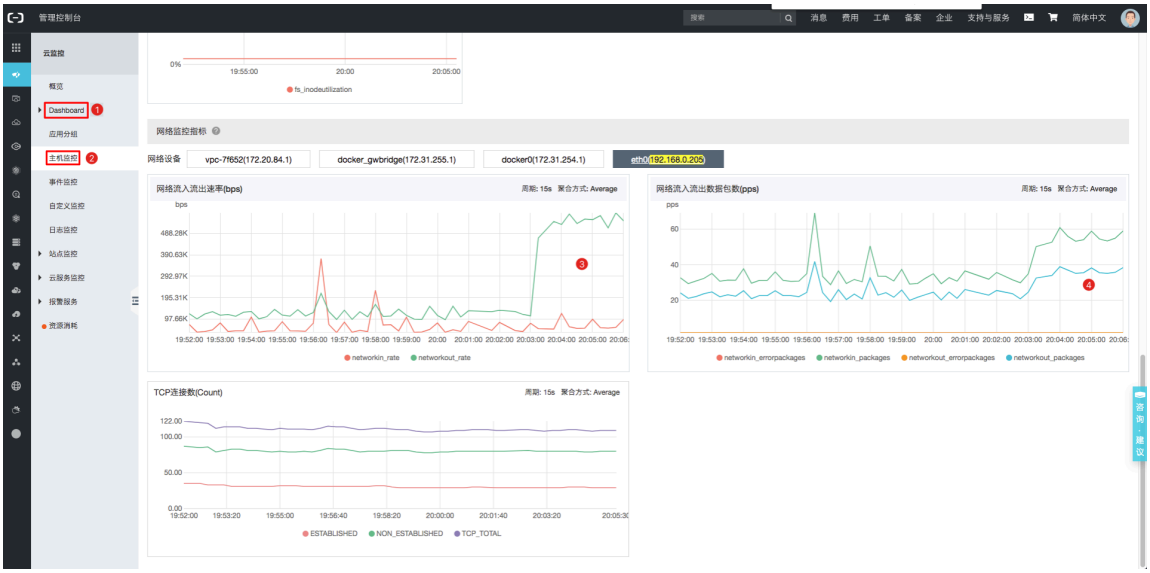

管理控制台
搜索
消息
费用
工单
备案
企业
支持与服务
简体中文
云监控
概览
Dashboard
应用分组
主机监控
事件监控
自定义监控
日志监控
站点监控
云服务监控
报警服务
资源消耗
内存使用率(%)
周期: 60s 聚合方式: Average
未能获取近1小时的监控数据
平均负载
周期: 60s 聚合方式: Average
没有监控数据
(windows不存在均衡监控数据)
磁盘IO读(Bytes/s)
周期: 60s 聚合方式: Average
未能获取近1小时的监控数据
磁盘IO写(Bytes/s)
周期: 60s 聚合方式: Average
未能获取近1小时的监控数据
磁盘使用率(%)
周期: 60s 聚合方式: Average
未能获取近1小时的监控数据
TCP连接数(个)
周期: 60s 聚合方式: Average
TCP状态统计，类似于linux下 netstat -anp 命令,当TCP连接过多时,会消耗比较多的CPU时间,所以默认关闭。您可以通过修改cloudmonitor/config/conf.properties配置文件的netstat.tcp.disable属性,为false来开启采集
进程总数(个)
周期: 60s 聚合方式: Average
未能获取近1小时的监控数据
inode使用率(%)
周期: 60s 聚合方式: Average
只有安装最新的插件才能有最新的监控数据
点击 查看 如何安装新版插件.
咨询·建议

2. 在Swarm集群节点上手动安装ECS云监控插件。

   a. 登录云监控控制台，在左侧导航栏选择Dashborad > 主机监控，在实例列表页签，勾选目标实例，并单击升级插件或批量安装，选择对应的ECS机器列表升级插件即可。



   b. 安装完成后，可以在Dashborad > 主机监控中，看到网络监控指标中，该网络设备的网络流入流出速率的曲线图。



10.4 迁移集群配置

本文以swarm-piggymetrics-cluster集群为例，为您介绍如何迁移该集群的配置，配置迁移主要包含创建Kubernetes集群、VPC网络互联互通验证、数据卷迁移、配置项迁移等操作。

前提条件

待迁移的Swarm集群已完成标准化改造。请参见标准化Swarm集群。

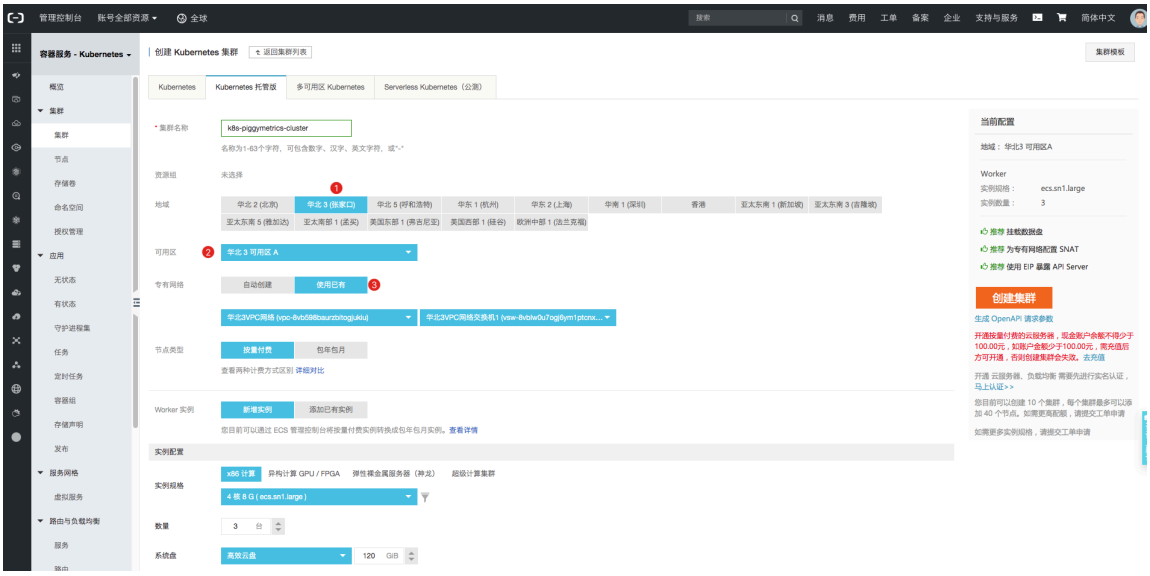
背景信息

本文以swarm-piggymetrics-cluster集群为例，为您介绍如何迁移该集群的配置，配置迁移主要包含创建Kubernetes集群、VPC网络互联互通验证、数据卷迁移、配置项迁移等操作。

创建Kubernetes集群

基于已有Swarm集群（swarm-piggymetrics-cluster）创建一个Kubernetes集群，建议您创建Kubernetes 托管版集群。创建集群时，您需要注意以下情况：

- 地域和可用区选择上，选择跟swarm-piggymetrics-cluster集群同地域同可用区，以保证像VPC网络这类无法跨地域的资源能正常迁移。
- Kubernetes 集群只支持VPC网络，不支持经典网络。
  - 如果swarm-piggymetrics-cluster集群为VPC网络，请创建Kubernetes 托管版集群时共用swarm-piggymetrics-cluster集群的VPC网络。
  - 如果swarm-piggymetrics-cluster集群为经典网络，则需要做VPC网络迁移，详情参见混挂混访方案。



- 请创建Kubernetes 托管版集群时，请确保已安装云监控插件，方便后续灰度或切流时，可通过云监控平台查看ECS实例的监控信息。
- 如果swarm-piggymetrics-cluster集群有使用阿里云日志服务，请创建Kubernetes 托管版集群时打开日志服务，并选择复用已有Project，避免需要重新迁移日志服务配置。

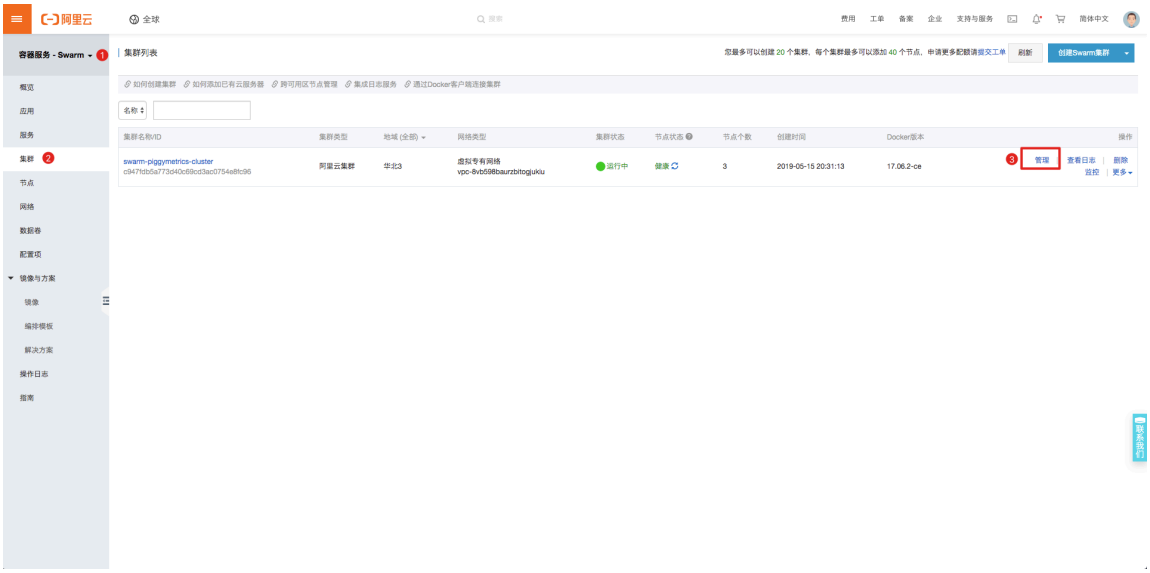
- 如果swarm-piggymetrics-cluster集群有使用RDS云产品，请在创建Kubernetes 托管版集群时，选择对应的RDS白名单，避免出现网络权限问题。

迁移Node标签

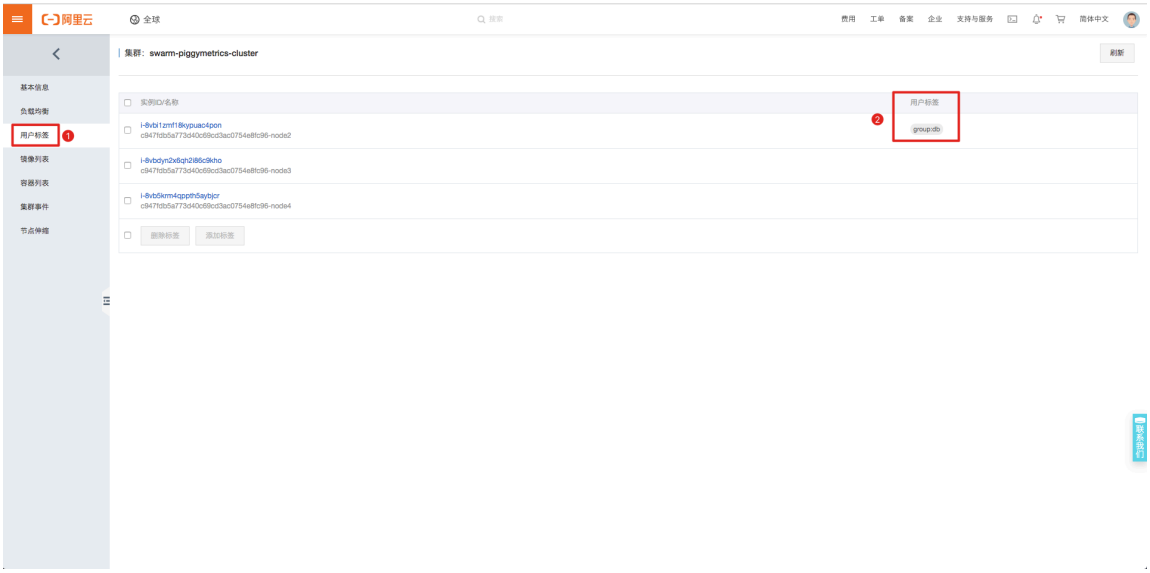
如果swarm-piggymetrics-cluster集群有对 Node 节点打用户标签，并在应用配置里面通过用户标签指定Node 节点部署；则需要做 Node 用户标签的迁移。迁移过程如下：

1. 查看Swarm 集群用户标签。

   a) 登录容器服务Swarm控制台，在左侧导航栏单击集群，在目标集群右侧单击管理。

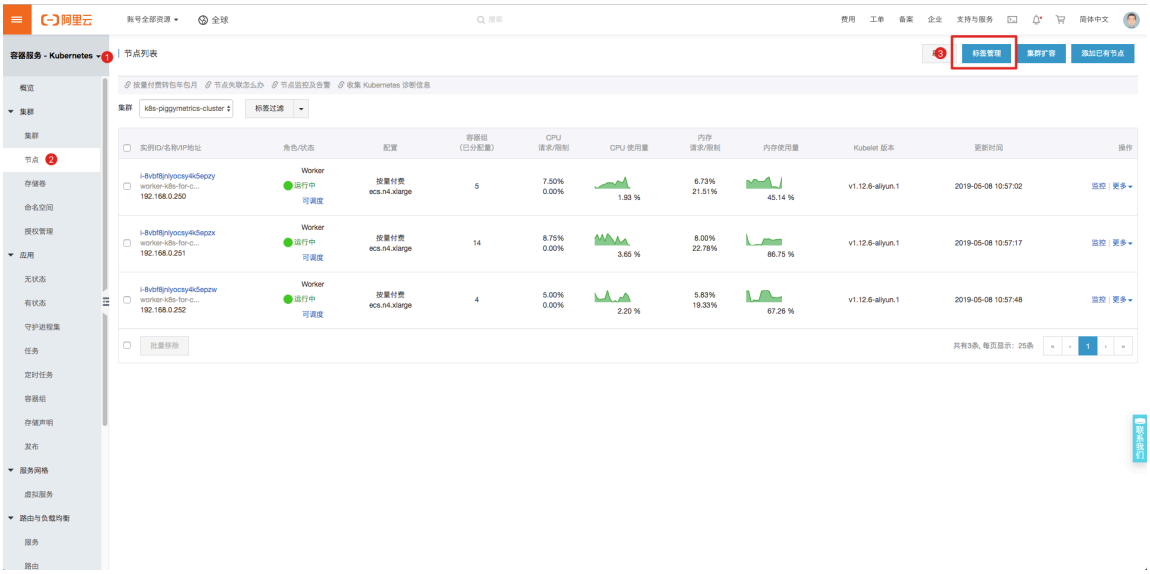


   b) 单击用户标签，在swarm-piggymetrics-cluster集群右侧可以看到对应的用户标签。

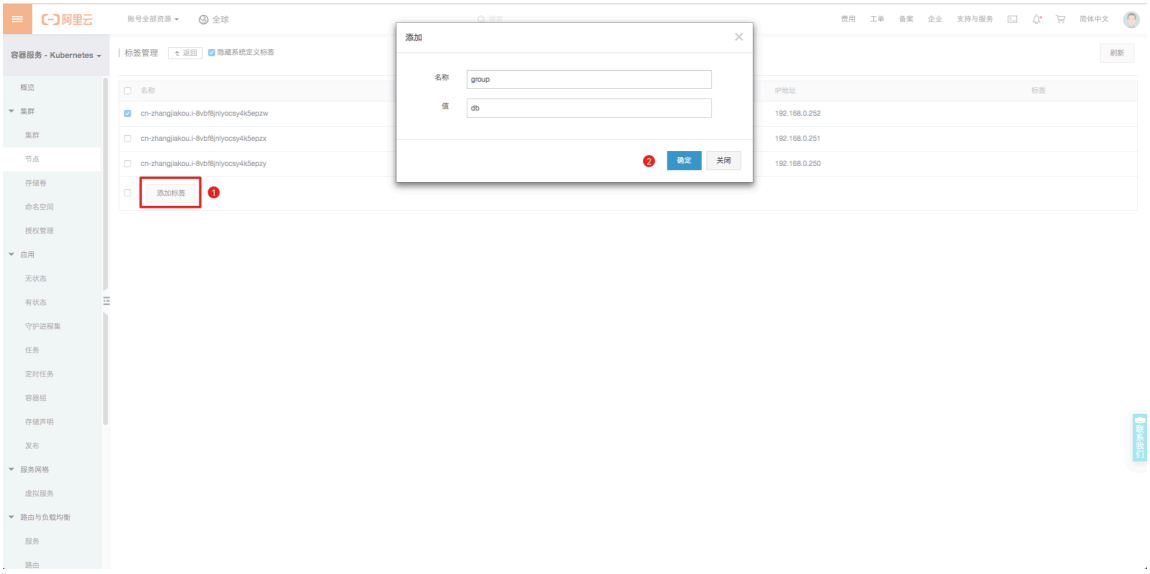


2. 在创建Kubernetes 托管版集群中对应 Node 节点打上同样的标签。

a) 登录容器服务管理控制台，选择集群 > 节点，在节点列表右侧单击标签管理。



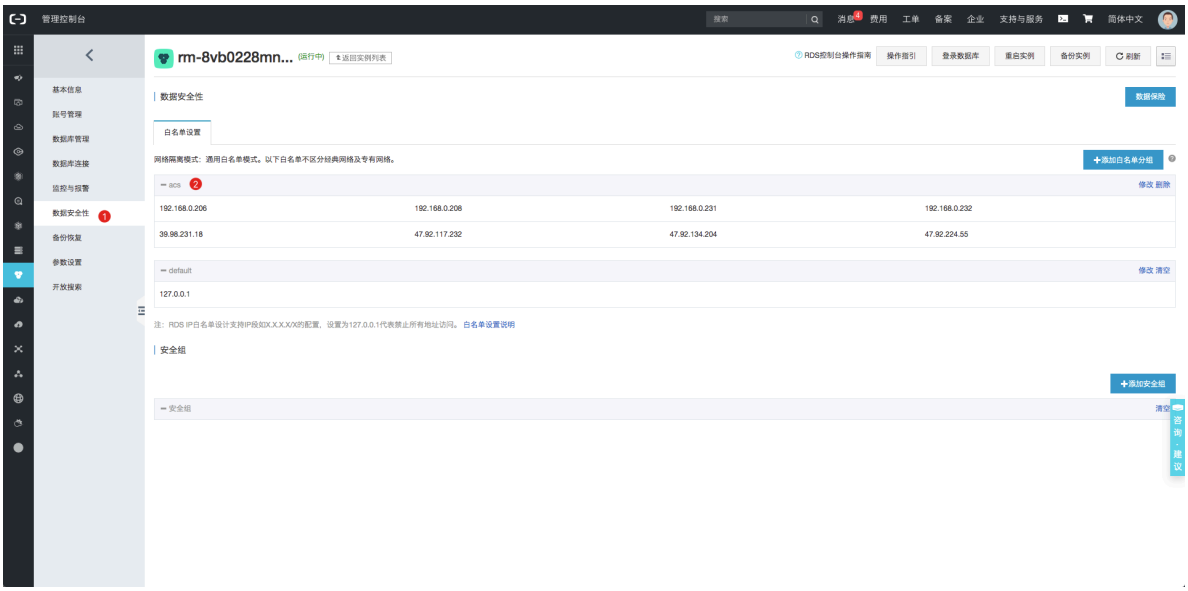
b) 在标签管理页面，勾选目标节点，单击添加标签，填写标签的名称及值，单击确定。



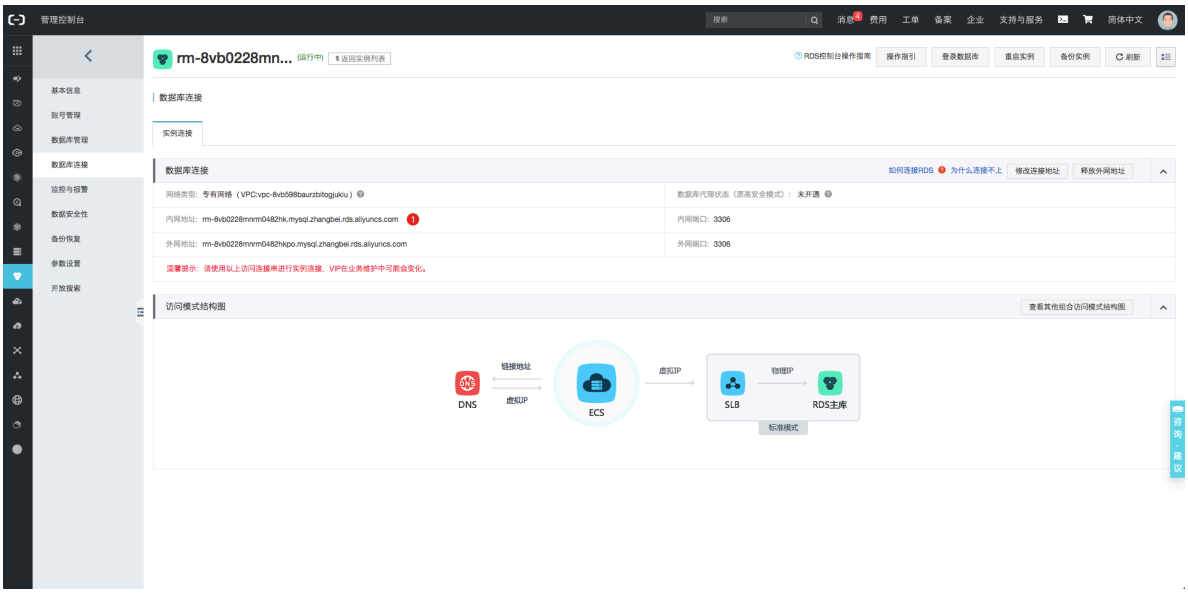
VPC互联互通验证

如果swarm-piggymetrics-cluster集群是经典网络，而创建Kubernetes 托管版集群是VPC网络，则可能访问RDS、OSS 这类产品时，出现网络不通的情况；其中RDS会有白名单限制（我们在上述创建Kubernetes 托管版集群时，已选择添加RDS白名单），而OSS本身则针对经典网络和VPC网络提供两个不同的访问域名，故需要在创建Kubernetes 托管版集群上进行网络连通性验证。

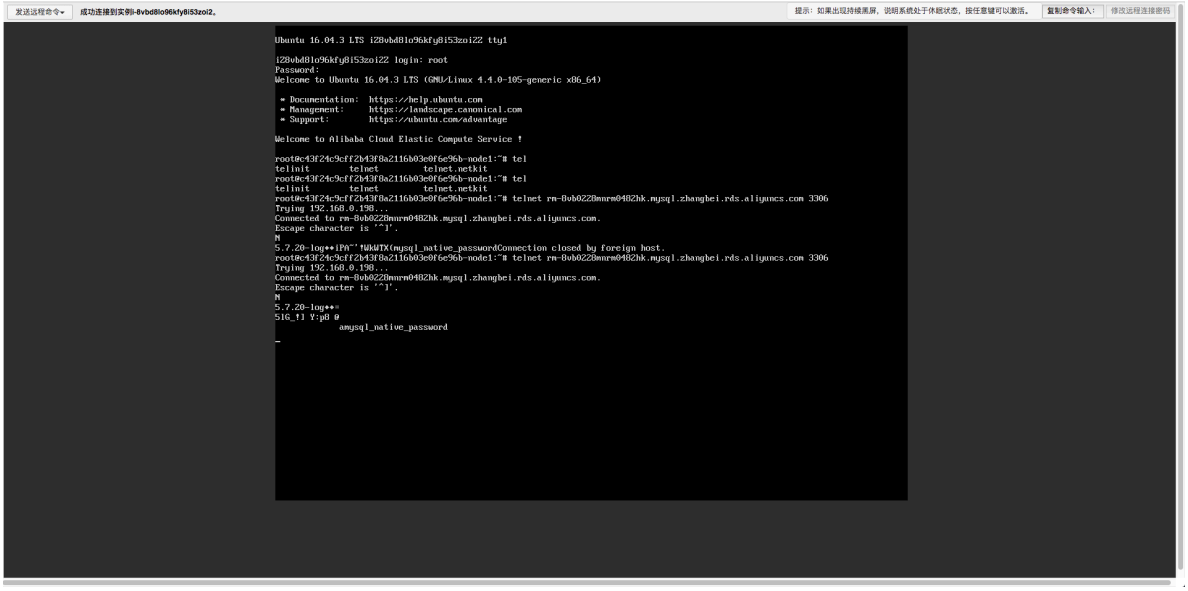
1. 登录RDS控制台，单击目标实例，进入实例基本信息页面，单击左侧导航栏数据安全性，在白名单设置中，确认RDS白名单已添加成功。



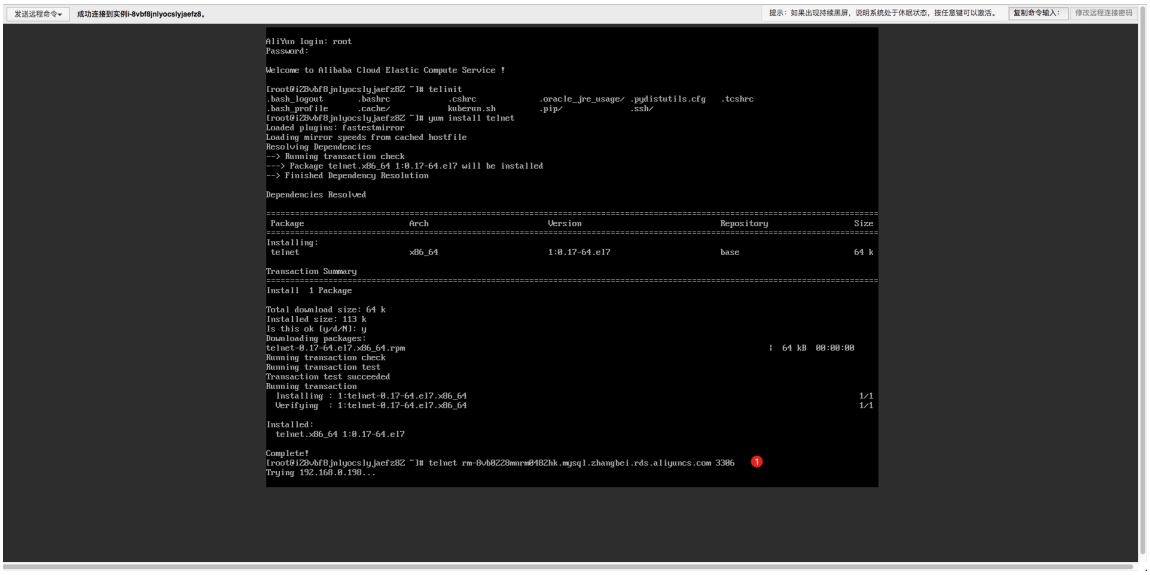
2. 单击数据库连接，在实例连接页签中，看到的内网地址即为RDS的VPC网络域名。



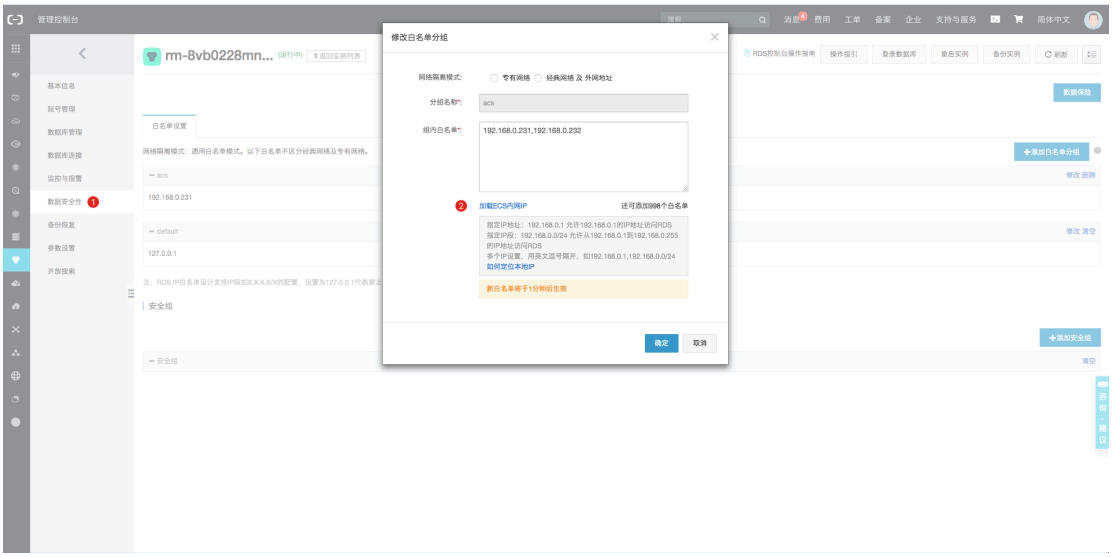
3. 登录ESC管理控制台，根据界面提示操作，确认网络连通性是否正常。



- 网络连通，进行下一步操作。
- 网通不同，可能存在如下原因。

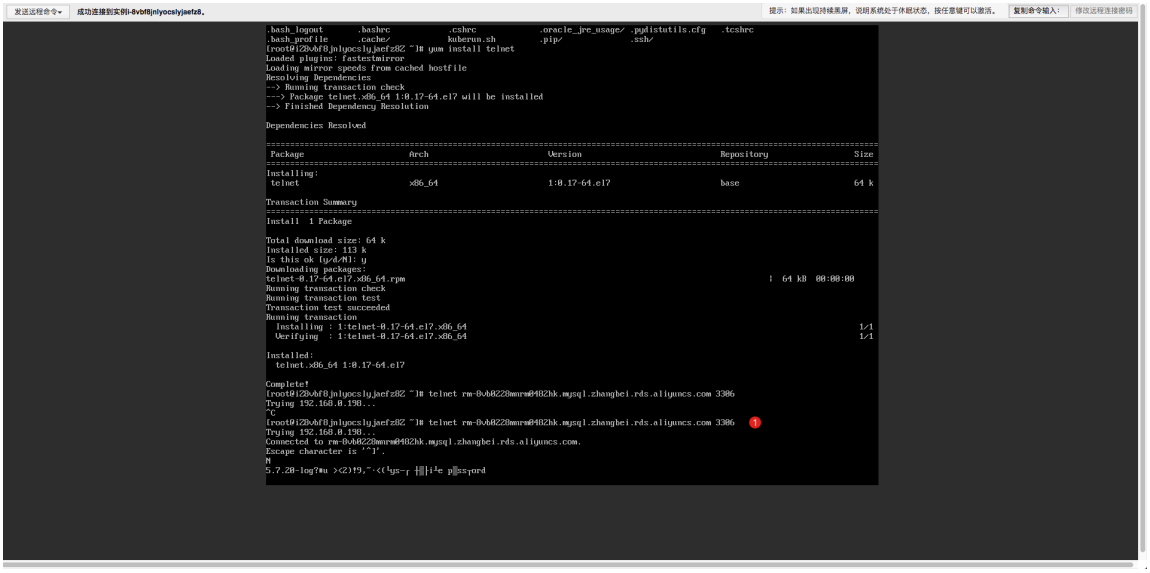


- RDS白名单添加失败。并没有将当前创建Kubernetes 托管版集群的ECS实例添加到RDS的白名单列表中；您可以登录RDS控制台，在RDS实例 > 数据安全性上，设置访问白名单，具体参见设置白名单。

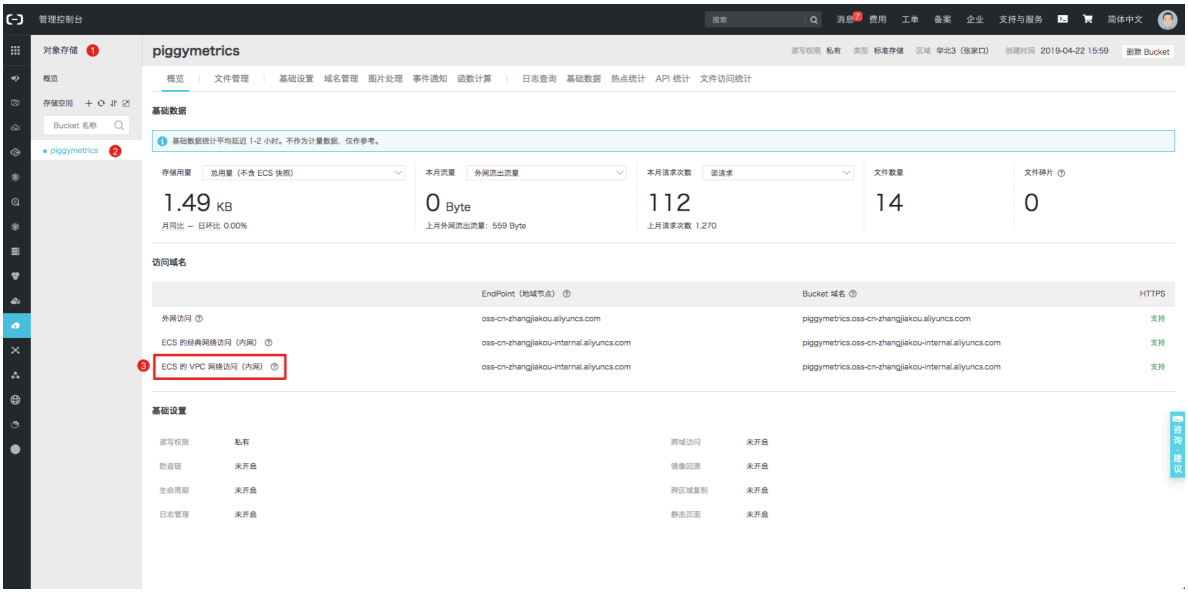


- 其他问题请参见解决无法连接实例问题。

请排查并修改完成后，再登录ESC管理控制台，确保网络已连通后，进行下一步操作。



4. 登录OSS 管理控制台，可以看到针对经典网络和VPC网络有各自对应的域名地址。



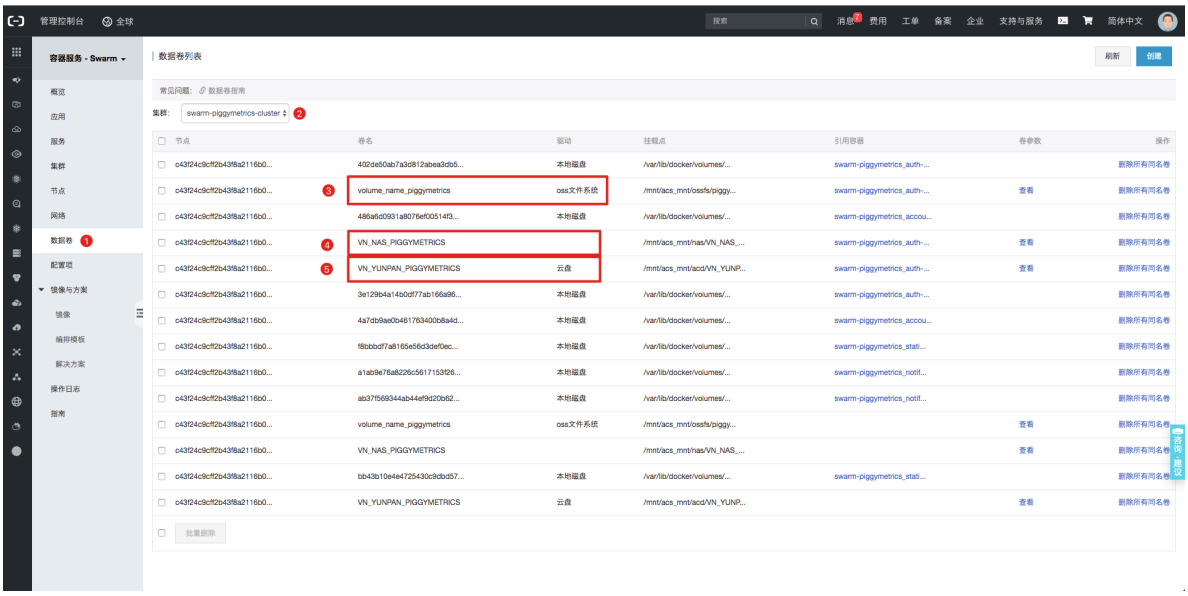
如果swarm-piggymetrics-cluster集群通过经典网络域名访问OSS的，后续需修改为通过VPC网络域名访问（后面集群数据卷迁移时也需要用到）。

数据卷迁移

Swarm数据卷，对应到 Kubernetes 的存储卷都是挂在集群维度下面的；所以，可按集群维度先统一做数据卷配置本身的迁移；当后面应用迁移时，只需做数据卷引用迁移即可。一个Swarm数据卷，对应到Kubernetes的存储卷PV和存储声明 PVC 两个概念。

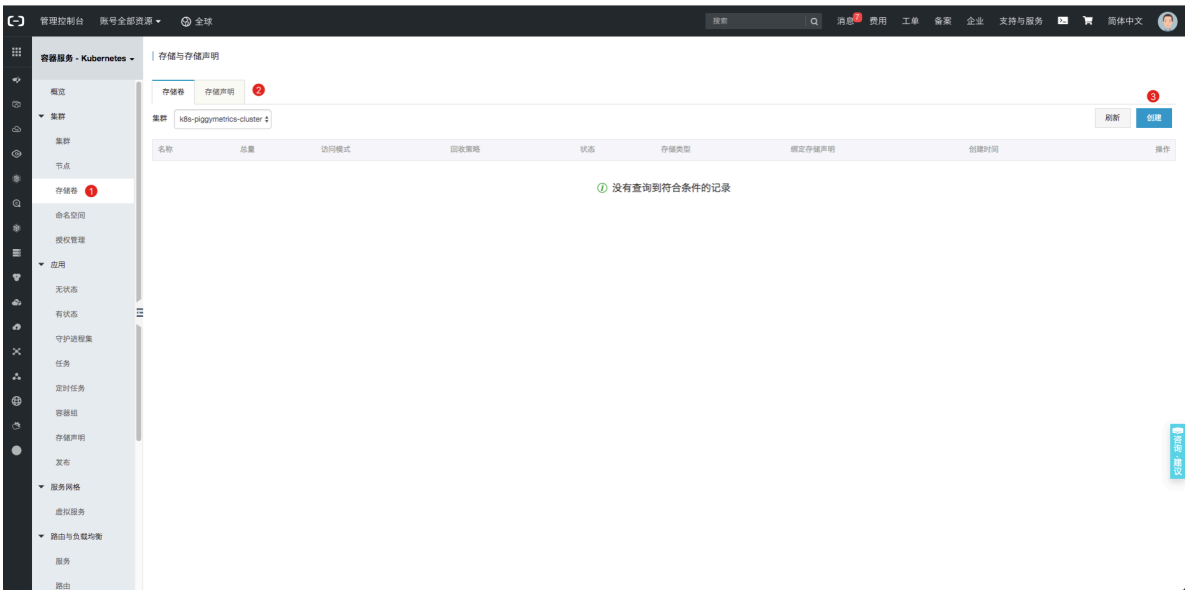
Swarm数据卷支持NAS、云盘、OSS 三种类型，而这三种类型在中使用方式上均支持直接使用Volume方式、使用 PV/PVC 两种方式来使用存储卷（详细参见存储管理概述），本文档中以使用 PV/PVC使用方式为示例。

1. 登录容器服务Swarm控制台，在左侧导航栏单击数据卷，集群选择`swarm-piggymetrics-cluster`，在数据卷列表中分别找到NAS、云盘、OSS这三种类型的数据卷。



2. 在创建的Kubernetes托管版集群中，依次创建这三类数据卷对应的存储卷PV和存储声明PVC。分别可以参见如下文档：

   - 使用阿里云云盘
   - 使用阿里云 NAS
   - 使用阿里云 OSS

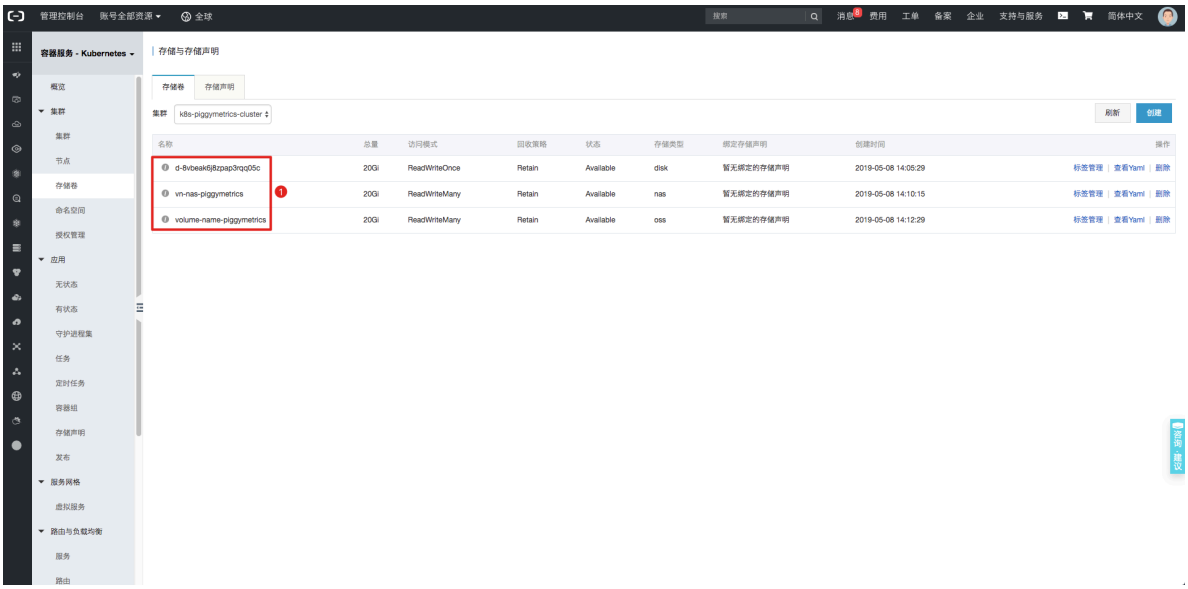


数据卷迁移说明

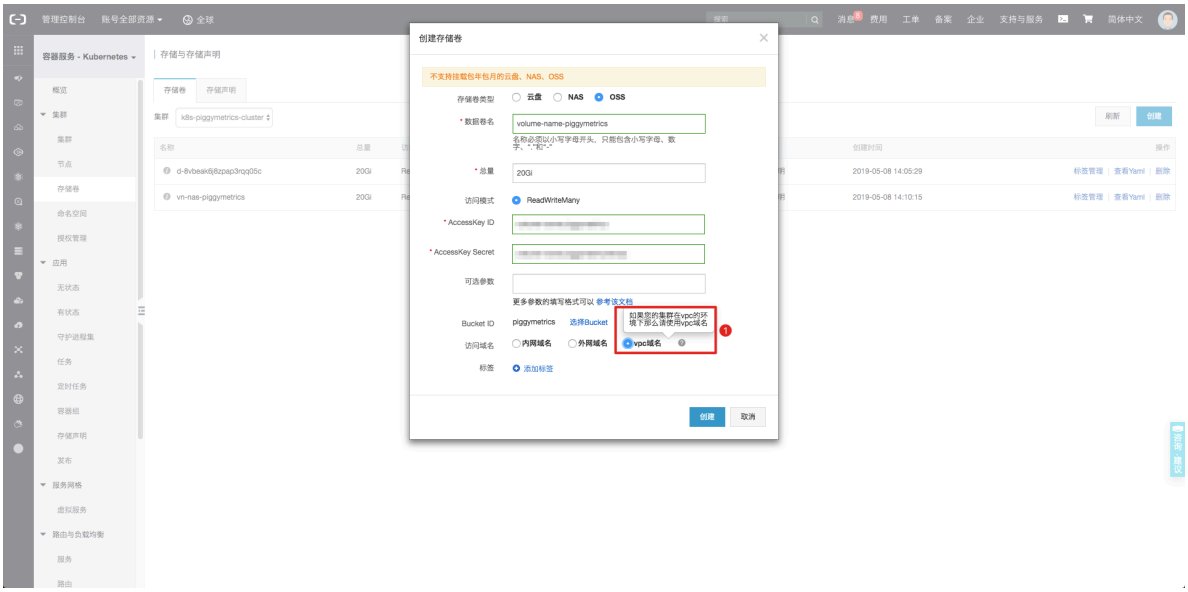
- 因为云盘是阿里云存储团队提供的非共享存储，只能同时被一个pod挂载；若Swarm和Kubernetes 一起写入会报错。因此，若在Swarm集群里面存在使用云盘的场景则无法做

到不停机迁移，请选择切换到共享存储NAS/OSS或在业务低峰期做停机迁移。只有将云盘先从Swarm集群卸载之后，才能在Kubernetes集群里面挂载。云盘操作详见卸载数据盘和查看或删除数据卷。

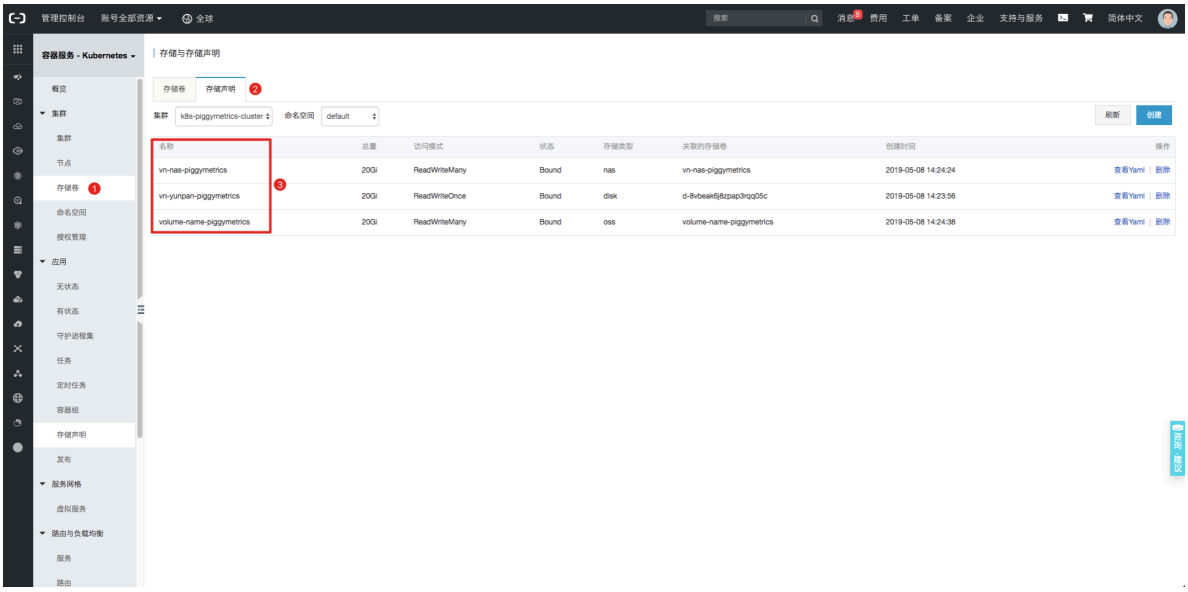
- Kubernetes集群中，数据卷名不支持大写字母和下划线；所以，如果数据卷名存在大写字母或下划线，请做如下统一转换：
  - 大写字母请转换为小写字母。
  - 下划线请转换为中划线。



- 在创建OSS存储卷时，访问域名需要选择为vpc域名，因为Kubernetes只支持VPC网络。



- 创建存储声明PVC，名称请默认与数据卷名一致（后面应用配置转换工具kompose工具会与此保持一致）。



配置项迁移

在Swarm里面，支持创建配置项，并通过配置参数传递配置，以方便对多个容器的环境变量进行统一管理，在应用发布时通过选择配置文件进而实现替换编排模板yaml文件里面的变量符$。

但是阿里云容器服务Swarm控制台提供的增强功能，在标准Kubernetes 里面并没有完全对应的功能实现。其中，Kubernetes集群的配置项实体与此有差异，暂无法实现自动迁移；只能在迁移应用配置概述章节，手动处理Swarm编排模板yaml文件里面的变量符$。

10.5 迁移应用配置

10.5.1 迁移应用配置概述

本文将介绍在运维人员完成集群配置迁移后，开发人员如何将Swarm集群里面的应用配置迁移到Kubernetes 集群。其主要包括准备应用迁移环境、预处理和转换应用的Swarm编排文件；然后部署转换得到的Kubernetes 资源文件，针对其中不支持自动转换的应用配置做手动迁移，最后再逐一调试解决应用启动过程中暴露的问题。

概念对比

Swarm 和 Kubernetes 的概念较多，且两者在应用、服务、访问方式上均有较大不同；二者实体概念的差异请参见概念比对。

前提条件

您已经存在一个待迁移Swarm应用，且已完成集群维度配置迁移，这里以swarm-piggymetrics应用作为示例，通过迁移应用配置，将其部署到k8s-piggymetrics-cluster 集群上。

其中，swarm-piggymetrics是个微服务架构应用，其来源于PiggyMetrics。PiggyMetrics是github上的一个SpringCloud应用项目。在本文中我们对其做了部分增强并引入更多Swarm标签，以便更好体现整个应用配置迁移过程。

迁移应用配置流程

1. 准备迁移环境。
2. 预处理Swarm编排文件。
3. 转换Swarm编排文件。
4. 部署 Kubernetes 资源文件。
5. 手动迁移应用配置。
6. 应用启动调试。
7. 迁移应用日志配置。

相关参考

迁移过程中常见问题，请参见应用配置迁移异常解决方案。

关于标签的映射关系，请参见附录：标签映射列表。

10.5.2 准备迁移环境

背景信息

在做应用配置迁移时，kompose是个开源工具，可以一键将Swarm compose文件转换为Kubernetes 资源文件；阿里云在些基础上针对阿里云定制标签做了部分增强。

当我们通过kompose工具转换得到Kubernetes 资源文件后，可以通过 kubectl 连接Kubernetes 集群，并将资源文件部署到Kubernetes 集群。

由于每个应用配置迁移中均需用到kompose 和 kubectl工具，建议找一台独立ECS部署，避免每次迁移时都需要重新准备迁移环境。

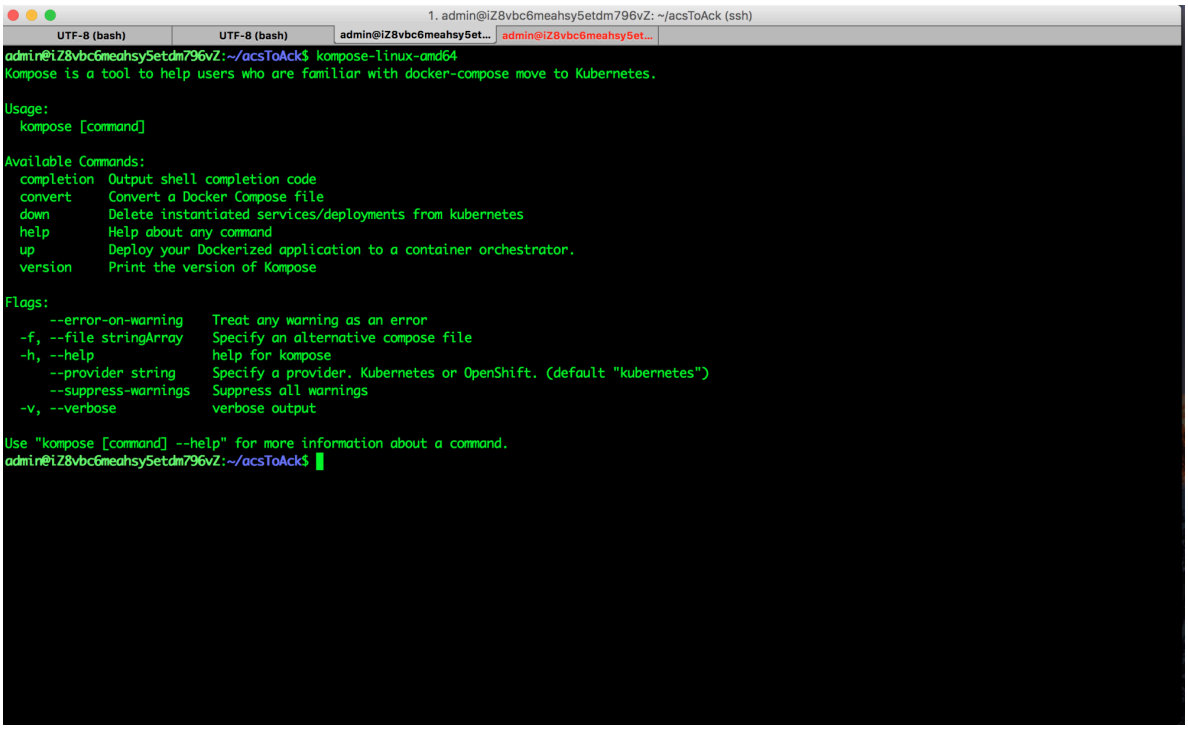
操作步骤

1. 安装kompose工具。

   kompose工具可以一键将Swarm compose文件转换为Kubernetes 资源文件；阿里云在其基础上针对阿里云定制标签做了部分增强。阿里云kompose工具项目参见：AliyunContainerService/kompose。

   安装过程：直接到github上按OS类型下载最新版本可执行文件：https://github.com/AliyunContainerService/kompose/releases

```
kompose-linux-amd64
```



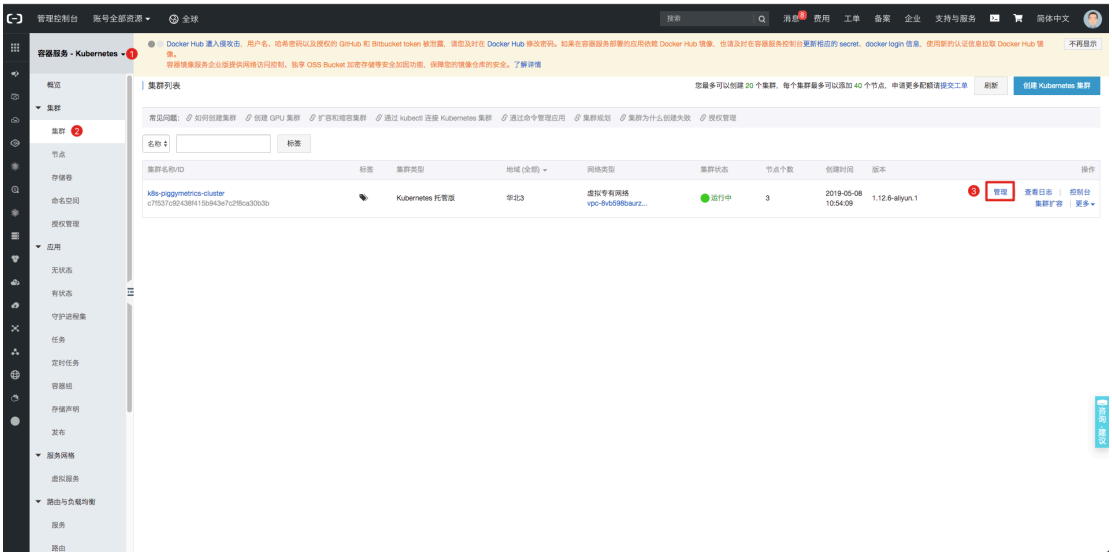
2. 配置kubectl环境。

   a) 从 Kubernetes 版本页面下载最新版 kubectl 客户端。

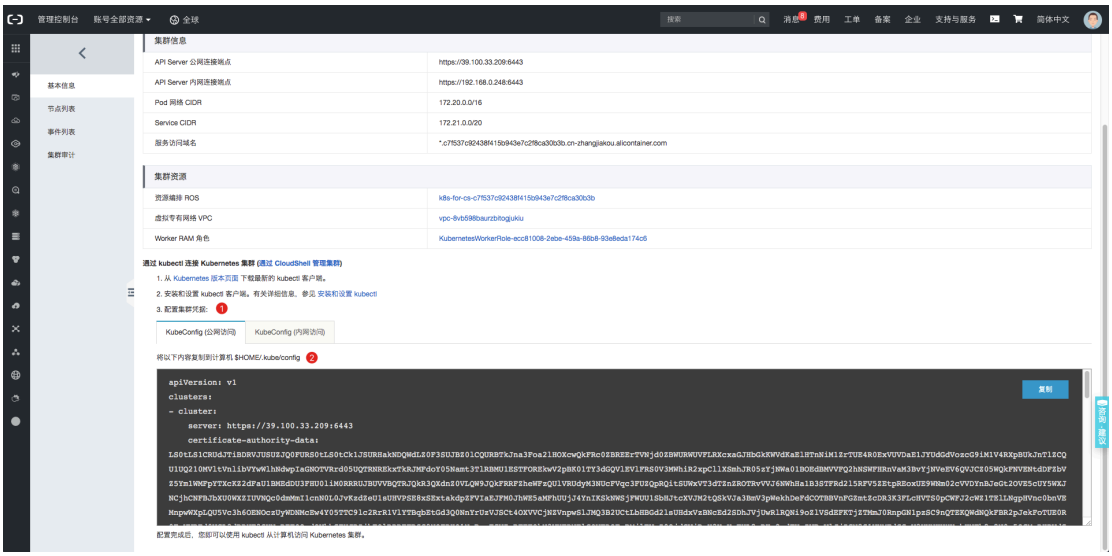
   b) 安装和设置 kubectl 客户端。有关详细信息，参见安装和设置 kubectl。

   c) 配置Kubernetes 集群票据。

      A. 容器服务管理控制台，单击集群 > 集群，在目标集群右侧单击管理。



      B. 在基本信息页签，将KubeConfig的内容粘贴到计算机*$HOME/.kube/config*。



   d) 安装配置成功后，可以通过如下命令测试下是否安装成功，及集群票据配置是否成功。

```
kubectl version

kubectl cluster-info
```

预期结果

当出现如下红框内容时，表示kubectl环境准备完成。

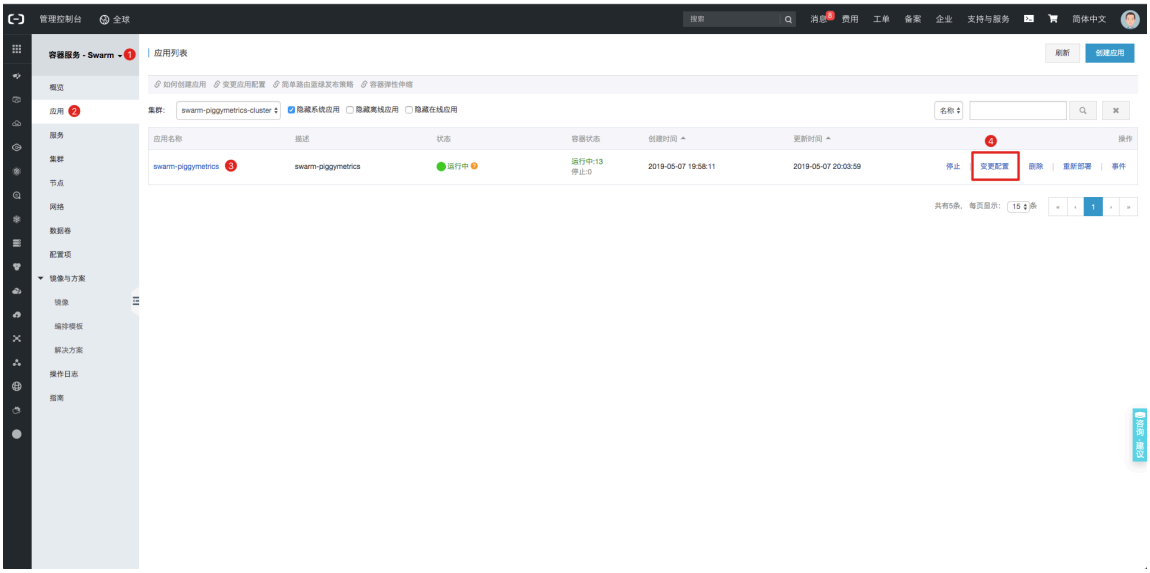
```
UTF-8 (bash)   UTF-8 (bash)   admin@iZ8vbc6meahsy5et...   admin@iZ8vbc6meahsy5et...
e:"2019-04-08T17:11:31Z", GoVersion:"go1.12.1", Compiler:"gc", Platform:"linux/amd64"}
The connection to the server localhost:8080 was refused - did you specify the right host or port?
admin@iZ8vbc6meahsy5etdm796vZ:~$ vim ~/.kube/config
admin@iZ8vbc6meahsy5etdm796vZ:~$ pwd
/home/admin
admin@iZ8vbc6meahsy5etdm796vZ:~$ ll
total 56
drwxr-xr-x 7 admin admin 4096 May  8 17:02 ./
drwxr-xr-x 3 root  root  4096 Apr 16 15:36 ../
drwxrwxr-x 2 admin admin 4096 May  8 15:57 acsToAck/
-rw------- 1 admin admin 2993 Apr 16 22:02 .bash_history
-rw-r--r-- 1 admin admin  220 Apr 16 15:36 .bash_logout
-rw-r--r-- 1 admin admin 4234 May  8 16:42 .bashrc
drwxrwxr-x 2 admin admin 4096 May  8 17:00 chsbin/
drwxrwxr-x 3 admin admin 4096 Apr 16 16:12 code/
drwxrwxr-x 4 admin admin 4096 Apr 16 16:20 .embedmongo/
drwxrwxr-x 3 admin admin 4096 Apr 16 16:14 .m2/
-rw-r--r-- 1 admin admin  655 Apr 16 15:36 .profile
-rw-r--r-- 1 admin admin    0 Apr 16 15:49 .sudo_as_admin_successful
-rw------- 1 admin admin 1048 May  8 17:02 .viminfo
-rw-rw-r-- 1 admin admin  167 May  8 16:36 .wget-hsts
admin@iZ8vbc6meahsy5etdm796vZ:~$ cd ^C
admin@iZ8vbc6meahsy5etdm796vZ:~$ vim ~/.kube/config
admin@iZ8vbc6meahsy5etdm796vZ:~$ mkdir .kube
admin@iZ8vbc6meahsy5etdm796vZ:~$ vim ~/.kube/config
admin@iZ8vbc6meahsy5etdm796vZ:~$ vim ~/.kube/config
admin@iZ8vbc6meahsy5etdm796vZ:~$ kubectl version
Client Version: version.Info{Major:"1", Minor:"14", GitVersion:"v1.14.1", GitCommit:"b7394102d6ef778017f2ca4046abbaa23b88c290", GitTreeState:"clean", BuildDat
e:"2019-04-08T17:11:31Z", GoVersion:"go1.12.1", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"12+", GitVersion:"v1.12.6-aliyun.1", GitCommit:"8cb561c", GitTreeState:"", BuildDate:"2019-04-22T11:34:20Z", Go
Version:"go1.10.8", Compiler:"gc", Platform:"linux/amd64"}
admin@iZ8vbc6meahsy5etdm796vZ:~$ kubectl cluster-info
Kubernetes master is running at https://39.100.33.209:6443
metrics-server is running at https://39.100.33.209:6443/api/v1/namespaces/kube-system/services/heapster/proxy
KubeDNS is running at https://39.100.33.209:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
admin@iZ8vbc6meahsy5etdm796vZ:~$
```

10.5.3 预处理Swarm编排文件

操作步骤

1. 下载Swarm应用编排文件。

   a) 登录容器服务Swarm控制台，在左侧导航栏单击应用，在目标应用右侧单击变更配置。

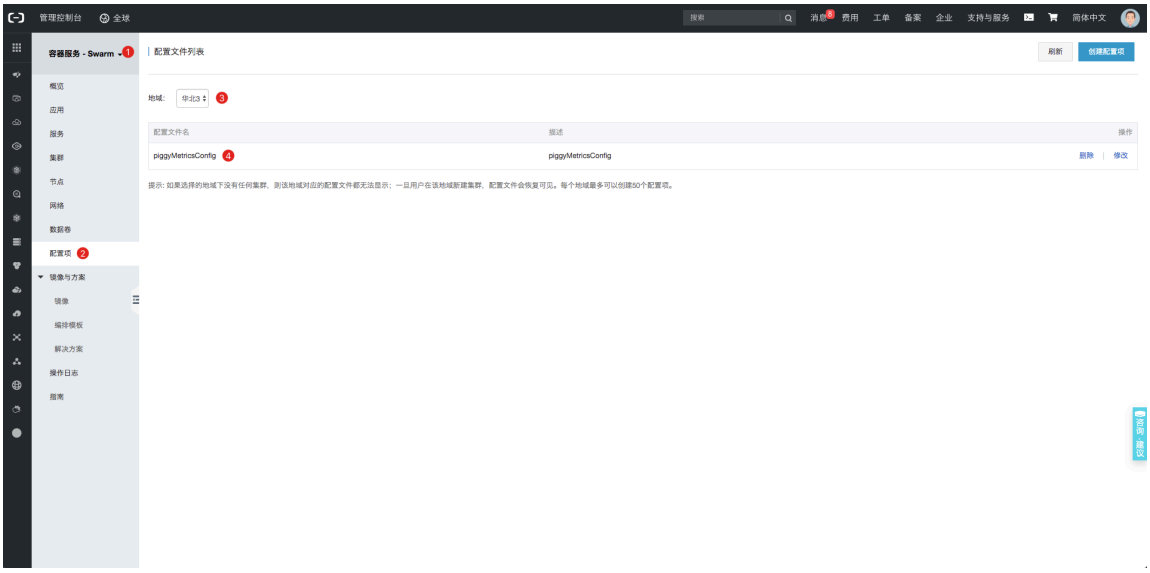


   b) 拷贝编排文件并保存到本地，文件名后缀为`.yaml`，本文命名为`swarm-piggymetrics.yaml`。

2. 预处理swarm 编排文件里面环境变量。

   swarm编排文件支持配置项替换，是由阿里云Swarm控制台实现的功能，在Kubernetes 里面没有与其对应的实体支持；需要我们在转换前，手动将编排文件里面的变量符$替换成该配置项真实值。具体的操作如下：

   a. 登录容器服务Swarm控制台，单击配置项，选择对应的地域，找到目标配置文件。



   b. 参考该配置文件，手动将编排文件`swarm-piggymetrics.yaml`里面的变量符$替换成该配置项真实值。

3. 预处理部分标签。

   您需要预处理下部分 Swarm 标签，否则kompose 因格式或不支持等原因，会报错并中断整个转换过程。标签的预处理主要包含以下 两 个方面，具体针对每个标签的详细操作请参见附录：标签映射列表。

   - 以下标签的值需要将布尔型修改为字符串类型，如：true/false 修改为"true"/"false"
     - aliyun.global
     - aliyun.latest_image
   - 包含以下标签的 service 无法做迁移，需要先收到删除，后续再人工迁移。
     - external

4. 修改编排文件版本号。

   因kompose工具目前主要支持V2版本的compose文件；所以，需要将文件头的版本声明从version: '2.X' 修改为version: '2'；否则转换时会报错。

10.5.4 转换Swarm编排文件

操作步骤

1. kompose工具转换文件。

   在对Swarm编排文件预处理之后，接下来可以用kompose工具做转换，操作命令如下：

```
kompose-linux-amd64 convert -f source/swarm-piggymetrics.yaml --
volumes PersistentVolumeClaimOrHostPath
```

```
admin@iZ8vbc6meahsy5etdm796vZ:~/acsToAck$ kompose convert -f source/swarm-piggymetrics.yaml --volumes PersistentVolumeClaimOrHostPath
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported aliyun.routing.session_sticky label key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported aliyun.depends label key - ignoring
WARN Unsupported links key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported name key - ignoring
WARN Unsupported memswap_limit key - ignoring
WARN Unsupported shm_size key - ignoring
WARN Unsupported memswap_reservation key - ignoring
WARN Unsupported kernel_memory key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported links key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported depends_on key - ignoring
WARN Unsupported aliyun.log.timestamp label key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported external_links key - ignoring
WARN Unsupported hostname key - ignoring
WARN Unsupported logging key - ignoring
WARN Unsupported devices key - ignoring
INFO Kubernetes file "gateway-service.yaml" created
INFO Kubernetes file "monitoring-service.yaml" created
INFO Kubernetes file "rabbitmq-service.yaml" created
INFO Kubernetes file "registry-service.yaml" created
INFO Kubernetes file "turbine-stream-service-service.yaml" created
INFO Kubernetes file "account-mongodb-deployment.yaml" created
INFO Kubernetes file "account-service-deployment.yaml" created
INFO Kubernetes file "auth-mongodb-deployment.yaml" created
INFO Kubernetes file "auth-service-deployment.yaml" created
INFO Kubernetes file "config-deployment.yaml" created
INFO Kubernetes file "gateway-deployment.yaml" created
INFO Kubernetes file "monitoring-deployment.yaml" created
INFO Kubernetes file "notification-mongodb-deployment.yaml" created
INFO Kubernetes file "notification-service-deployment.yaml" created
INFO Kubernetes file "rabbitmq-deployment.yaml" created
INFO Kubernetes file "registry-deployment.yaml" created
INFO Kubernetes file "statistics-mongodb-deployment.yaml" created
INFO Kubernetes file "statistics-service-deployment.yaml" created
INFO Kubernetes file "turbine-stream-service-deployment.yaml" created
admin@iZ8vbc6meahsy5etdm796vZ:~/acsToAck$
```

说明:

正常情况下，会生成对应Kubernetes 资源文件，针对不支持自动转换的标签，会有WARN提示，这部分标签请按后面的转换异常列表进行处理，其一般会有收下3种处理方式：

- 手动调整原Swarm 编排文件，然后重新通过kompose转换。
- 手动修改kompose工具转换好的Kubernetes 资源文件。
- 暂时忽略，先部署转换好的Kubernetes 资源文件，再通过容器服务控制台手动迁移配置。

2. 手动修复转换问题。

   kompose工具支持转换大部分标签，但少量Swarm标签无法支持自动转换（后续我们会进一步增强），需要我们根据kompose工具输出的警告信息，手动调整Swarm编排文件并重新用kompose工具转换；具体各类异常信息及对应解决方案参见应用配置迁移异常解决方案。

3. 手动完善 Kubernetes 资源文件。

   在通过 kompose工具转换之后，部分kompose 无法支持的标签，需要我们手动编写或完善Kubernetes 资源文件。主要包括以下几类标签：

   - aliyun.routing.port_
   - aliyun.global
   - external
   - environment: constraint
   - extra_hosts
   - net
   - dns

   针对每个标签的详细操作请参见附录：标签映射列表。

10.5.5 部署 Kubernetes 资源文件

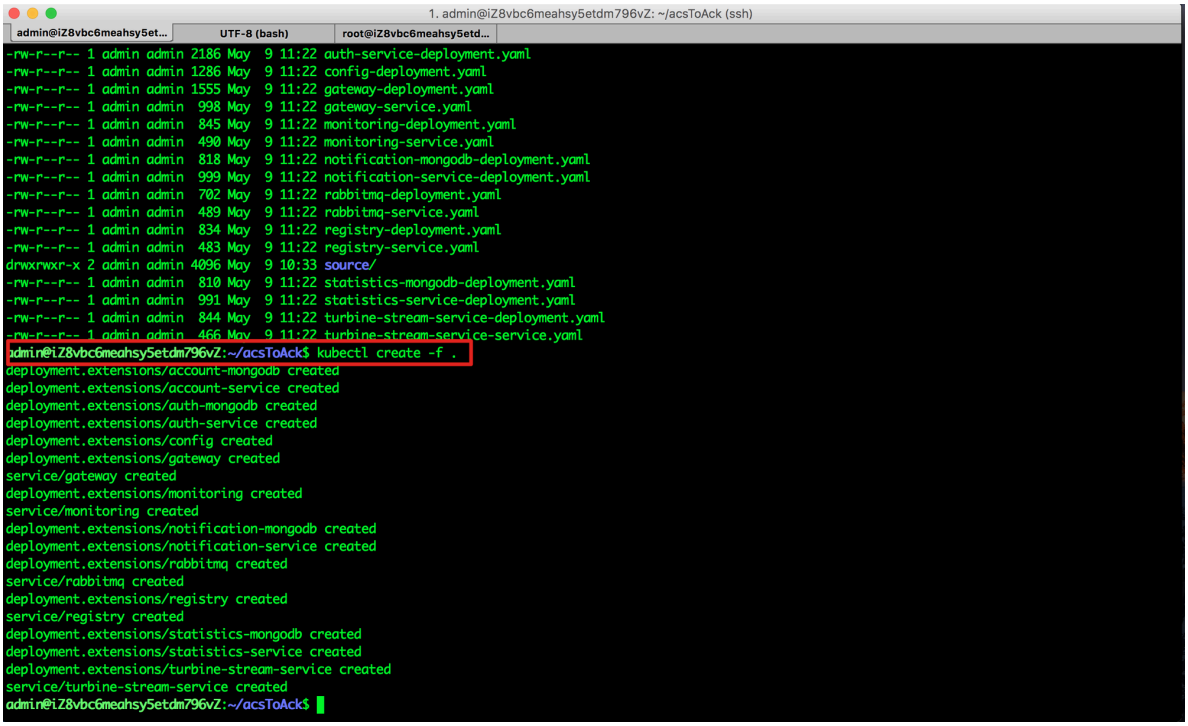
背景信息

针对前面kompose 转换成功并预处理后的Kubernetes 资源文件，我们可以通过kubectl工具部署到Kubernetes 集群中；其中，集群票据在配置kubectl环境 已配置完成。

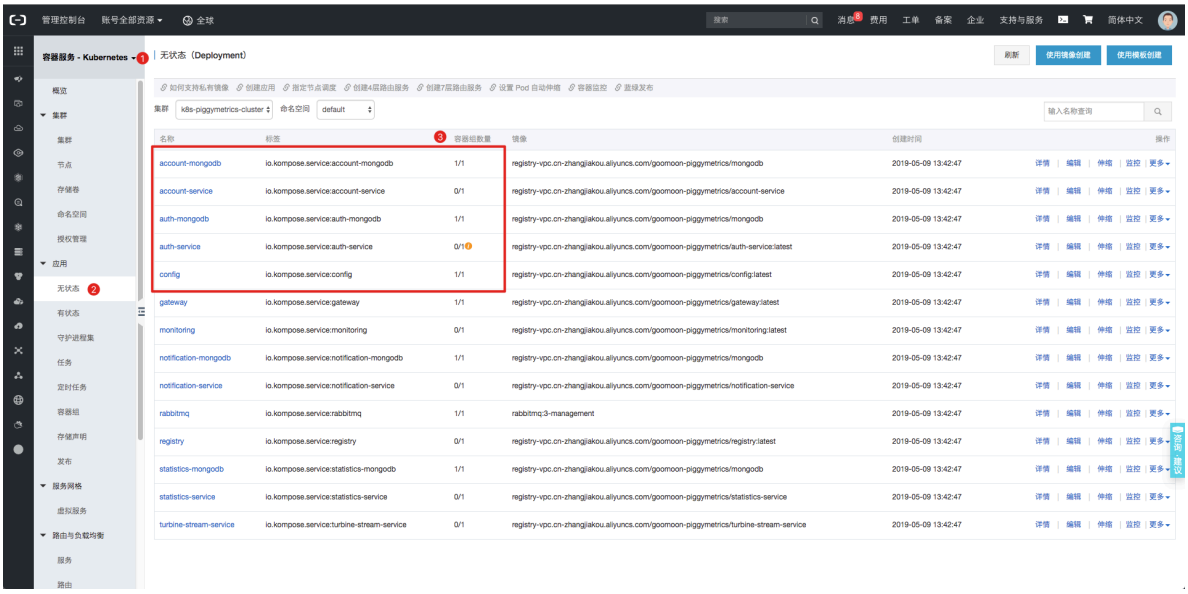
操作步骤

1. 可通过如下命令，批量部署当前目录下的所有资源文件。

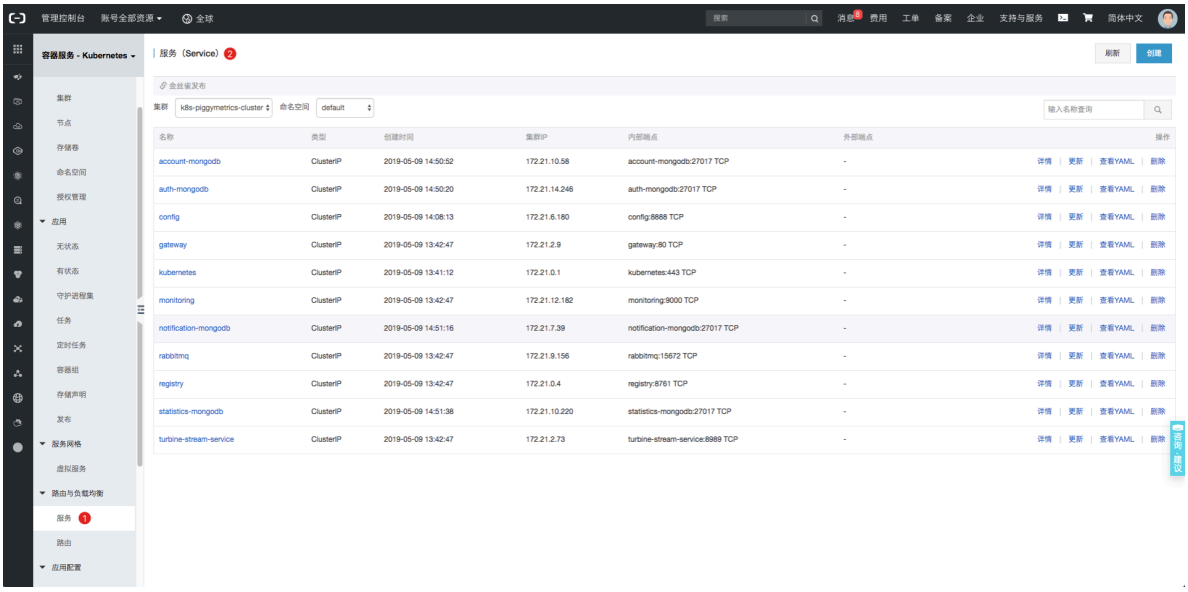
```
kubectl create -f .                    #通过文件名或标准输入创建资源，.表示基于当前目录下的资源文件创建
```



2. 登录容器服务管理控制台，在左侧导航栏选择应用 > 无状态，查看应用部署情况。



3. 在路由与负载均衡下，单击路由或者服务，查看服务及路由配置情况。



10.5.6 手动迁移应用配置

针对少量kompose工具无法支持的Swarm标签，需要您登录容器服务管理控制台进行手动迁移。主要包括以下几类标签，针对每个标签的详细操作请附录：标签映射列表。

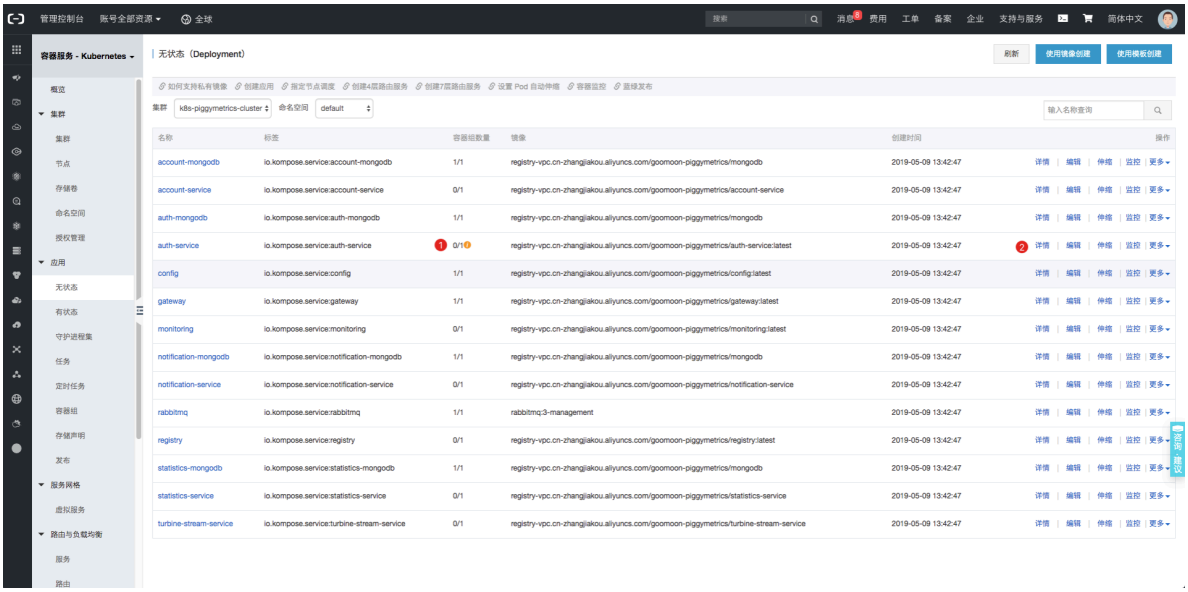
- aliyun.auto_scaling.
- hostname
- links
- external_links
- aliyun.lb.port_
- aliyun.log_ttl_

10.5.7 应用启动调试

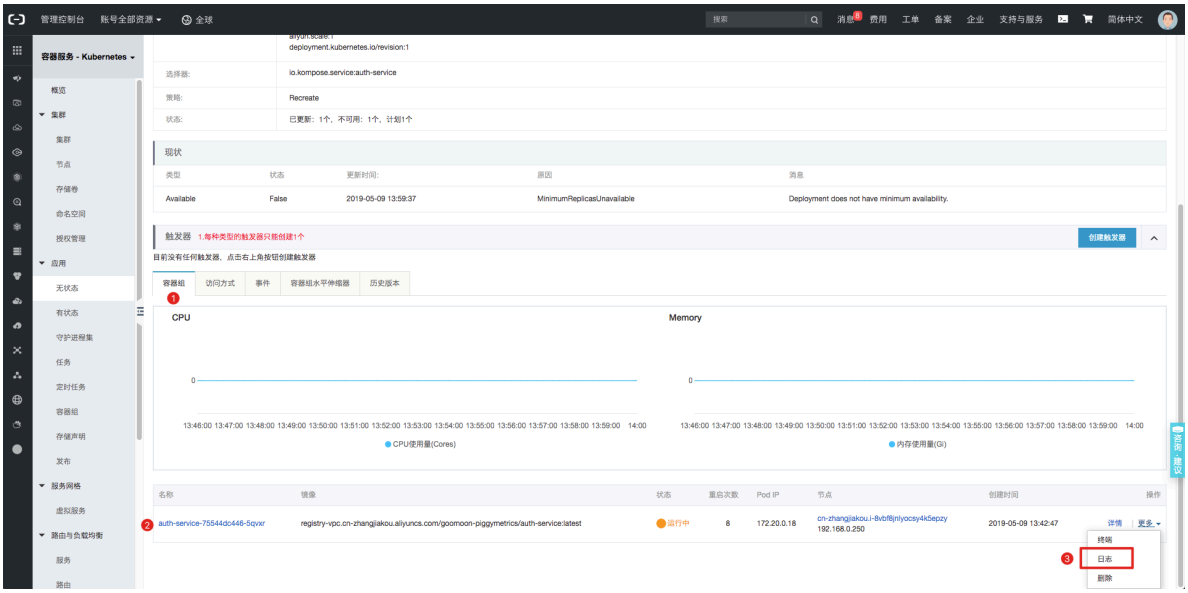
查看应用启动情况

通过kompose工具或手动完成应用配置迁移之后，接下来我们可以通过控制台查看应用是否正常启动，或在启动过程中是否有问题。我们可以通过K8S控制台查看应用启动日志定位问题。

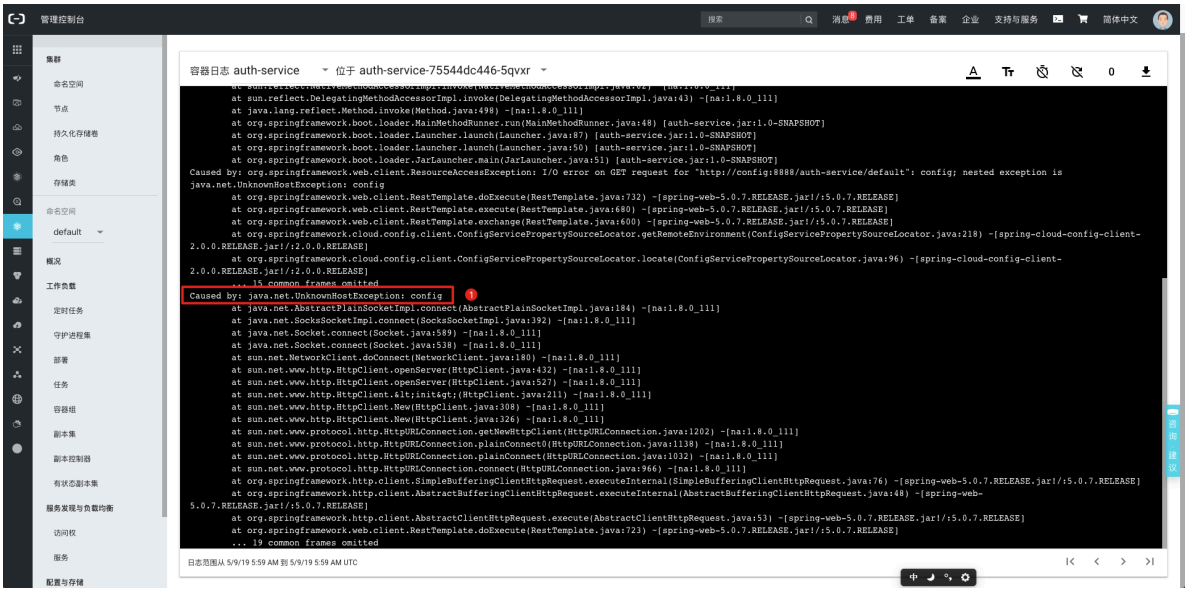
1. 登录容器服务管理控制台，在左侧导航栏选择应用 > 无状态，选择对应的集群及命名空间，在目标应用（即，运行状态异常的应用）右侧单击详情。



2. 在容器组页签，在目标容器右侧选择详情 > 日志。



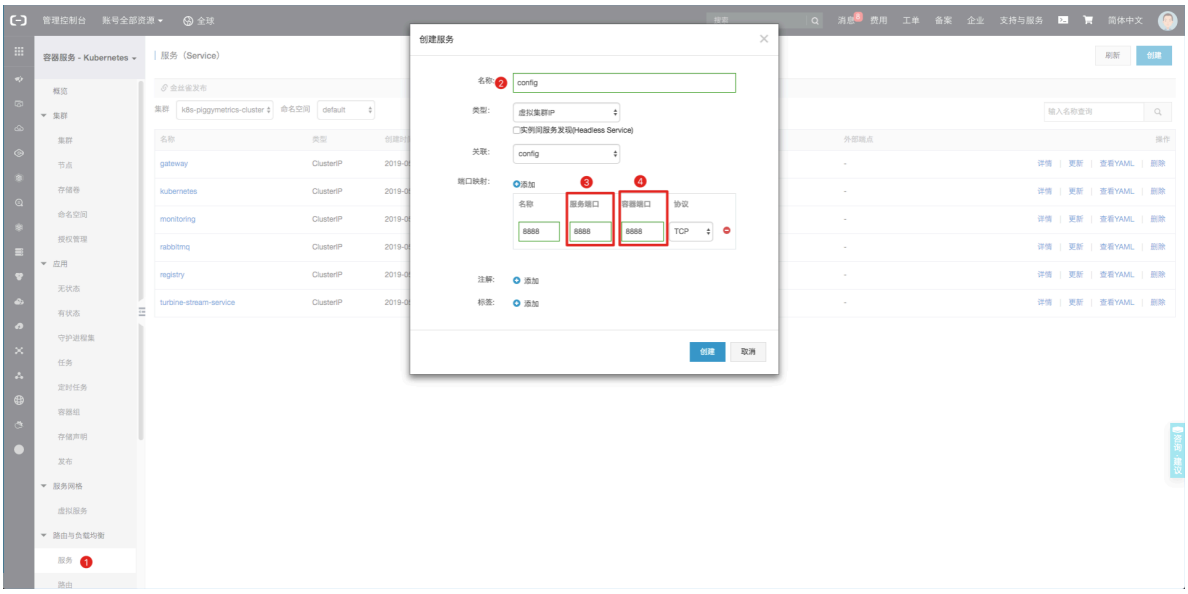
3. 在容器日志页面，查看日志内容。



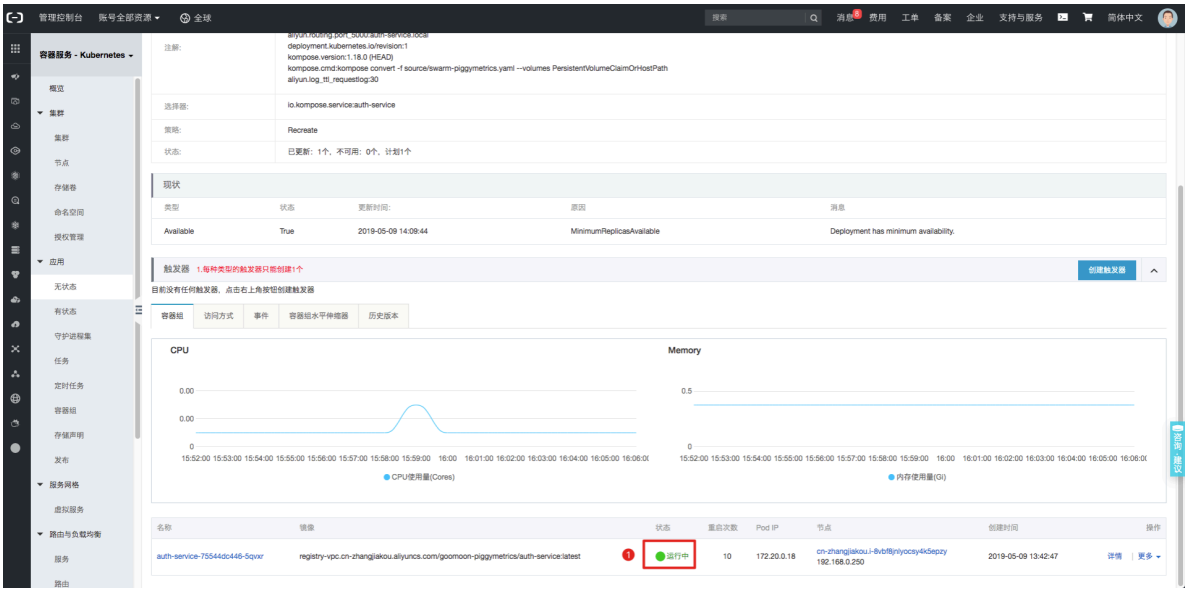
修复应用启动问题

针对应用启动出现的问题，我们需要逐个分析并修复。例如，上述日志显示故障原因为auth-service这个应用无法访问config这个域名。解决方案如下：

1. 通过手动创建一个Kubernetes集群的服务，来支持其它应用访问config应用即可，其中下图中的标识2表示服务名，必须要跟hostname保持一致，其他像端口映射没有特殊要求，可以将端口名称、服务端口设置成跟容器端口一致即可。详细操请参见创建服务。



2. 修复完成后，过一会容器重新部署之后再确认问题已修复，如下图所示表示容器状态为运行中，表示容器已正常启动。

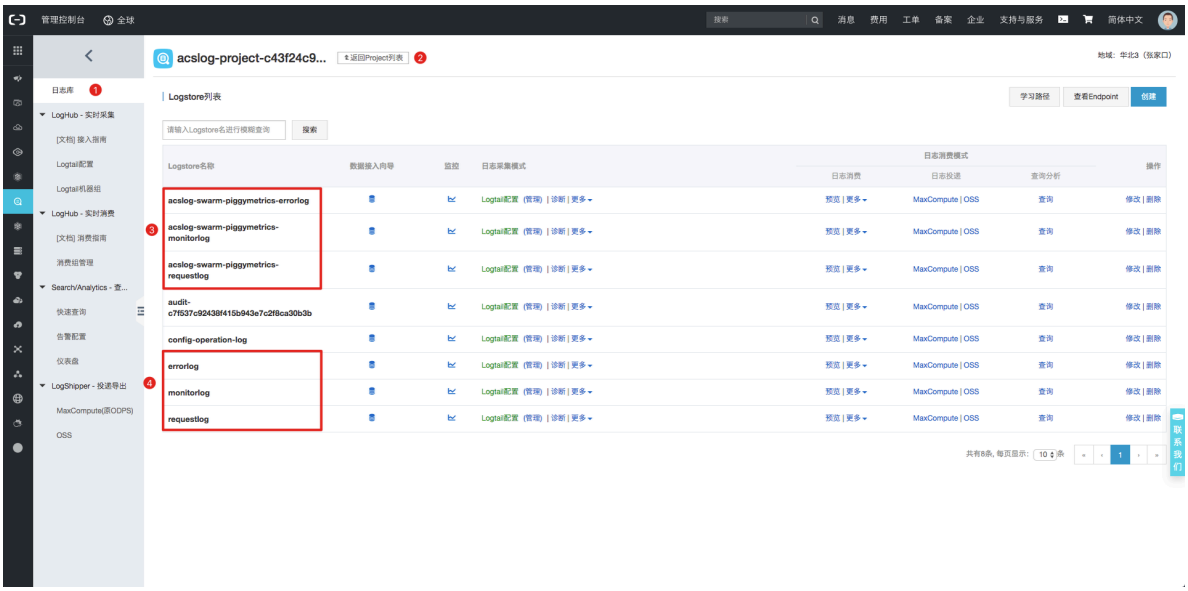


各个业务在应用启动过程中可能会出现各种异常，具体各类异常信息及对应解决方案参见应用配置迁移异常解决方案。

10.5.8 迁移应用日志配置

如果您的应用里面使用到阿里云日志服务，则在迁移时，需要确认Kubernetes集群的日志输出是否正常，以及基于日志服务上的云监控等产品是否正常。

1. 登录日志服务控制台，单击Project名称。
2. 在日志库页签，在Logstore列表中，可以看到对应的Logstore。



说明：

我们在创建Kubernetes集群时，复用了已有Swarm集群的日志Project，并通过kompose工具自动完成应用日志配置的迁移。但由于Swarm容器服务和K8S容器服务对于Logstore的生成规则不一样，Kubernetes的应用在对应日志Project下面，还是会生成不同的Logstore。

- 图中的标注3为Swarm容器服务自动生成的Logstore，其生成格式为`acslog-${app}-${name}`。
- 图中的标注4则为Kubernetes容器自动生成的Logstore，其生成格式为`${name}`，并与Swarm容器服务Logstore一一对应。
- 由于Kubernetes的Logstore不支持大写，所以kompose转换工具在转换时，会自动将大写字母转换化小写字母。

因此，所有底层依赖于日志服务，通过swarm容器服务创建的Project采集应用日志的其它阿里云产品的配置均需要做配置迁移或切换。目前主要包含以下几类，具体使用方式参见日志服务-实时消费完成日志切换和配置变更：

- 实时消费： SDK、Storm Spout、Spark Client、Web Console、云监控、业务实时监控ARMS等。
- 实时查询：用于日志的实时查询分析。
- 投递存储：通过将日志投递到种类存储，用于后续加工。
- 安全日志服务：日志服务与安全云产品对接，可通过 ISV 消费云产品日志。

10.5.9 应用配置迁移异常解决方案

您可通过本文了解应用配置迁移过程中的常见问题及解决方法。

文件版本不正确

异常输出

```
FATA Version 2.1 of Docker Compose is not supported. Please use
 version 1, 2 or 3
```

异常原因

kompose工具只支持V1、V2、V3版本的compose格式，不支持V2.X版本；导致整个转换中断。

解决方案

将version: '2.X' 修改为version: '2'，重新转换。

标签解析异常

- 异常输出

```
ERRO Could not parse config for project source : Unsupported config
 option for account-db service: 'external'
```

异常原因

kompose 工具无法解析external标签，导致转换中断。

解决方案

出现 ERRO或 FATA 级别的错误，修改原编排文件，移除配置，后续手动迁移配置，迁移方式参见附录：标签映射列表。

- 异常输出

```
ERRO Could not parse config for project source : Unsupported config
 option for gateway service: 'net'
```

异常原因

kompose 工具无法解析 net 标签，导致转换中断。

解决方案

修改原编排文件，移除配置，后续手动迁移配置，迁移方式参见附录：标签映射列表。

标签值格式不正确

- 异常输出

```
ERRO Could not parse config for project source : Service 'auth-
service' configuration key 'labels' contains an invalid type, it
 should be an array or object Unsupported config option for auth-
service service: 'latest_image'
```

异常原因

根据提示是latest_image配置值格式无效，kompose 工具无法转换。

解决方案

修改原编排文件，修复值配置；将bool 值修改成字符串形式，true 修改成'true'。

> 说明:
> 以下标签列表均有此问题：
>
> - aliyun.latest_image: true
> - aliyun.global: true

- 异常输出

```
ERRO Could not parse config for project source : Cannot unmarshal
 '30' of type int into a string value
```

异常原因

值类型不对，查看下compose文件*aliyun.log_**标签值是否为30，但未带单引号"，因为其对应结构体应该是string不是int所以得加引号。

解决方案

修改原编排文件，修复值配置，然后重新转换； 将30修改成'30'。

标签不支持提醒

- 异常输出

```
WARN Unsupported hostname key - ignoring
```

异常原因

Swarm里面，支持 hostName 作为域名来访问服务，而在compose文件里面并没有声明对应的容器端口，使得kompose工具无法自动创建对应的Kubernetes 服务，导致在Kubernetes里面无法通过hostName来访问应用。

解决方案

先部署资源文件，然后在Kubernetes控制台手动创建Kubernetes 服务，其中服务名称即为hostName；服务端口与容器端口号一致。

- 异常输出

```
WARN Unsupported links key - ignoring
```

异常原因

links 底层跟 hostName 一样，支持通过links名称或别名访问服务；而在 compose 文件里面并没有声明对应的容器端口，使得 kompose 工具无法自动创建对应的 Kubernetes 服务，导致在 Kubernetes 里面无法通过links名称或别名来访问应用。

解决方案

先部署资源文件，然后在Kubernetes控制台手动创建Kubernetes Service，其中Service名称即为links名称或别名；服务端口与容器端口号一致。

标签转换有缺陷

- **异常输出**

```
WARN Handling aliyun routings label: [{rabbitmq.your-cluster-
id.alicontainer.com  http 15672}]
```

异常原因

swarm简单路由里面配置了测试域名(swarm容器服务提供的域名)；其跟集群标识cluster-id有关。因为不知道Kubernetes新集群id，kompose无法自动完成新测试域名的转换，需要手动手动对应的ingress文件，更新域名。

解决方案

找到各ingress文件*-ingress.yaml，将其中的*host: your-cluster-id.alicontainer.com*修改成对应Kubernetes集群的测试域名，您可以通过以下操作查看测试域名：

1. **登录容器服务管理控制台，在左侧导航栏选择集群 > 集群，在目标集群 Kubernetes-piggymetrics-cluster 右侧单击管理。**
2. **在基本信息页面，可以看到测试域名的值为`*.c7f537c92438f415b943e7c2f8ca30b3b.cn-zhangjiakou.alicontainer.com`，将其替换到各个ingress文件的`.your-cluster-id.alicontainer.com`字符串。**

- **异常输出**

```
WARN Handling aliyun routings label: [{gateway.your-cluster-
id.alicontainer.com  http 4000} {gateway.swarm.piggymetrics.com
  http 4000}]
```

异常原因

Swarm 里面简单路由配置多域名，但 kompose 工具转换时只支持单域名转名，后续域名需要自己添加 ingress 规则

解决方案

修改转换后的 Kubernetes 资源文件，修复方式参见附录：标签映射列表。

部署应用时报错

- **异常输出**

```
error: error validating "logtail2-daemonset.yaml": error
 validating data: ValidationError(DaemonSet.status):
 missing required field "numberReady" in
```

```
io.Kubernetes.api.extensions.v1beta1.DaemonSetStatus; if you choose
to ignore these errors, turn validation off with --validate=false
```

异常原因

针对 Swarm 里面的aliyun.global: true 这类服务，kompose 工具会转成 Kubernetes 里面的DaemonSet，但自动转换后的资源文件里面会有status信息，用来记录中间状态；进而导致部署失败。

status:

- **currentNumberScheduled: 0**
- **desiredNumberScheduled: 0**
- **numberMisscheduled: 0**

解决方案

在转换后的 Kubernetes 资源文件*-daemonset.yaml里面，删除 status 信息，再部署应用。

- **异常输出**

```
error: error parsing auth-service-deployment.yaml: error converting
 YAML to JSON: yaml: line 26: did not find expected key
```

异常原因

对应 Kubernetes 资源文件对应行的标签定义不符合要求，解析失败。最常见的是缩进问题，将当前标签误以为上一个标签的子标签，其实应该是兄弟标签等。

解决方案

按照标签列表和 Kubernetes 官方文档，确认当前行的正确配置，并修改 Kubernetes 资源文件。

- **异常输出**

```
error: error parsing auth-service-deployment.yaml: error converting
 YAML to JSON: yaml: line 34: found character that cannot start any
 token
```

异常原因

对应 Kubernetes 资源文件对应行上存在非 token 字符，最常见是tab 按键。

解决方案

在转换后的 Kubernetes 资源文件*-daemonset.yaml里面，对应行 tab 修改成空格键。

启动失败

- 异常输出

  Pod容器一直在重启，并显示健康检查失败。

```
Initialized: True
Ready: False
ContainersReady: False
PodScheduled: True
```

  异常原因

  在 Kubernetes 里面会通过 liveness probe 和 readiness probe 来验证业务容器是否存活或就绪；类似于Swarm里面的 aliyun.probe.url、aliyun.probe.cmd。但 aliyun.probe.url、aliyun.probe.cmd 如果有问题只会标注容器的健康状态，kompose 工具会将其转换为 liveness probe；而在Kubernetes中，如果liveness probe设置有问题，检查失败，则会自动触发重启容器，强制中断原有容器业务的初始化。

  解决方案

  1. 先验证是否为探针设置问题。

     先去掉 liveness probe 或 readiness probe探针，如果应用能启动成功，则说明是探针时间设置有问题。

  2. 修复问题。

     根据业务需要，确认容器真正启动完成耗时，到 Kubernetes 对应应用配置里合理设置 liveness probe 的配置，尤其是其中的延迟探测时间、执行探测频率、超时时间等字段，具体参见镜像创建无状态Deployment应用。

- 异常输出

  Pod容器一直在重启，并显示健康检查失败。

```
Initialized: True
Ready: False
ContainersReady: False
PodScheduled: True
```

  进入容器查看日志如下：

```
2019-05-22 06:42:09.245  INFO [gateway,,,] 1 --- [           main] c
.c.c.ConfigServicePropertySourceLocator : Connect Timeout Exception
```

```
 on Url - http://config:8888. Will be trying the next url if
available
```

异常原因

在这里请求 config:8888 超时，之前是因为该 Pod 的网络模型设置出错，设置了hostNetwork: true，通过ECS 网络栈导致解析不了 Kubernetes service 名称。

但我们已经修复gateway-deployment.yaml里面的配置，并通过kubectl apply重新部署，但还是一直报错。

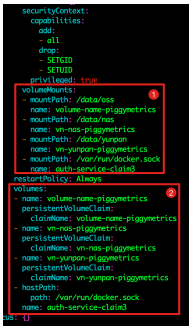
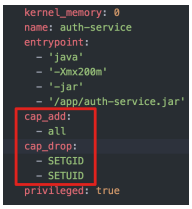
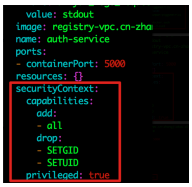
解决方案

这种情况下，有时是因各种原因导致新配置`*-deployment.yaml`没有真正生效。

可以在保存配置后，登录容器服务管理控制台，删除对应的应用，再通过 kubectl 重新部署。

10.5.10 附录： 标签映射列表

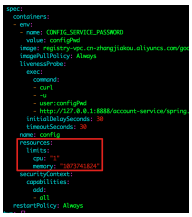
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
name	service 名称 无特殊作用，可忽略	Kubernetes 不支持，迁移忽略	略	略	无法迁移
image	容器镜像	image标签	略	略	kompose 自动迁移
ports	容器端口映射	containerPort标签 若有对外访问诉求，则需要自己配置NodePort	略	略	kompose 自动迁移
environment	容器环境变量	env标签	略	略	kompose 自动迁移

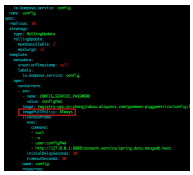
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
volumes	挂载宿主机目录或存储卷	volumeMounts标签 volumes标签		 详细参见auth-service-deployment.yaml配置	kompose 自动迁移
cap_add/cap_drop	增加或移除容器对内核的修改权限	securityContext: capabilities标签		 详细参见 auth-service-deployment.yaml配置	kompose 自动迁移

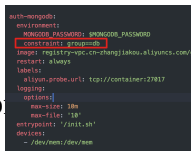
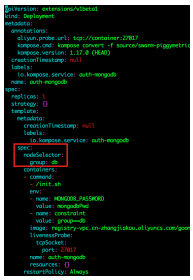
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移

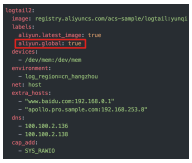
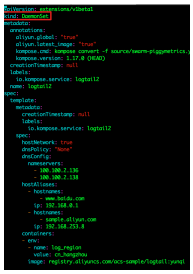
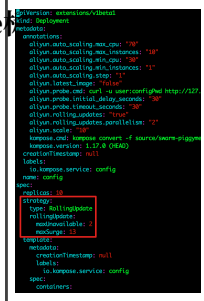
privileged: true	使用该参数，container内的root拥有真正的root权限。 否则，container内的root只是外部的一个普通用户权限。
	privileged启动的容器，可以看到很多host上的设备，并且可以执行mount。甚至允许你在docker容器中启动docker容器。
		securityContext: privileged标签	`name: auth-service`
			`entrypoint:`
			`- 'java'`
			`- '-Xmx200m'`
			`- '-jar'`
			`- '/app/auth-service.jar'`
			`cap_add:`
			`- all`
			`cap_drop:`
			`- SETGID`
			`- SETUID`
			`privileged: true`
				`image: registry-vpc.cn-zhangjiakou.al`
				`name: auth-service`
				`ports:`
				`- containerPort: 5000`
				`resources: {}`
				`securityContext:`
				`capabilities:`
				`add:`
				`- all`
				`drop:`
				`- SETGID`
				`- SETUID`
				`privileged: true`
				`volumeMounts:`
				`- mountPath: /data/oss`
				`name: volume-name-piggymetrics`
				`- mountPath: /data/nas`
				详细参见 auth-service-deployment.yaml配置
					kompose 自动迁移

mem_limit	指定容器对内存资源限制	resource :request/ limits标签	`labels:`
			`aliyun.probe.cmd: >-`
			`curl -u user:configPwd`
			`http://127.0.0.1:8888/account-service/spring.da`
			`aliyun.probe.initial_delay_seconds: '30'`
			`aliyun.probe.timeout_seconds: '30'`
			`aliyun.scale: '10'`
			`aliyun.rolling_updates: 'true'`
			`aliyun.rolling_updates.parallelism: '2'`
			`aliyun.auto_scaling.max_cpu: '70'`
			`aliyun.auto_scaling.min_cpu: '30'`
			`aliyun.auto_scaling.step: '1'`
			`aliyun.auto_scaling.max_instances: '10'`
			`aliyun.auto_scaling.min_instances: '1'`
			`aliyun.latest_image: 'false'`
			`oom-kill-disable: true`
			`memswap_limit: 2000000000`
			`shm_size: 67108864`
			`memswap_reservation: 536870912`
			`kernel_memory: 0`
			`names: config`
			`mem_limit: 1073741824`
			`cpu_shares: 100`
			`cap_add:`
			`- all`
				`spec:`
				`containers:`
				`- env:`
				`- name: CONFIG_SERVICE_PASSWORD`
				`value: configPwd`
				`image: registry-vpc.cn-zhangjiakou.aliyuncs.com/ge`
				`imagePullPolicy: Always`
				`livenessProbe:`
				`exec:`
				`command:`
				`- curl`
				`- -u`
				`- user:configPwd`
				`- http://127.0.0.1:8888/account-service/spring.`
				`initialDelaySeconds: 30`
				`timeoutSeconds: 30`
				`name: config`
				`resources:`
				`limits:`
				`cpu: "1"`
				`memory: "1073741824"`
				`securityContext:`
				`capabilities:`
				`add:`
				`- all`
				`restartPolicy: Always`
				详细参见config-deployment.yaml配置
					kompose 自动迁移

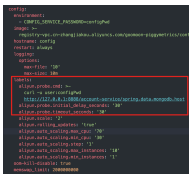
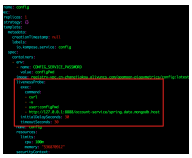
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
cpu_shares	指定容器对cpu资源限制	resource :request/ limits标签		 详细参见config-deployment.yaml配置	kompose 自动迁移
kernel_memory	同 docker run 参数	Kubernetes 不支持，迁移忽略	略	略	无法迁移
memswap_reservation	同 docker run 参数	Kubernetes 不支持，迁移忽略	略	略	无法迁移
memswap_limit	同 docker run 参数	Kubernetes 不支持，迁移忽略	略	略	无法迁移
shm_size	同 docker run 参数	Kubernetes 不支持，迁移忽略	略	略	无法迁移
oom-kill-disable	设置是否禁止 OOM Killer, 和 `docker run` 命令中的`--oom-kill-disable` 参数语义一致。	Kubernetes 不支持，迁移忽略	略	略	无法迁移
发布类标签列表					
restart	容器重启策略	restartPolicy 标签	略	略	kompose 自动迁移

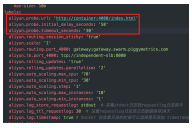
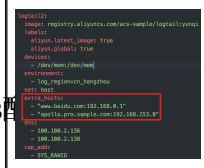
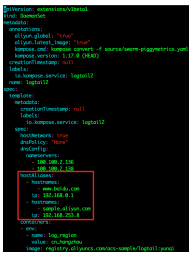
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.latest_image	是否拉取最新镜像	imagePullPolicy标签	aliyun.latest_image: true	 详细参见config-deployment.yaml配置	手动迁移：kompose 转换后，修改该应用对应资源文件`*-deployment.yaml`，添加imagePullPolicy配置。如果不填写，默认策略是always
depends_on	设置服务的依赖关系。	利用init container，liveness/readiness探针等技术实现服务健康检查，依赖检查等功能；详细参见解决服务依赖	略	略	依赖业务实现
aliyun.depends	设置服务的依赖关系。	利用init container，liveness/readiness探针等技术实现服务健康检查，依赖检查等功能；详细参见解决服务依赖	略	略	依赖业务实现

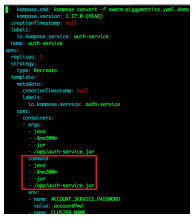
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
environment: affinity: service!=db	设置服务的部署约束条件。参见服务部署约束（affinity:service）	在Kubernetes中可以通过affinity.nodeAffinity affinity.podAffinity affinity.podAntiAffinity标签做Node 或Pod 亲和性配置，参见Affinity and anti-affinity。	略	略	依赖业务实现
environment: constraint: group==1	表示在所有带有`group:1`标签的节点上部署	在Kubernetes可以通过nodeSelector做Node过滤，参见nodeSelector。		 详细参见auth-mongodb-deployment.yaml配置	手动迁移：kompose 转换后，修改该应用对应资源文件`*-deployment.yaml`，添加nodeSelector配置。

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.global	设置该服务为全局服务。	通过DaemonSet类型应用支持，创建方式，参见DaemonSet		 详细参见logtail2-daemonset.yaml配置	kompose 自动迁移+手动修复 手动修复： kompose转换后，需要删除资源文件中的 status 章节（用来记录中间状态），才能部署成功。
aliyun.scale	设置该服务的容器数量，横向扩展服务。	replicas标签	aliyun.scale: '10'	replicas: 10	kompose 自动迁移
aliyun.rolling_updates	是否开启服务滚动更新。	strategy.type.Rolling标签，参见Strategy。			手动迁移： kompose转换后，需要手动配置滚动更新策略。
aliyun.rolling_updates.parallelism	每次并行更新的容器数量	maxUnavailable标签，参见Strategy。		参见config-deployment.yaml配置	手动迁移： kompose转换后，需要手动配置滚动更新策略。

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.auto_scaling.*	容器自动伸缩，根据服务的容器资源占用情况自动调整容器数量。参见容器自动伸缩。	无法简单做标签映射，阿里云容器服务支持在控制台界面上快速创建支持HPA的应用，实现容器资源的弹性伸缩。参见使用HPA弹性伸缩容器。			手动迁移： 部署Kubernetes资源文件后，在控制台配置。

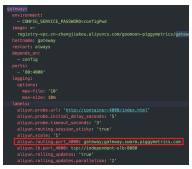
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.probe.cmd	健康检查执行的检查 Shell 命令，参见probe。	livenessProbe 标签			kompose 自动迁移
aliyun.probe.initial_delay_seconds	在容器启动后延迟几秒开始健康检查，参见probe。	livenessProbe 标签 说明： · 在 Swarm 里面健康检查只是用来表示容器状态，当并不会影响其对外提供服务。 · 但在 Kubernetes 中，如果 livenessProbe检查失败，则 Kubernetes 会强制重启容器，所以延迟该标签的值一定要设置合理，否则会导致 Kubernetes 不断重启容器，导致服务不可用		详细参见config-deployment.yaml配置	kompose自动迁移

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.probe.timeout_seconds	健康检查的超时时间，参见probe。	livenessProbe 标签			kompose 自动迁移
aliyun.probe.url	HTTP、TCP 请求的 URL，参见probe	livenessProbe 标签		 详细参见gateway-deployment.yaml 配置	kompose 自动迁移
extra_hosts	添加容器 Host 文件绑定配置	对应Kubernetes的hostAliases配置，参见Adding Additional Entries with HostAliases		 详细参见logtail2-daemonset.yaml 配置	手动迁移：kompose 转换后，修改该应用对应资源文件`*-deployment.yaml`，添加 hostAliases 配置。

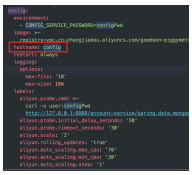
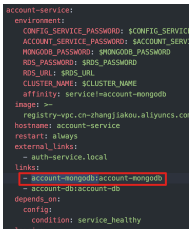
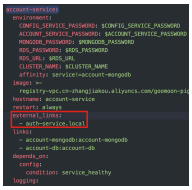
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
entrypoint	覆盖默认入口点	command 标签		 详细参见 auth-service-deployment.yaml配置	kompose 自动迁移
command	重写默认命令	args 标签		 详细参见 auth-service-deployment.yaml配置	kompose 自动迁移
网络配置类标签					

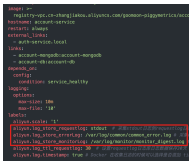
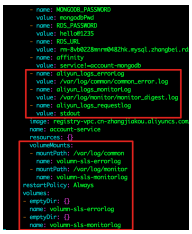
Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移

net	V1版compose文件格式，设置网络模式，同docker 命令行参数--net；参见net。	部分值，可以直接转换；部分值在Kubernetes不支持定制。其中：net:"host"可直接转换为hostNetwork: true；其余值在Kubernetes里面不支持。参见Host namespaces。	```
			logtail2:
			  image: registry.aliyuncs.com/acs-sample/logtail:
			  labels:
			    aliyun.latest_image: true
			    aliyun.global: true
			  devices:
			    - /dev/mem:/dev/mem
			  environment:
			    - log_region=cn_hangzhou
			  net: host
			  extra_hosts:
			    - "www.baidu.com:192.168.0.1"
			    - "apollo.pro.sample.com:192.168.253.8"
			  dns:
			    - 100.100.2.136
			    - 100.100.2.138
			  cap_add:
			    - SYS_RAWIO
			``
			net: host
				``
				apiVersion: extensions/v1beta1
				kind: DaemonSet
				metadata:
				  annotations:
				    aliyun.global: "true"
				    aliyun.latest_image: "true"
				    kompose.cmd: kompose convert -f sources/swarm-piggymetrics.y
				    kompose.version: 1.17.0 (HEAD)
				  creationTimestamp: null
				  labels:
				    io.kompose.service: logtail2
				  name: logtail2
				spec:
				  template:
				    metadata:
				      creationTimestamp: null
				      labels:
				        io.kompose.service: logtail2
				    spec:
				      hostNetwork: true
				      containers:
				      - env:
				        - name: log_region
				          value: cn_hangzhou
				        image: registry.aliyuncs.com/acs-sample/logtail:yunqi
				        name: logtail2
				        resources: {}
				        securityContext:
				          capabilities:
				            add:
				            - SYS_RAWIO
				      restartPolicy: Always
				``
				详细参见logtail2-daemonset.yaml配置
					手动迁移：
					kompose 转换后，手动修改Kubernetes 资源文件，添加hostNetwork配置。

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.routing.port_	· Swarm 简单路由，用于设置该服务的访问域名并支持多域名设置，参见routing。 · Swarm 简单路由，当设置了.local后缀域名时，仅用于集群内集群内服务间路由和负载均衡，参见集群内服务间路由和负载均衡。	对应Kubernetes ingress，但kompose 工具支持部分转换，但转换过程只支持转换第一个域名，后续域名规则需人工添加，且针对测试域名需要人工替换。参见Ingress 支持。		 详细参见gateway-ingress.yaml配置	kompose 自动迁移+手动修复 手动修复： · 普通域名：kompose转换后，需要手动修改Kubernetes资源文件，找到Kubernetes对应ingress资源文件`*-ingress.yaml`。 - 将其中的host：your-cluster-id.alicontainer.com修改成对应Kubernetes 集群的测试域名，其中，k8s-piggymetrics-

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.routing.session_sticky	设置 routing 在做请求路由的时候，是否保持 session sticky，即会话保持。 说明：此标签必须和 `routing` 配合使用，单独使用无效。	Kubernetes 不支持，迁移忽略	略	略	无法迁移

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
hostname	可用作跨Node容器间访问（基于DNS+LB），常用于跨服务访问。	转换成service的方式，实现跨容器访问；参见创建服务。		通过控制台创建一个同名(标注 1)的service 用于服务访问，服务端口为容器端口 	手动迁移 1. 先部署Kubernetes 资源文件。 2. 然后从控制台界面创建Service (cluster IP)。由于在Swarm 编排文件里面不知道容器对应端口，因此无法通过kompose 工具自动转换。
links	1. 通过服务名或别名访问跨主机容器组。 2. 跟depends on类似，决定启动顺序。	1. 基于服务名和别名，分别创建service的方式，实现跨容器访问；参见创建服务。 2. 启动顺序依赖，需要根据业务特点处理。		通过控制台分别创建一个同名的 service 用于服务访问。 	
external_links	将两个服务链接起来，使之可以通信；其相对 links 标签来说，支持链接到其它编排文件或其它方式创建的容器。	Kubernetes 只支持通过service 方式来实现跨容器访问；且service 对应的后端容器必须由Kubernetes 管理。 迁移方案： 1. 被external_links链接的容器通过		通过控制台创建一个同名的service 用于服务访问 	说明：由于Kubernetes Service 名只支持小写和中划线-，如果hostname、links、exter带有其它字符如.local这类带.字符的，迁移Kubernetes时，只能修改应用代码

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
dns	指定容器上游DNS服务器	对应Kubernetes的Pod级别的DNS 配置，详细参见Pod’s DNS Config。 说明：如果指定了DNS，则对应的Service 名称的解析可能会失效。		 详细参见logtail2-daemonset.yaml配置	手动迁移： kompose 转换后，修改该应用对应资源文件`*-deployment.yaml`，添加dnsConfig 配置。
日志配置类标签					
aliyun.log_store_<logstore_name>	将容器日志保存到阿里云日志服务	aliyun_logs环境变量，参见使用日志服务进行Kubernetes日志采集。		 详细参见account-service-deployment.yaml配置	kompose 自动迁移

Swarm标签名	Swarm标签含义	对应Kubernetes配置方案	Swarm 配置示例	Kubernetes 配置示例	如何迁移
aliyun.log_ttl_<logstore_name>	阿里云日志服务定制标签，用来设置日志库初始日志保存时间，单位为天，有效值为1~365 天，不配置则默认初始化为 2 天。参见集成日志服务。 这里设置的是初始配置值，如果后期您需要修改日志保存时间，需要到日志服务控制台进行设置。	没有对应标签支持，需要自己到日志服务控制台进行设置。	aliyun.log_ttl_requestlog: '30'		手动迁移过程： 1. 先部署Kubernetes 资源文件，并调试应用通过（这时会在日志服务生成对应的Logstore）。 2. 登录日志服务控制台修改Logstore配置，将数据保存时间调长。
aliyun.log.timestamp	Docker 在收集日志的时候选择是否添加timestamp，参见集成日志服务。	Kubernetes 不支持，迁移忽略	略	略	无法迁移 一般输出都带这个默认的time

10.5.11 附录：标签配置样例

account-db-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: account-db
```

```
  namespace: default
spec:
  type: ExternalName
  externalName: rm-8vb0228mnrm0482hk.mysql.zhangbei.rds.aliyuncs.com
```

account-service-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.log_store_errorLog: /var/log/common/common_error.log
    aliyun.log_store_monitorLog: /var/log/monitor/monitor_digest.log
    aliyun.log_store_requestlog: stdout
    aliyun.scale: "1"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --
volumes PersistentVolumeClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: account-service
  name: account-service
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: account-service
    spec:
      containers:
      - env:
        - name: ACCOUNT_SERVICE_PASSWORD
          value: accountPwd
        - name: CLUSTER_NAME
          value: K8SCluster
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        - name: MONGODB_PASSWORD
          value: mongodbPwd
        - name: RDS_PASSWORD
          value: hello@1235
        - name: RDS_URL
          value: rm-8vb0228mnrm0482hk.mysql.zhangbei.rds.aliyuncs.com
        - name: affinity
          value: service!=account-mongodb
        - name: aliyun_logs_errorlog
          value: /var/log/common/common_error.log
        - name: aliyun_logs_monitorlog
          value: /var/log/monitor/monitor_digest.log
        - name: aliyun_logs_requestlog
          value: stdout
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-
piggymetrics/account-service
        name: account-service
        resources: {}
        volumeMounts:
        - mountPath: /var/log/common
          name: volumn-sls-errorlog
        - mountPath: /var/log/monitor
          name: volumn-sls-monitorlog
      restartPolicy: Always
```

```
      volumes:
      - emptyDir: {}
        name: volumn-sls-errorlog
      - emptyDir: {}
        name: volumn-sls-monitorlog
status: {}
```

auth-mongodb-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.probe.url: tcp://container:27017
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --
volumes PersistentVolumeClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: auth-mongodb
  name: auth-mongodb
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: auth-mongodb
    spec:
      nodeSelector:
        group: db
      containers:
      - command:
        - /init.sh
        env:
        - name: MONGODB_PASSWORD
          value: mongodbPwd
        - name: constraint
          value: group==db
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-
piggymetrics/mongodb
        livenessProbe:
          tcpSocket:
            port: 27017
        name: auth-mongodb
        resources: {}
      restartPolicy: Always
status: {}
```

auth-service-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.latest_image: "true"
    aliyun.log_store_requestlog: stdout
    aliyun.routing.port_5000: auth-service.local
    aliyun.scale: "1"
```

```
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --
volumes PersistentVolumeClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: auth-service
  name: auth-service
spec:
  replicas: 1
  strategy:
    type: Recreate
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: auth-service
    spec:
      containers:
      - args:
        - java
        - -Xmx500m
        - -jar
        - /app/auth-service.jar
        command:
        - java
        - -Xmx200m
        - -jar
        - /app/auth-service.jar
        env:
        - name: ACCOUNT_SERVICE_PASSWORD
          value: accountPwd
        - name: CLUSTER_NAME
          value: K8SCluster
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        - name: MONGODB_PASSWORD
          value: mongodbPwd
        - name: NOTIFICATION_SERVICE_PASSWORD
          value: notificationPwd
        - name: STATISTICS_SERVICE_PASSWORD
          value: statisticsPwd
        - name: aliyun_logs_requestlog
          value: stdout
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-
piggymetrics/auth-service:latest
        name: auth-service
        ports:
        - containerPort: 5000
        resources: {}
        securityContext:
          capabilities:
            add:
            - all
            drop:
            - SETGID
            - SETUID
          privileged: true
        volumeMounts:
        - mountPath: /data/oss
          name: volume-name-piggymetrics
        - mountPath: /data/nas
          name: vn-nas-piggymetrics
        - mountPath: /data/yunpan
          name: vn-yunpan-piggymetrics
```

```
        - mountPath: /var/run/docker.sock
          name: auth-service-claim3
        - mountPath: /data/tmp
          name: auth-service-claim4
      restartPolicy: Always
      volumes:
      - name: volume-name-piggymetrics
        persistentVolumeClaim:
          claimName: volume-name-piggymetrics
      - name: vn-nas-piggymetrics
        persistentVolumeClaim:
          claimName: vn-nas-piggymetrics
      - name: vn-yunpan-piggymetrics
        persistentVolumeClaim:
          claimName: vn-yunpan-piggymetrics
      - hostPath:
          path: /var/run/docker.sock
        name: auth-service-claim3
      - hostPath:
          path: /tmp
        name: auth-service-claim4
status: {}
```

config-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.auto_scaling.max_cpu: "70"
    aliyun.auto_scaling.max_instances: "10"
    aliyun.auto_scaling.min_cpu: "30"
    aliyun.auto_scaling.min_instances: "1"
    aliyun.auto_scaling.step: "1"
    aliyun.latest_image: "false"
    aliyun.probe.cmd: curl -u user:configPwd http://127.0.0.1:8888/
account-service/spring.data.mongodb.host
    aliyun.probe.initial_delay_seconds: "30"
    aliyun.probe.timeout_seconds: "30"
    aliyun.rolling_updates: "true"
    aliyun.rolling_updates.parallelism: "2"
    aliyun.scale: "3"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --
volumes PersistentVolumeClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: config
  name: config
spec:
  replicas: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 13
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: config
    spec:
      containers:
```

```
      - env:
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-
piggymetrics/config:latest
        imagePullPolicy: Always
        livenessProbe:
          exec:
            command:
            - curl
            - -u
            - user:configPwd
            - http://127.0.0.1:8888/account-service/spring.data.
mongodb.host
          initialDelaySeconds: 30
          timeoutSeconds: 30
        name: config
        resources:
          limits:
            cpu: "1"
            memory: "1073741824"
        securityContext:
          capabilities:
            add:
            - all
      restartPolicy: Always
status: {}
```

gateway-deployment.yaml

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  annotations:
    aliyun.auto_scaling.max_cpu: "70"
    aliyun.auto_scaling.max_instances: "10"
    aliyun.auto_scaling.min_cpu: "30"
    aliyun.auto_scaling.min_instances: "1"
    aliyun.auto_scaling.step: "1"
    aliyun.log_store_requestlog: stdout
    aliyun.probe.initial_delay_seconds: "50"
    aliyun.probe.timeout_seconds: "30"
    aliyun.probe.url: http://container:4000/index.html
    aliyun.rolling_updates: "true"
    aliyun.rolling_updates.parallelism: "2"
    aliyun.routing.port_4000: gateway;gateway.swarm.piggymetrics.com
    aliyun.scale: "1"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --
volumes PersistentVolumeClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: gateway
  name: gateway
spec:
  replicas: 1
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: gateway
    spec:
```

```
      containers:
      - env:
        - name: CONFIG_SERVICE_PASSWORD
          value: configPwd
        - name: aliyun_logs_requestlog
          value: stdout
        image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-
piggymetrics/gateway:latest
        livenessProbe:
          httpGet:
            path: /index.html
            port: 4000
          initialDelaySeconds: 50
          timeoutSeconds: 30
        name: gateway
        ports:
        - containerPort: 4000
      restartPolicy: Always
status: {}
```

gateway-ingress.yaml

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  creationTimestamp: null
  labels:
    io.kompose.service: gateway
  name: gateway
spec:
  rules:
  - host: gateway.c7f537c92438f415b943e7c2f8ca30b3b.cn-zhangjiakou.
alicontainer.com
    http:
      paths:
      - backend:
          serviceName: gateway
          servicePort: 80
  - host: gateway.swarm.piggymetrics.com
    http:
      paths:
      - backend:
          serviceName: gateway
          servicePort: 80
status:
  loadBalancer: {}
```

logtail2-daemonset.yaml

```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  annotations:
    aliyun.global: "true"
    aliyun.latest_image: "false"
    kompose.cmd: kompose convert -f source/swarm-piggymetrics.yaml --
volumes PersistentVolumeClaimOrHostPath
    kompose.version: 1.17.0 (HEAD)
  creationTimestamp: null
  labels:
    io.kompose.service: logtail2
  name: logtail2
```

```
spec:
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.service: logtail2
    spec:
      hostNetwork: true
      dnsPolicy: "None"
      dnsConfig:
        nameservers:
        - 100.100.2.136
        - 100.100.2.138
      hostAliases:
      - ip: "192.168.0.1"
        hostnames:
        - "www.baidu.com"
      - ip: "192.168.253.8"
        hostnames:
        - "sample.aliyun.com"
      containers:
      - env:
        - name: log_region
          value: cn_hangzhou
        image: registry.aliyuncs.com/acs-sample/logtail:yunqi
        name: logtail2
        resources: {}
        securityContext:
          capabilities:
            add:
            - SYS_RAWIO
      restartPolicy: Always
```

swarm-piggymetrics.yaml

```
version: '2'
services:
  rabbitmq:
    image: >-
      registry-vpc.cn-hangzhou.aliyuncs.com/hd_docker_images/rabbitmq:
latest
    hostname: rabbitmq
    restart: always
    ports:
      - '15672:15672'
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.scale: '1'
      aliyun.routing.port_15672: rabbitmq
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: rabbitmq

  config:
    environment:
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
```

```
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
config:latest
    hostname: config
    restart: always
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.probe.cmd: >-
        curl -u user:configPwd
        http://127.0.0.1:8888/account-service/spring.data.mongodb.host
      aliyun.probe.initial_delay_seconds: '30'
      aliyun.probe.timeout_seconds: '30'
      aliyun.scale: '10'
      aliyun.rolling_updates: 'true'
      aliyun.rolling_updates.parallelism: '2'
      aliyun.auto_scaling.max_cpu: '70'
      aliyun.auto_scaling.min_cpu: '30'
      aliyun.auto_scaling.step: '1'
      aliyun.auto_scaling.max_instances: '10'
      aliyun.auto_scaling.min_instances: '1'
      aliyun.latest_image: 'false'
    oom-kill-disable: true
    memswap_limit: 2000000000
    shm_size: 67108864
    memswap_reservation: 536870912
    kernel_memory: 0
    name: config
    mem_limit: 1073741824
    cpu_shares: 100
    cap_add:
      - all

  registry:
    environment:
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
registry:latest
    hostname: registry
    restart: always
    depends_on:
      - config
    ports:
      - '8761:8761'
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.scale: '1'
      aliyun.routing.port_8761: eureka
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: registry

  gateway:
    environment:
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
```

```
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
gateway:latest
    hostname: gateway
    restart: always
    depends_on:
      - config
    ports:
      - '80:4000'
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.probe.url: 'http://container:4000/index.html'
      aliyun.probe.initial_delay_seconds: '50'
      aliyun.probe.timeout_seconds: '30'
      aliyun.routing.session_sticky: 'true'
      aliyun.scale: '1'
      aliyun.routing.port_4000: gateway;gateway.swarm.piggymetrics.com
      aliyun.lb.port_4000: tcp://independent-slb:8080
      aliyun.rolling_updates: 'true'
      aliyun.rolling_updates.parallelism: '2'
      aliyun.auto_scaling.max_cpu: '70'
      aliyun.auto_scaling.min_cpu: '30'
      aliyun.auto_scaling.step: '1'
      aliyun.auto_scaling.max_instances: '10'
      aliyun.auto_scaling.min_instances: '1'
      aliyun.log_store_requestlog: stdout  # 采集stdout日志到requestlog
日志库中
      aliyun.log_ttl_requestlog: 30  # 设置requestlog日志库日志数据保存30
天
      aliyun.log.timestamp: true # Docker 在收集日志的时候可以选择是否添加
timestamp
    external_links:
      - auth-service.local
    oom-kill-disable: true
    name: gateway

  auth-service:
    environment:
      - MONGODB_PASSWORD=mongodbPwd
      - NOTIFICATION_SERVICE_PASSWORD=notificationPwd
      - STATISTICS_SERVICE_PASSWORD=statisticsPwd
      - ACCOUNT_SERVICE_PASSWORD=accountPwd
      - CONFIG_SERVICE_PASSWORD=configPwd
      - CLUSTER_NAME=$CLUSTER_NAME
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
auth-service:latest
    command: ["java", "-Xmx500m", "-jar", "/app/auth-service.jar"]
    restart: always
    logging:
      options:
        max-file: '10'
        max-size: 10m
    labels:
      aliyun.scale: '1'
      aliyun.log_store_requestlog: stdout  # 采集stdout日志到requestlog
日志库中
      aliyun.log_ttl_requestlog: 30  # 设置requestlog日志库日志数据保存30
天
      aliyun.log.timestamp: false # Docker 在收集日志的时候可以选择是否添
加 timestamp
      aliyun.routing.session_sticky: "true"
```

```
      aliyun.routing.port_5000: auth-service.local
      aliyun.latest_image: 'true'
      aliyun.depends: config
    links:
      - auth-mongodb
    volumes:
      - 'volume_name_piggymetrics:/data/oss:rw'
      - 'VN_NAS_PIGGYMETRICS:/data/nas:rw'
      - 'VN_YUNPAN_PIGGYMETRICS:/data/yunpan:rw'
      - '/var/run/docker.sock:/var/run/docker.sock:rw'
      - '/tmp:/data/tmp'
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: auth-service
    hostname: auth-service
    entrypoint:
      - 'java'
      - '-Xmx200m'
      - '-jar'
      - '/app/auth-service.jar'
    cap_add:
      - all
    cap_drop:
      - SETGID
      - SETUID
    privileged: true

  auth-mongodb:
    environment:
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetri
cs/mongodb
    restart: always
    labels:
      aliyun.probe.url: tcp://container:27017
    logging:
      options:
        max-size: 10m
        max-file: '10'
    entrypoint: '/init.sh'
    devices:
      - /dev/mem:/dev/mem

  account-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
      ACCOUNT_SERVICE_PASSWORD: $ACCOUNT_SERVICE_PASSWORD
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      RDS_PASSWORD: $RDS_PASSWORD
      RDS_URL: $RDS_URL
      CLUSTER_NAME: $CLUSTER_NAME
      affinity: service!=account-mongodb
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
account-service
    hostname: account-service
    restart: always
    external_links:
      - auth-service.local
    links:
      - account-mongodb:account-mongodb
```

```
      - account-db:account-db
    depends_on:
      config:
        condition: service_healthy
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.scale: '1'
      aliyun.log_store_requestlog: stdout  # 采集stdout日志到requestlog
日志库中
      aliyun.log_store_errorLog: /var/log/common/common_error.log # 采
集error日志到requestlog日志库中
      aliyun.log_store_monitorLog: /var/log/monitor/monitor_digest.log
 # 采集monitor_digest日志到requestlog日志库中
      aliyun.log_ttl_requestlog: 30  # 设置requestlog日志库日志数据保存30
天
      aliyun.log.timestamp: true # Docker 在收集日志的时候可以选择是否添加
timestamp

  account-mongodb:
    environment:
      INIT_DUMP: account-service-dump.js
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetri
cs/mongodb
    restart: always
    logging:
      options:
        max-size: 10m
        max-file: '10'
  statistics-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      STATISTICS_SERVICE_PASSWORD: $STATISTICS_SERVICE_PASSWORD
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
statistics-service
    hostname: statistics-service
    external_links:
      - auth-service.local
    restart: always
    depends_on:
      config:
        condition: service_healthy
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.log_store_requestlog: stdout  # 采集stdout日志到requestlog
日志库中
      aliyun.log_ttl_requestlog: 30  # 设置requestlog日志库日志数据保存30
天
      aliyun.log.timestamp: true # Docker 在收集日志的时候可以选择是否添加
timestamp
  statistics-mongodb:
    environment:
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
```

```
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetri
cs/mongodb
    hostname: statistics-mongodb
    restart: always
    logging:
      options:
        max-size: 10m
        max-file: '10'
  notification-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      NOTIFICATION_SERVICE_PASSWORD: $NOTIFICATION_SERVICE_PASSWORD
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
notification-service
    hostname: notification-service
    external_links:
      - auth-service.local
    restart: always
    depends_on:
      config:
        condition: service_healthy
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.log_store_requestlog: stdout  # 采集stdout日志到requestlog
日志库中
      aliyun.log_ttl_requestlog: 30  # 设置requestlog日志库日志数据保存30
天
      aliyun.log.timestamp: true # Docker 在收集日志的时候可以选择是否添加
timestamp
  notification-mongodb:
    image: registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetri
cs/mongodb
    hostname: notification-mongodb
    restart: always
    environment:
      MONGODB_PASSWORD: $MONGODB_PASSWORD
      constraint: group==db
    logging:
      options:
        max-size: 10m
        max-file: '10'
  monitoring:
    environment:
      - CONFIG_SERVICE_PASSWORD=configPwd
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
monitoring:latest
    hostname: monitoring
    restart: always
    depends_on:
      - config
    ports:
      - '9000:9000'
    logging:
      options:
        max-size: 10m
        max-file: '10'
    labels:
      aliyun.scale: '1'
```

```
      aliyun.routing.port_9000: hystrix
    memswap_limit: 0
    shm_size: 0
    memswap_reservation: 0
    kernel_memory: 0
    name: monitoring
  turbine-stream-service:
    environment:
      CONFIG_SERVICE_PASSWORD: $CONFIG_SERVICE_PASSWORD
    image: >-
      registry-vpc.cn-zhangjiakou.aliyuncs.com/goomoon-piggymetrics/
turbine-stream-service
    hostname: turbine-stream-service
    restart: always
    depends_on:
      config:
        condition: service_healthy
    ports:
      - '8989:8989'
    logging:
      options:
        max-size: 10m
        max-file: '10'

  logtail2:
    image: registry.aliyuncs.com/acs-sample/logtail:yunqi
    labels:
      aliyun.latest_image: 'false'
      aliyun.global: 'true'
    devices:
      - /dev/mem:/dev/mem
    environment:
      - log_region=cn_hangzhou
    net: host
    extra_hosts:
      - "www.baidu.com:192.168.0.1"
      - "sample.aliyun.com:192.168.253.8"
    dns:
      - 100.100.2.136
      - 100.100.2.138
    cap_add:
      - SYS_RAWIO
```

10.6 应用回归测试

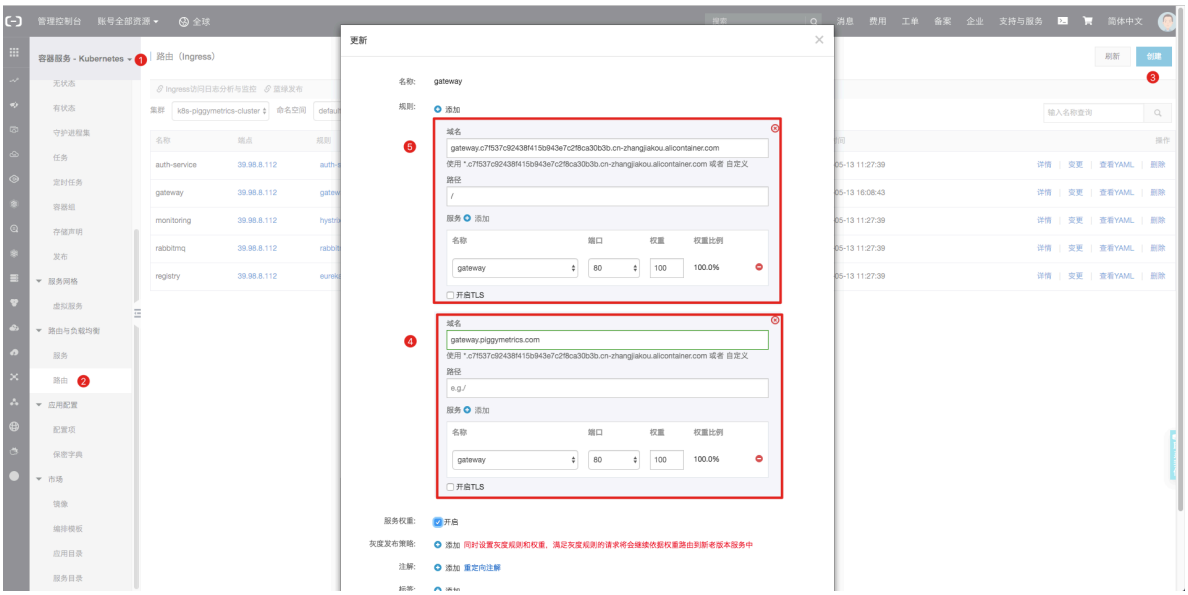
当应用启动成功后，您需要通过业务功能回归测试，确认应用没有问题。

配置测试域名

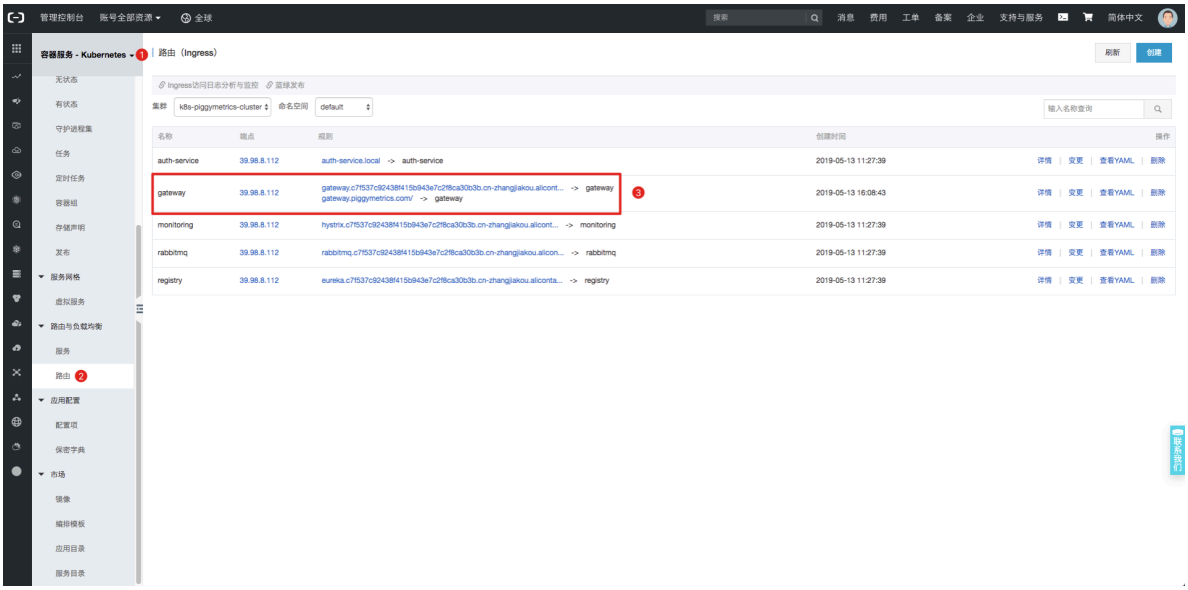
在Kubernetes 集群中，可以通过配置Service 实现集群内部的访问，通过配置路由实现从集群外部访问集群服务。

而在配置路由时，可以配置多域名，包括生产域名（客户自己备案的域名，完整域名）和测试域名（由容器平台提供域名，提供域名前缀即可）。你可以按照如下方法进行操作。

1. 登录容器服务管理控制台，在左侧导航栏选择路由与负载均衡 > 路由，单击右上角的创建，创建路由。



2. 创建完成后，您可以在路由列表中看到对应访问域名及端点（SLB公网IP）。



测试业务功能

客户可以通过上述配置的测试域名或客户自己的备案域名（备案域名因尚未更新DNS解析到该SLB公网IP，需在本地做域名绑定）就可以访问Kubernetes 集群提供的业务功能。示例域名如下：

图 10-5: 测试域名

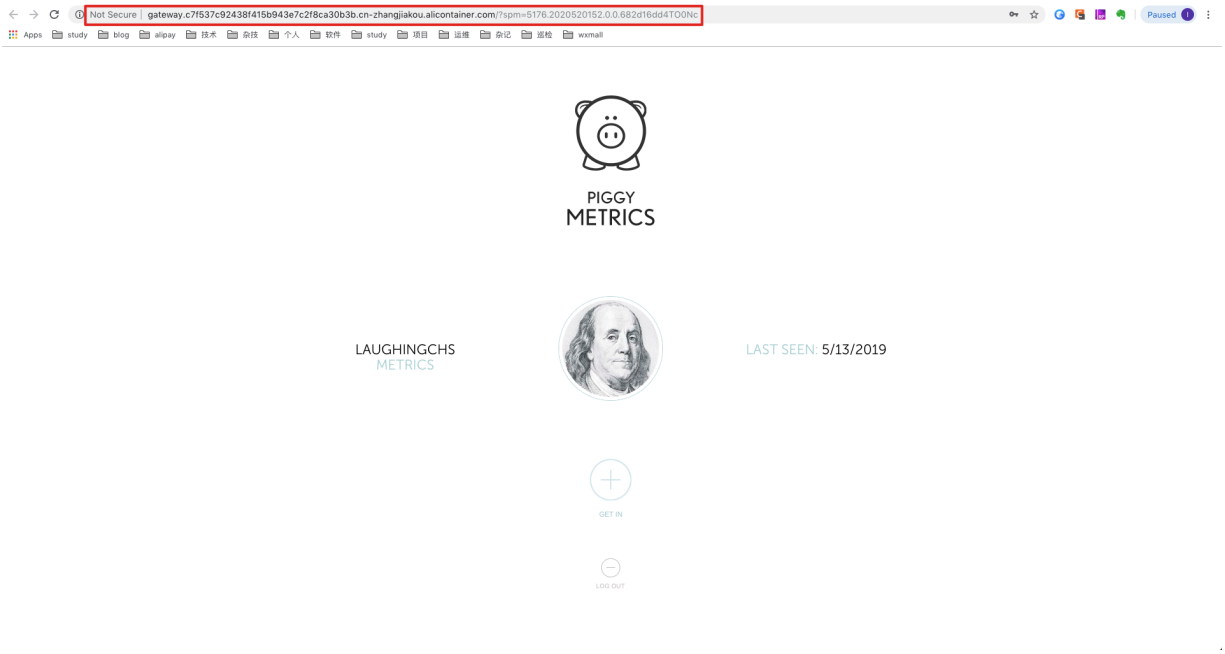
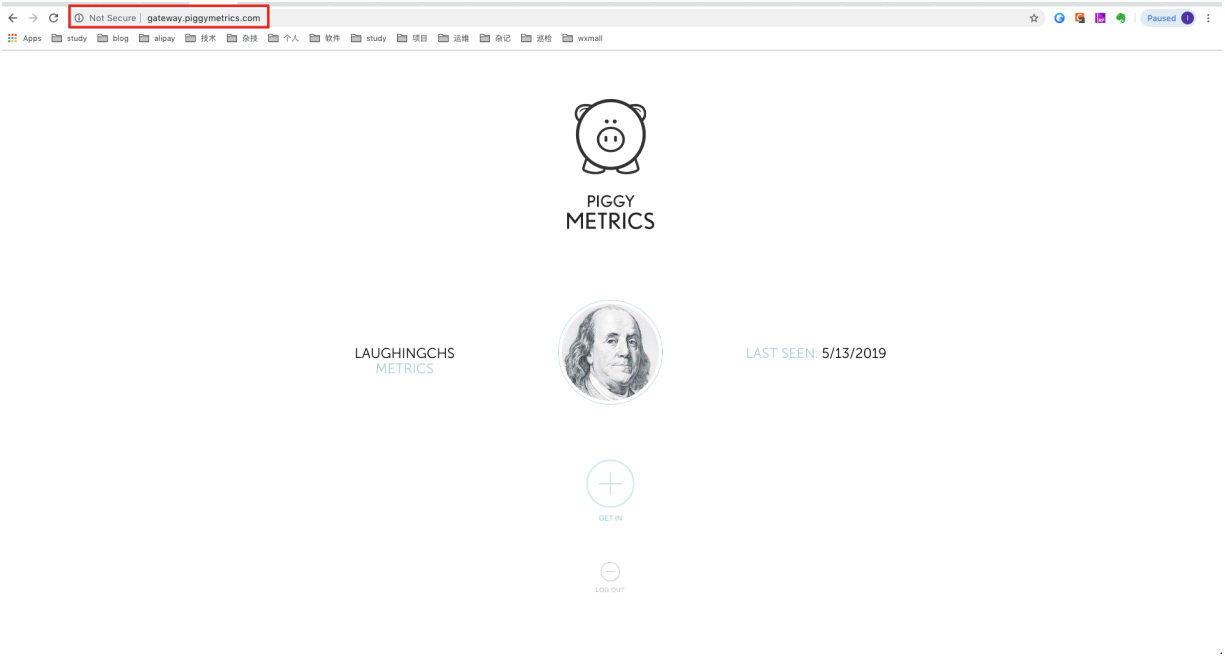


图 10-6: 用户备案域名



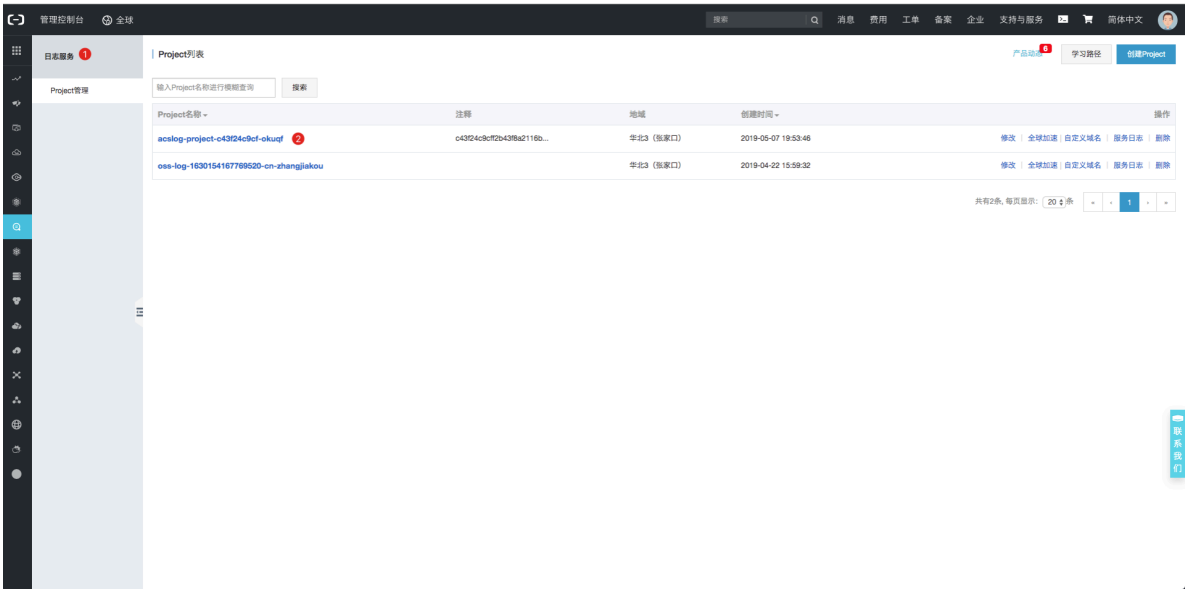
说明:

请根据测试域名或备案域名完成业务功能回归测试，以保证新集群提供的业务服务没有问题。

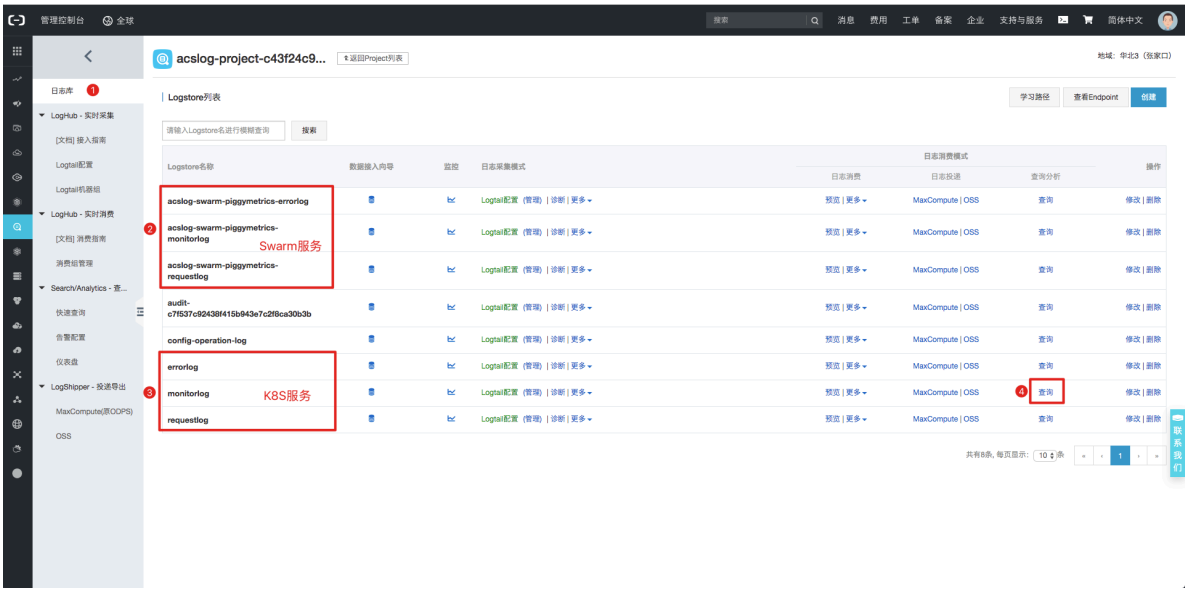
确认应用日志

在迁移应用日志配置中，Swarm和Kubernetes 的 Logstore 生成规则不同；所以，Swarm 集群生成的应用日志和 Kubernetes 生成的应用日志在日志服务同个Project，不同 Logstore下面。在测试过程中，我们可以查看 Kubernetes 创建的 Logstore 是否有业务日志输出，并确认输出是否正确。

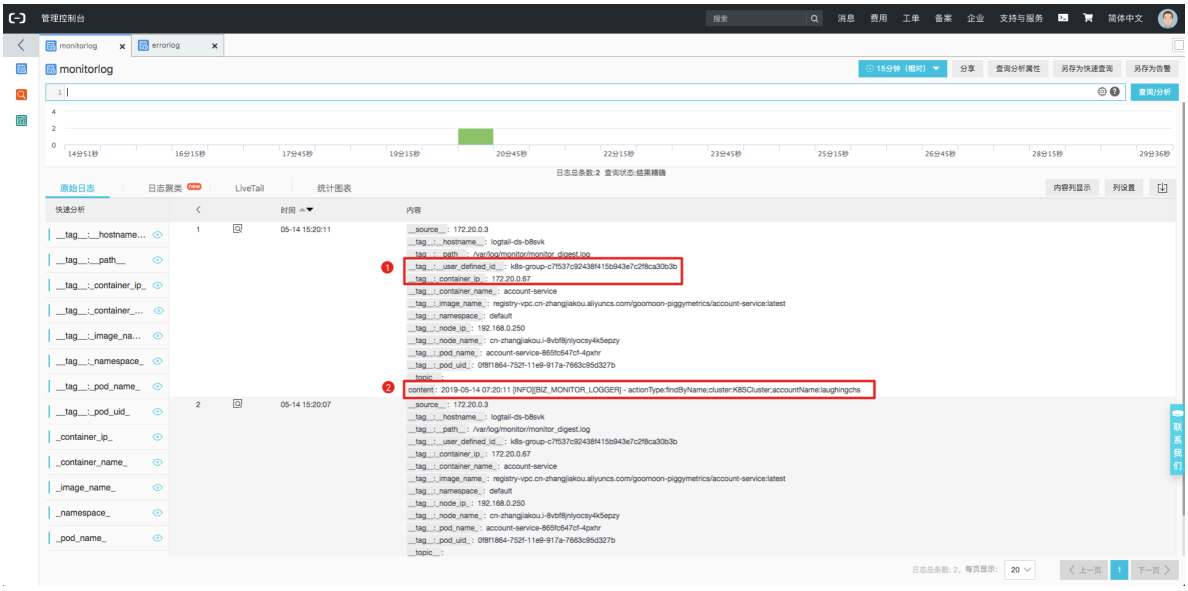
1. 登录日志服务控制台，单击Project名称。



2. 在日志库页签，在 Logstore 列表中，可以看到对应的 Logstore。



3. 单击查询，即可查看 Kubernetes 集群的日志信息，确认输出是否正常。



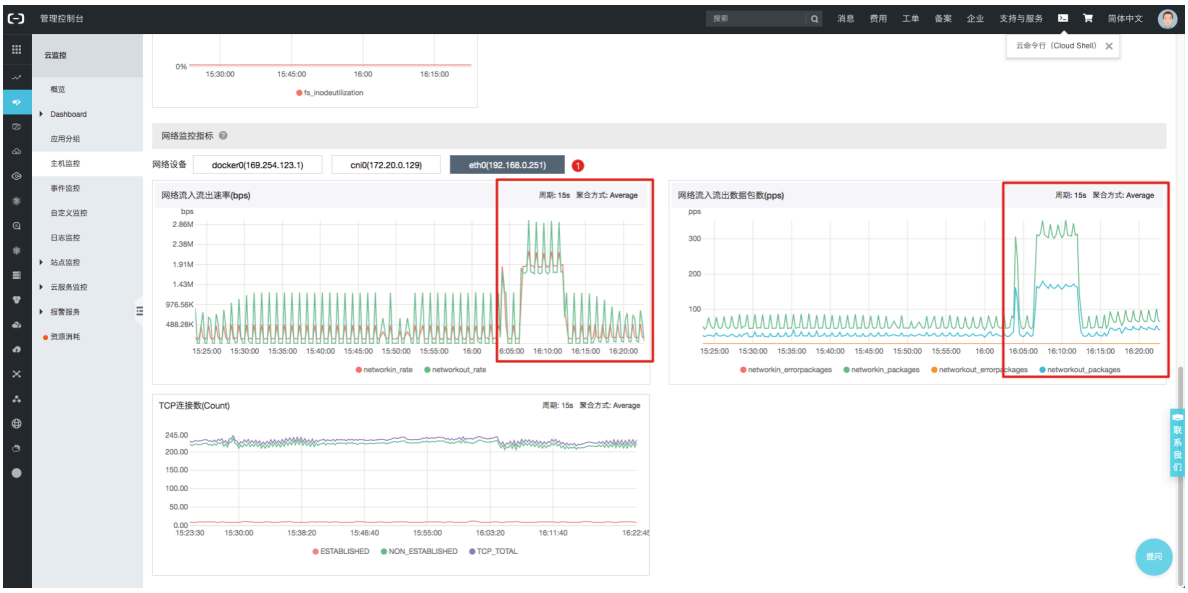
确认应用监控

在确认 Kubernetes 集群在新Logstore日志输出正常之后，需要进一步确认基于该日志Logstore基础之上的云监控、业务实时监控ARMS、消息投递存储等云产品是否正常配置，详情请参见日志服务-实时消费并按自身业务实际使用情况，确认各消费场景配置是否正常。

本例中，主要通过示例应用swarm-piggymetrics 云监控中的主机监控和日志监控确认Kubernetes 集群的服务运行情况，具体操作如下：

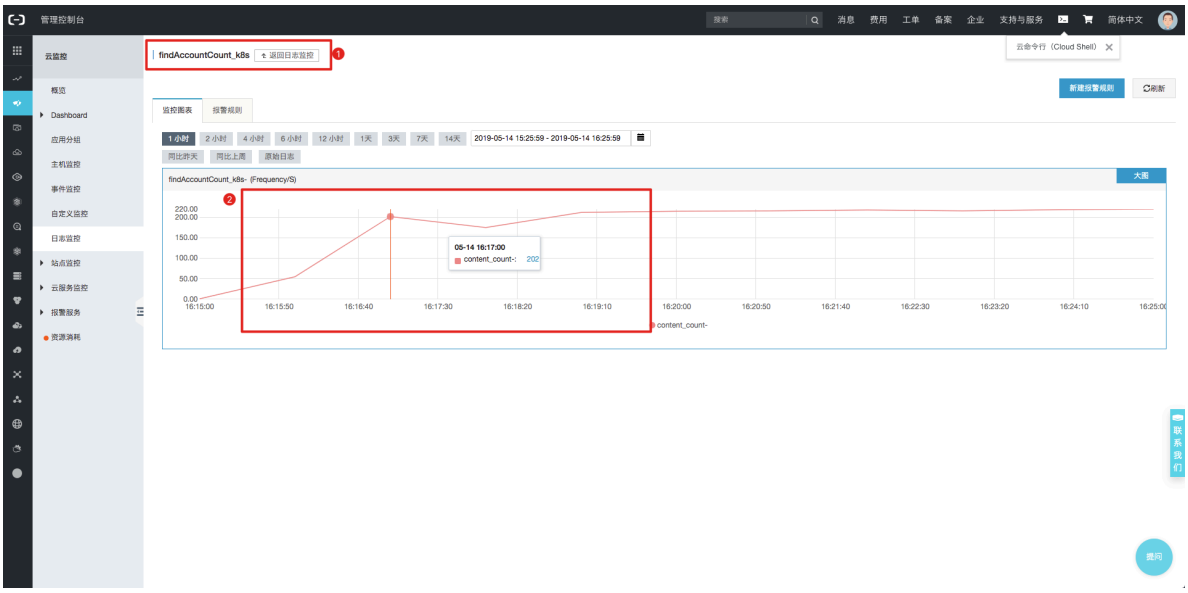
1. 查看主机监控。请参见云监控控制台如何查看指定日期的监控数据。

   在创建Kubernetes 集群时，我们选择安装云监控插件，可以监控Kubernetes 集群中各个Node节点机器的系统负载及网络监控信息，并从中观察我们测试流量是否有进入到新Kubernetes 集群。



2. 查看日志监控。请参见查看日志监控。

更详细业务监控，可以通过云日志监控或业务实时监控服务等云产品实现。这里以云监控中的日志监控为例，在原来Swarm集群的日志监控规则基础上，新创建一个日志监控findAccountCount_k8s用于监听K8S集群的机器日志，并配置对应的解析规则，然后可以通过监控图表可以查看到有新的测试流量进入。



说明:

通过上图，可以明显看到Kubernetes集群账户查询请求明显从无到有开始上涨，也验证了我们测试流量确实是由新集群k8s-piggymetrics-cluster在对外提供服务

10.7 SLB灰度引流

10.7.1 SLB灰度引流概述

本文主要为您介绍灰度部分生产流量到Kubernetes 集群，验证Kubernetes 集群的业务功能；避免回归测试未能覆盖所有业务功能。

前提条件

您已经完成业务功能回归测试，请参见应用回归测试。

NodePort引流

灰度引流主要有两种方式，基于Kubernetes 集群SLB引流 或 基于Swarm集群SLB引流；具体两种引流方式的情况如下：

- 基于Kubernetes SLB引流：更新客户端或DNS解析，将Kubernetes 集群SLB地址追加到客户端或DNS中，实现流量引入。
  - 引流成本高

    更新客户端或DNS解析配置
  - 回滚风险高
    - 万一Kubernetes 集群有问题，只能再次更新客户端或DNS配置。
    - 由于客户端发布时间较长或DNS运营商缓存问题，一般无法实时生效，导致线上业务有损。
- 基于Swarm SLB引流：将Kubernetes 集群服务挂到Swarm SLB下面并设置权重，实现流量引入。
  - 引流成本较低
    - 通过SLB 控制台，更新Swarm SLB配置即可。
    - 可以在SLB控制台权重配置，实现更精细的流量引入。
  - 回滚风险低
    - 通过SLB 控制台，更新Swarm SLB配置即可。
    - 实时生效。

操作步骤

1. 创建Kubernetes NodePort服务。
2. 验证Kubernetes NodePort服务。
3. 修改Swarm SLB配置。

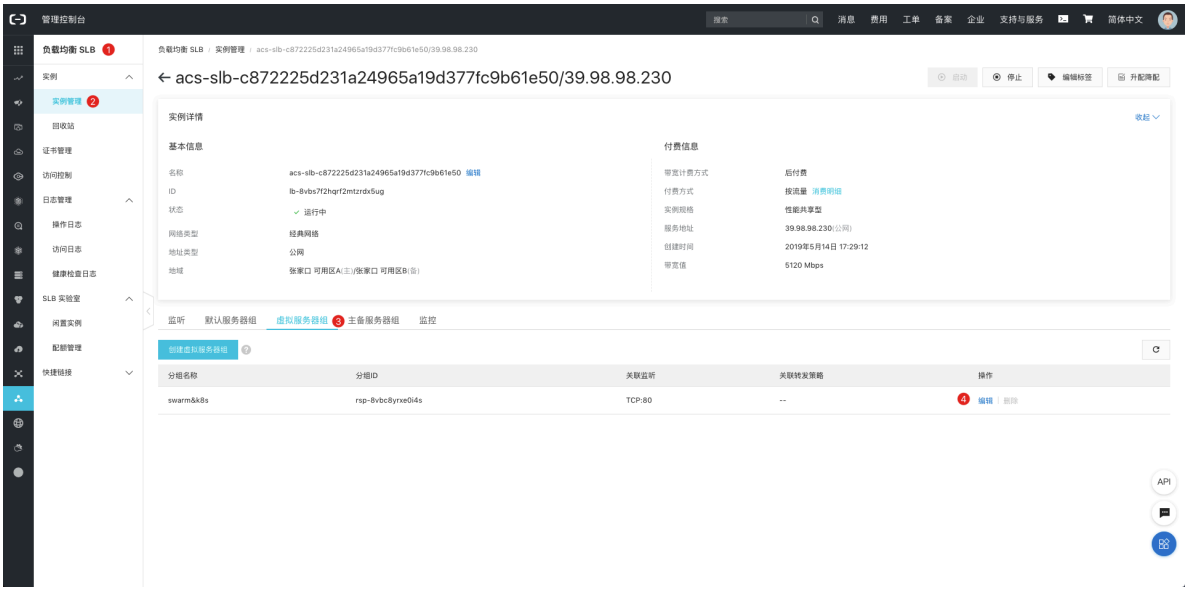
引流快速回滚

如果引入生产流量之后，发现Kubernetes 集群业务服务有问题；则调整虚拟服务器组，将Kubernetes 集群机器流量权重设置成0或将其剔除即可。

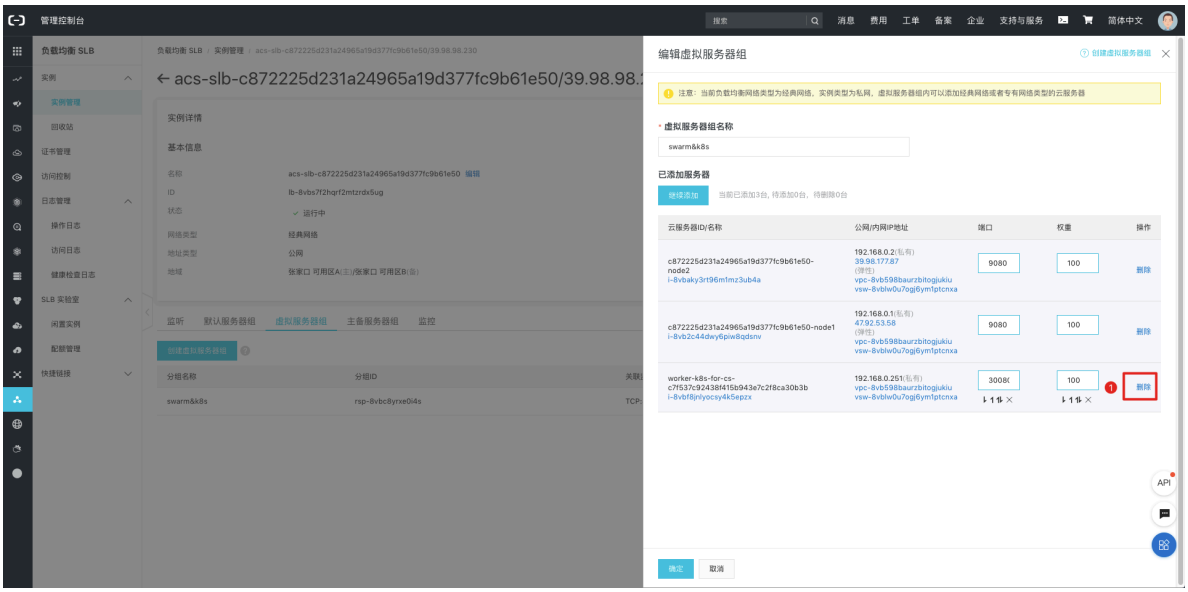
操作步骤

1. 登录负载均衡管理控制台，在实例管理页面单击目标实例。

2. 在实例详情页面，单击虚拟服务器组，在目标分组右侧单击编辑。



3. 从右侧滑出编辑虚拟服务器组页面，在目标虚拟服务右侧单击删除，并单击确定。



灰度引流总结

主要介绍如何通过Swarm集群SLB 虚拟服务器组机制，将生产流量部分引流到Kubernetes的NodePort Service，进一步验证Kubernetes 集群业务服务是否正常。

通过Swarm SLB + Kubernetes NodePort 引流方式，相较于传统客户端或DNS切流方式，成本更低（只需在SLB控制台操作），风险更可控（切流/回滚实时生效，流量控制更精细）。

10.7.2 创建Kubernetes NodePort服务

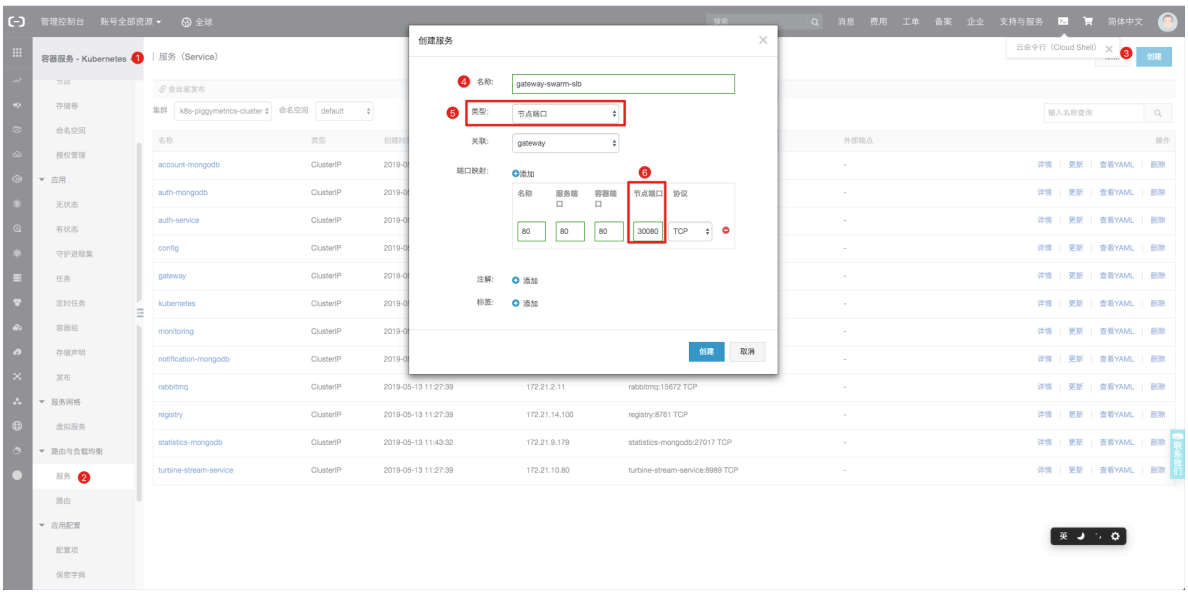
本文主要介绍在Kubernetes-piggymetrics-cluster集群中，创建一个NodePort类型的Service，用于后续从Swarm集群的SLB 上引入生产流量。

前提条件

您已经创建好一个Kubernetes托管版集群。请参见创建Kubernetes 托管版集群。

操作步骤

1. 登录容器服务管理控制台，在左侧导航栏选择路由与负载均衡 > 服务，在目标集群Kubernetes-piggymetrics-cluster右侧单击创建。
2. 在创建服务页面，填写对应的内容，并单击创建。请参考创建服务。



说明:

- 名称：输入服务的名称，本例中为 gateway-swarm-slb。
- 类型：选择服务类型，即服务访问方式，请选择节点端口类型。

  节点端口：即 NodePort，通过每个 Node 上的 IP 和静态端口（NodePort）暴露服务。NodePort 服务会路由到 ClusterIP 服务，这个 ClusterIP 服务会自动创建。通过请求 `<NodeIP>:<NodePort>`，可以从集群的外部访问一个 NodePort 服务。
- 关联部署：选择服务要绑定的后端对象，本例中是前面创建的 gateway服务。
- 端口映射：添加服务端口和容器端口，容器端口需要与后端的 pod 中暴露的容器端口一致。

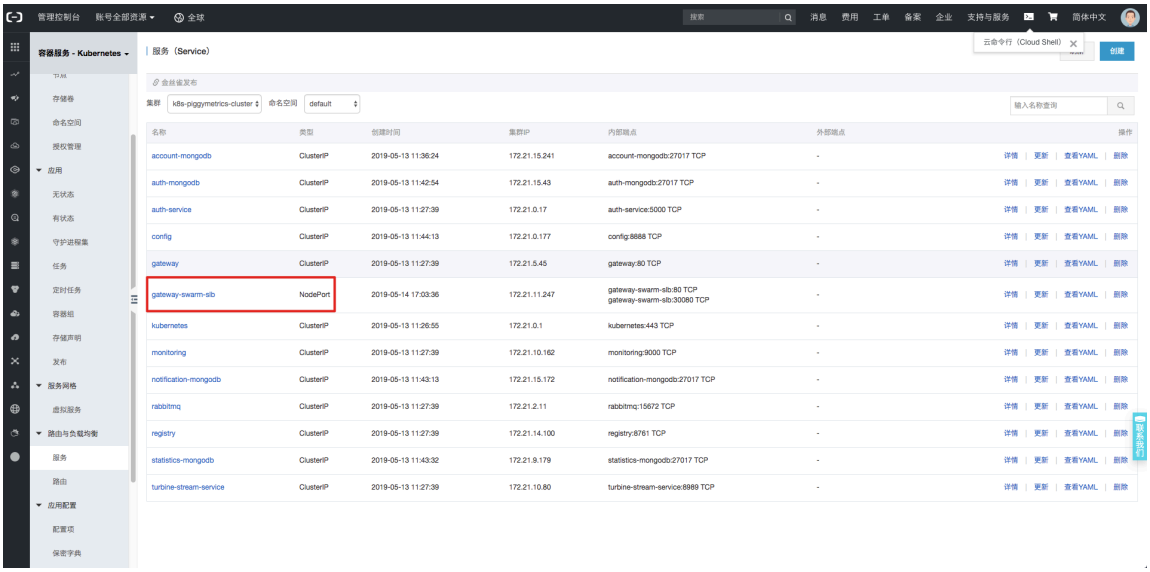
  节点端口范围为：30000~32767。

10.7.3 验证Kubernetes NodePort服务

操作步骤

1. 查看服务是否创建成功。

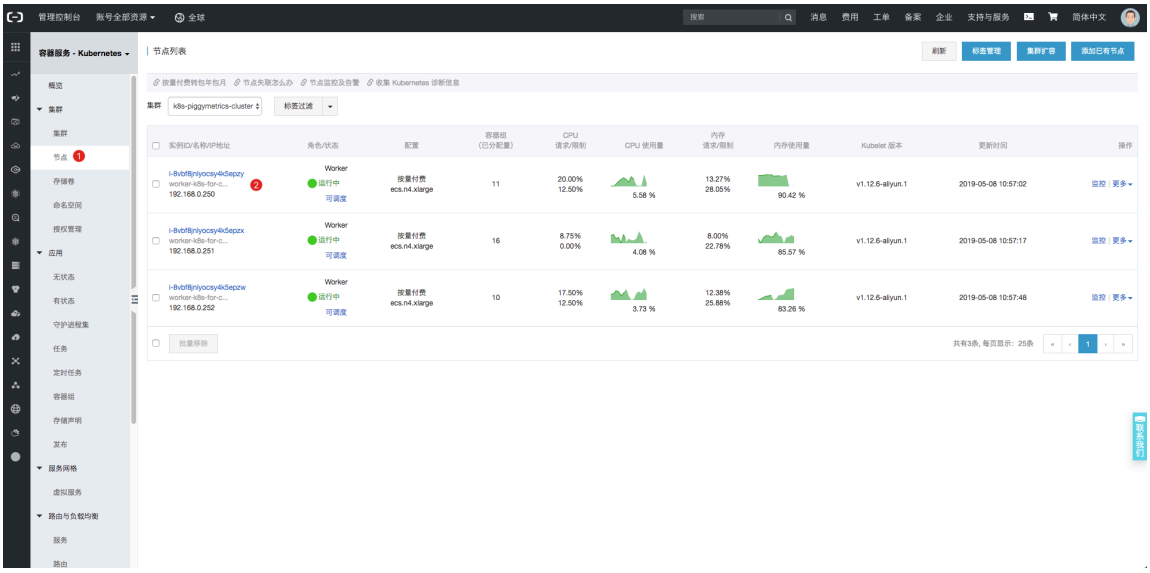
   a. 登录容器服务管理控制台，在左侧导航栏选择路由与负载均衡 > 服务，在服务列表中，查看`gateway-swarm-slb`为`NodePort`。



2. 验证服务是否正常。

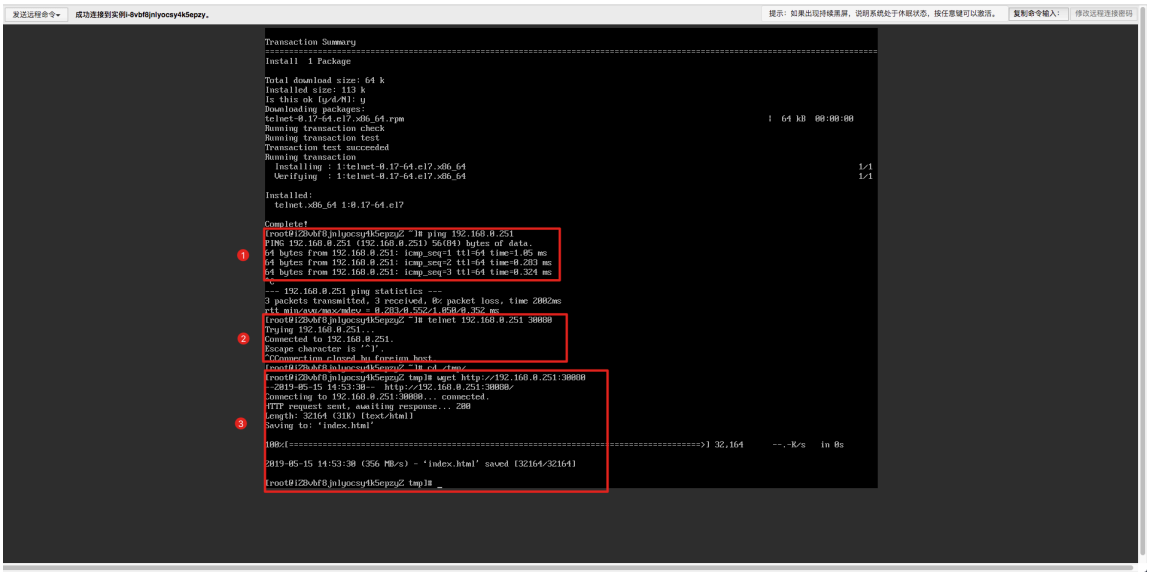
   本例中，我们通过随机登录一台Kubernetes 集群的ECS服务器，请求对应的NodePort服务，看能否正常访问服务；示例是30080端口，ECS机器IP为 192.168.0.250。

   a. 在左侧导航栏选择集群 > 节点，在节点列表页面，选择目标集群Kubernetes-piggymetrics-cluster，单击目标实例。



   b. 在实例基本信息页面，单击远程连接，根据界面提示登录该实例。

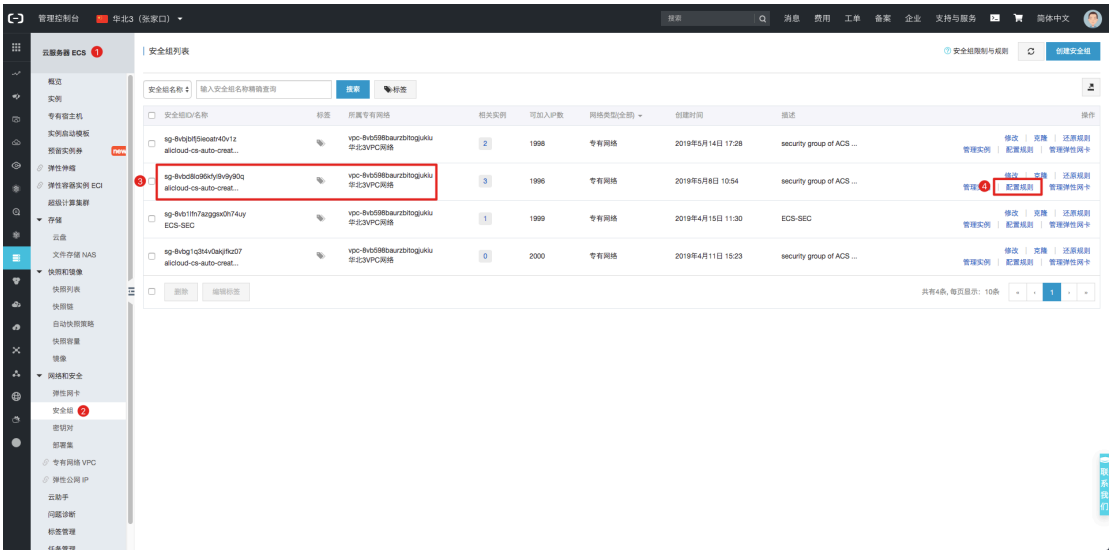
   c. 在192.168.0.250机器上，通过`ping`、`telnet`、`wget`等命令访问192.168.0.251:30080；确认该NodePort Service是否正常。



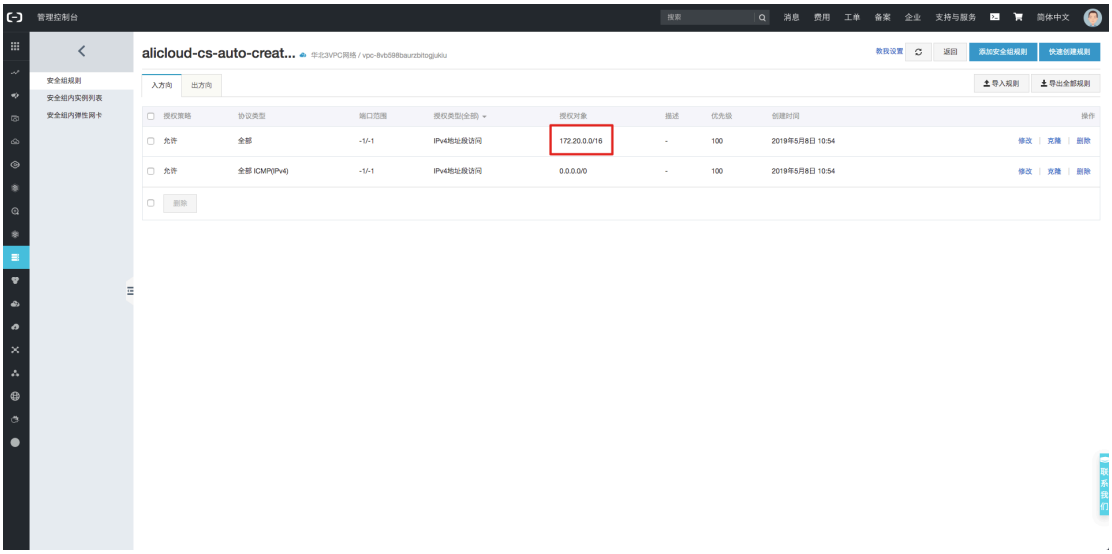
   由于容器服务针对每个集群ECS实例都会自动创建一个网络安全组，其默认只允许集群内容器所属网段IP访问各个ECS实例端口，集群外部无法访问（SLB不受影响）；因此，我们在

上述NodePort Service测试时选择利用Kubernetes 集群内的ECS实例做测试，您可以通过如下操作，访问网络安全组。

A. 登录ESC管理控制台，在右侧导航栏选择网络和安全 > 安全组，在目标安全组右侧单击配置规则。



B. 在安全组规则页面，在入方向页签，可以看到该网络安全组的IP及端口。



验证NodePort服务正常后，您需要通过修改Swarm SLB配置，修改Swarm集群所属SLB的监听规则，将我们的Kubernetes 集群的NodePort服务挂载到Swarm集群的SLB中。

10.7.4 修改Swarm SLB配置

在SLB中，监听到请求之后，需要将请求转发到后面服务器；而后端服务器可以通过3种方式挂载到监听规则中，分别为默认服务器组、主备服务器组、虚拟服务器组。

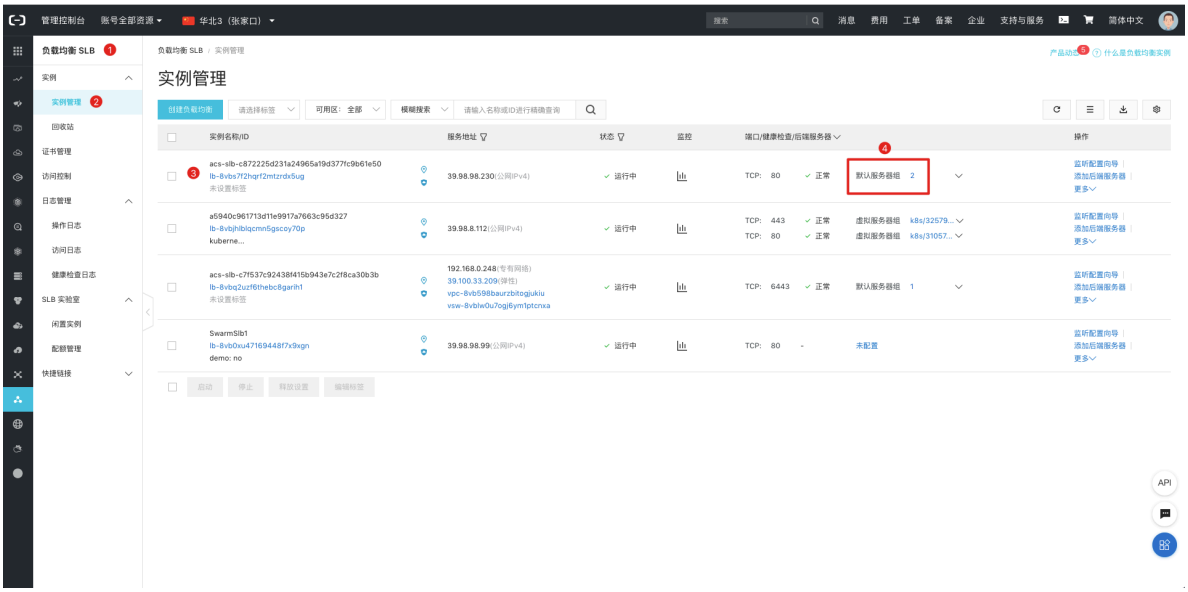
其中，默认服务器组和主备服务器组要求组内机器的后端机器端口是一样的，无法满足我们Kubernetes 引流的诉求（Swarm机器默认是通过9080端口从SLB引流，而Kubernetes 集群在

我们前一步配置NodePort服务时，只能通过30000~32767之间端口做引流）。所以，我们只能通过虚拟服务器组的方式配置后端服务器，进而实现给Kubernetes 集群机器引流。您可以按照以下步骤操作。

确认后端服务器组配置

确认当前Swarm SLB是否是已通过虚拟服务器组方式挂载后端机器。

1. 登录负载均衡管理控制台，在左侧导航栏选择实例 > 实例管理。
2. 进入实例管理页面，在目标实例右侧查看后端服务器的类型。

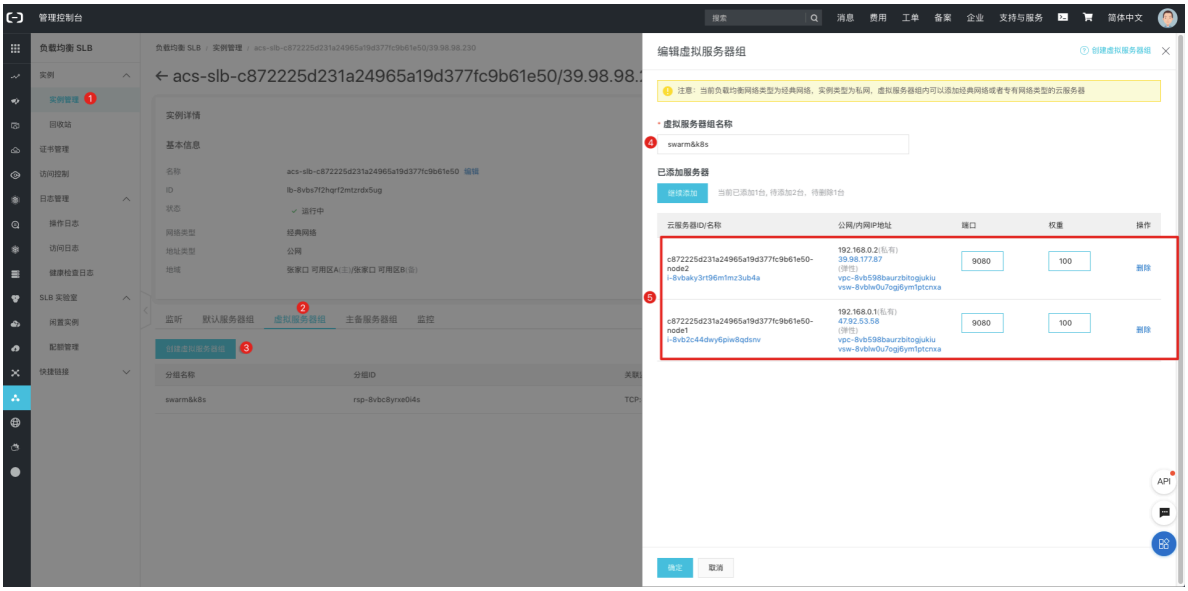


- 如果后端服务器类型为默认服务器，请继续下一步骤。
- 如果后端服务器类型为虚拟服务器组。则直接进入调整虚拟服务器组步骤。

创建虚拟服务器组

1. 在实例管理页面，单击目标实例，在实例详情页面，单击虚拟服务器组页签。

2. 单击创建虚拟服务器组，从右侧滑出编辑虚拟服务器组页面。



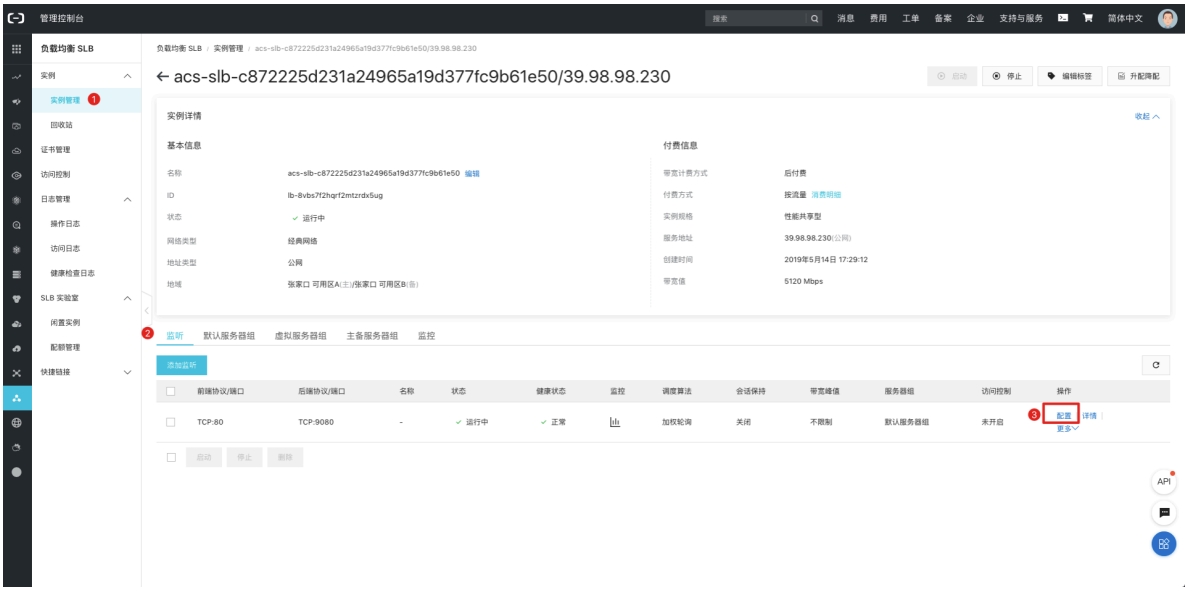
3. 填写对应的内容并单击确定，创建一个虚拟服务器组，请参考管理虚拟服务器组。

> **说明:**
>
> - 添加云服务时，swarm集群机器需全部选择，避免万一切换到该虚拟服务器组时，后端服务器撑不住已有流量，而Kubernetes 集群机器可以任选1或多台。
> - swam监听端口和权重设置一定要跟原先默认服务器组或主备服务器组配置保持一致。

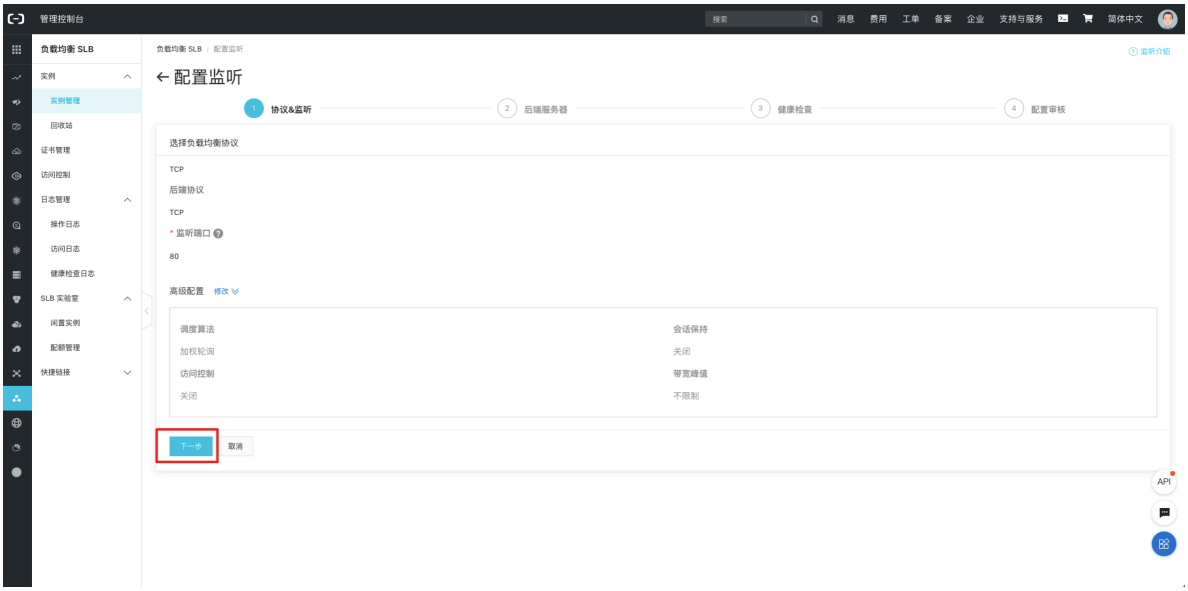
修改监听规则，切换到新服务器组

在创建好虚拟服务器组后，您需要修改SLB监听规则，将服务器组从默认服务器组或主备服务器组，切换到上述创建的虚拟服务器组上。

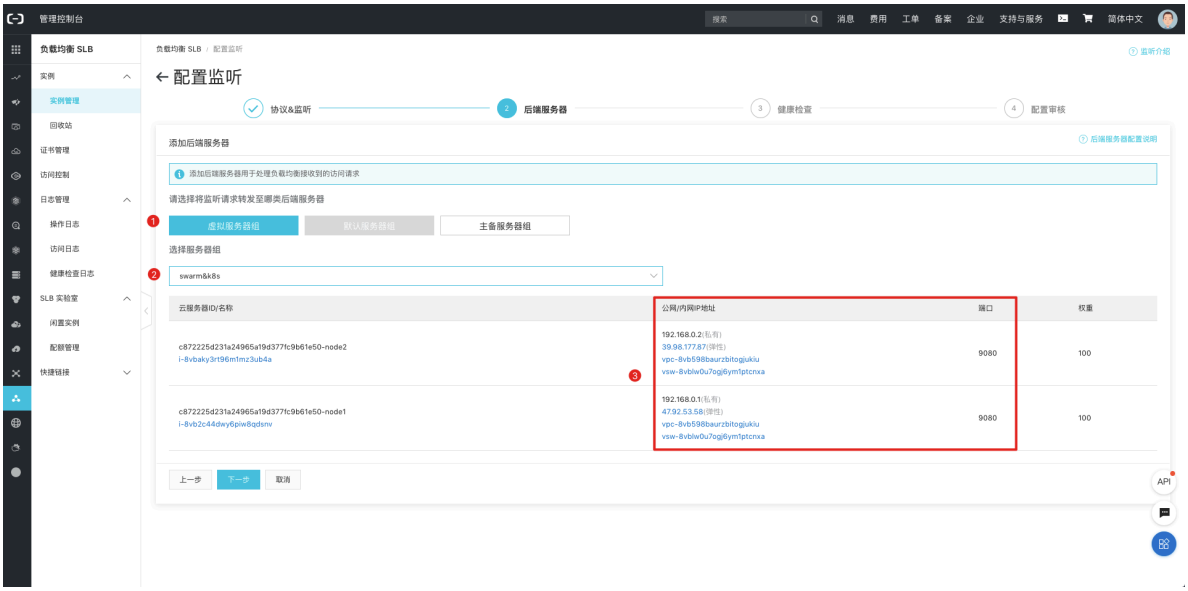
1. 在实例管理页面，单击目标实例，进入实例详情页面，在监听页签目标端口右侧单击配置。



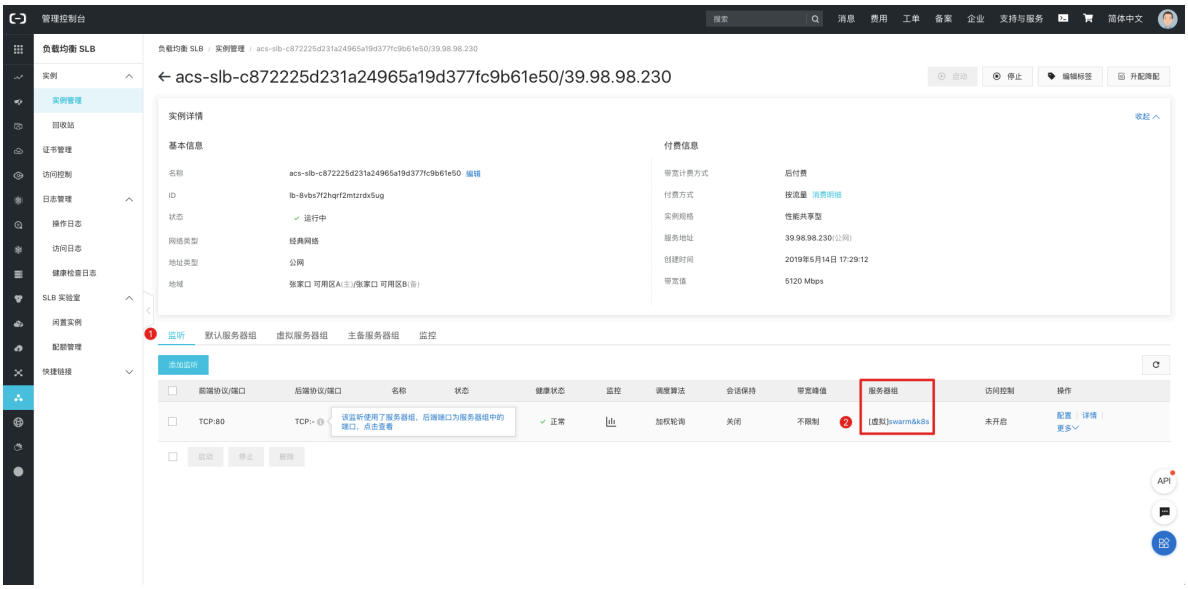
2. 在配置监听页面，修改配置并单击下一步。



3. 在后端服务器中，单击虚拟服务器组，选择服务器群组为swarm&K8s，确认公网/内网IP地址及端口（需要重点确认虚拟服务器组中的swarm机器列表及端口是否正确）正确后，单击下一步保存更新。



4. 返回监听页签，可以看到目标端口的服务器组已切换为虚拟swarm&k8s。



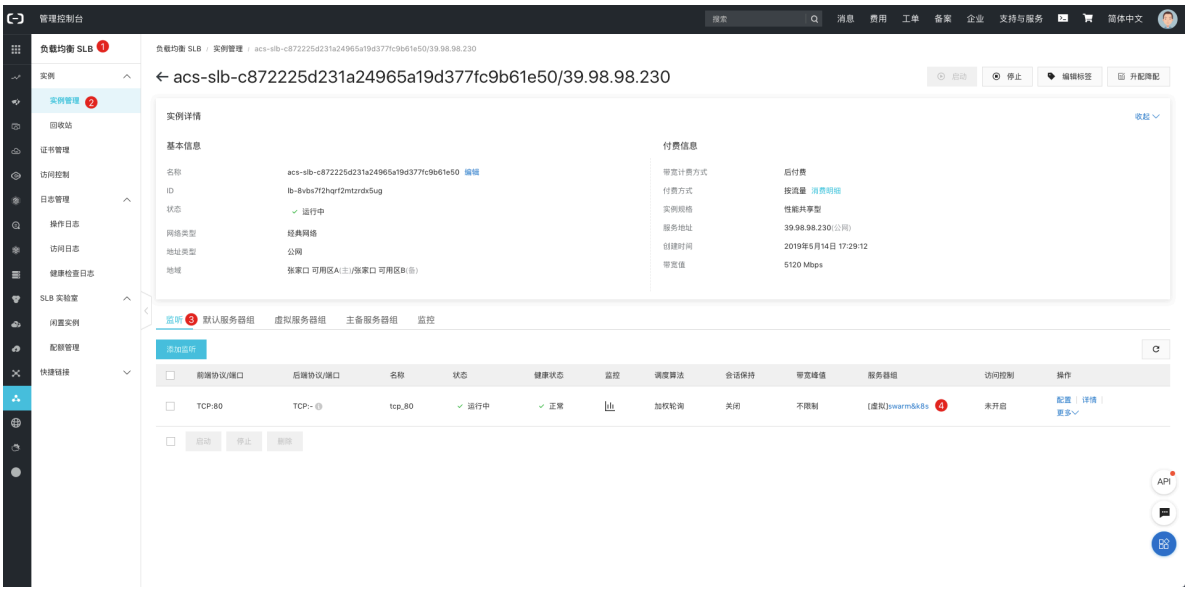
确认业务请求是否正常

请重点监控一段时间，确认线上流量业务请求处理是否正常，可通过云监控 > 主机监控，云监控 > 日志监控或者客户自身搭建的监控体系等手动观察业务服务响应情况。

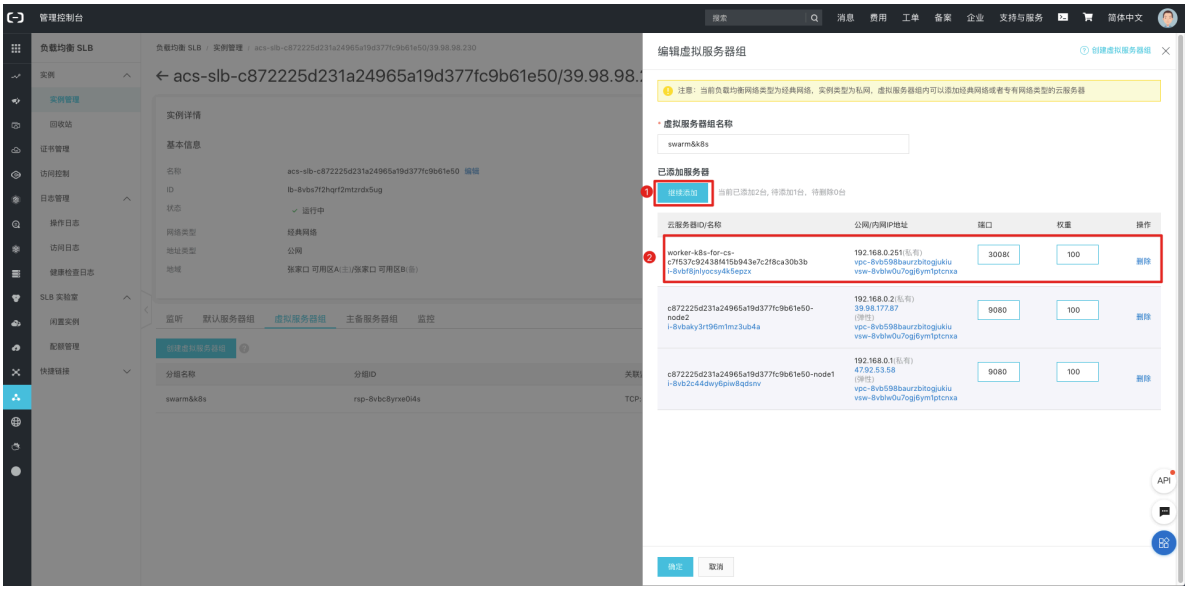
调整虚拟服务器组

在确认将SLB 的后端服务器切换成虚拟服务器组类型没问题之后，下一步我们要做的事情就是调整虚拟服务器组，加入Kubernetes 机器，从而实现将生产流量部分引流到Kubernetes 集群。

1. 在实例管理页面，单击目标实例，进入实例详情页面，在监听页签目标端口右侧单击服务器组。



2. 在虚拟服务器组页签，目标服务器组右侧单击编辑，在右侧滑出栏虚拟服务器编辑页面中，可通过添加按钮，选择添加Kubernetes 服务器并保存，即完成Kubernetes 机器的挂载引流。



说明:

- Kubernetes 集群机器可以按业务灰度流量需要选择加入一台/多台，方便做流量控制。
- Kubernetes 机器端口设置，需要跟前面NodePort设置的端口号一样。

  区间应该为30000~32767，这里示例值为30080端口，映射到Kubernetes 中的gateway-swarm-slb服务。
- 如有需要，可以通过权重更精细地控制Kubernetes 集群的流量占比。

确认流量进去Kubernetes 集群

请重点监控一段时间，确认线上流量业务请求处理是否正常；并确认Kubernetes 集群是否按虚拟服务器组设置的机器比例及权重配置引流成功。

说明:

这里只是通过 Swarm SLB 引入部分生产流量来验证 Kubernetes 集群业务功能是否正常；对应的 Swarm SLB 此时并不会下线，需要等到后续客户端切流完成之后，在下线Swarm集群章节下线 Swarm 集群及 Swarm SLB 。

10.8 客户端流量切换

本文档主要通过生产环境灰度引流验证，确认Kubernetes 集群业务服务正常之后，我们如何逐步将线上流量切换到Kubernetes 集群。主要包括基于DNS域名解析切流、通过客户端切流 两种方式。

前提条件

为了降低业务风险，切流之前，请务必确保 Kubernetes 集群应用服务正常，您可通过测试域名或备案域名对Kubernetes 集群再做一次业务功能回归验证，请参见应用回归测试。

DNS切流

若客户端是通过备案域名访问集群服务，可通过调整DNS配置逐步实现生产流量不停机迁移。

操作步骤

1. 获取新Kubernetes 集群默认SLB IP地址。本示例为*39.98.8.112*。
   a. 登录容器服务管理控制台，在左侧导航栏选择路由与负载均衡 > 路由，选择集群`k8s-piggymetrics-cluster`及命名空间`default`，获取目标路由的端点为*39.98.8.112*。
2. 前往备案域名所在DNS服务商，将 Kubernetes 集群的 SLB的IP地址*39.98.8.112*添加到备案域名*piggymetrics.com*的后端IP列表中。
3. 请重点观察线上生产流量访问Kubernetes集群是否正常。请参见确认应用日志。

   说明:
   您也可基于自身的监控体系，确认 Kubernetes 集群的流量和业务服务是否正常。
4. 观察一段时间，在确认生产流量访问Kubernetes集群正常后，请前往备案域名所在DNS服务商，将Swarm集群的SLB IP地址从备案域名的后端IP地址列表中删除。

   本示例，将*39.98.98.230* 这个Swarm集群SLB IP地址从piggymetrics.com域名的后端IP地址中删除。

客户端切流

若客户端通过HostIP访问Swarm SLB IP 或者通过测试域名访问Swarm集群；则由于测试域名跟集群标识有关，导致Swarm集群测试域名与Kubernetes 集群测试域名不一致；那么在原容器服务Swarm集群基础上无法做到不停机迁移业务，请选择业务量小的时候进行业务迁移。

操作步骤

1. 获取新Kubernetes 集群默认SLB IP地址。本示例为*39.98.8.112*。

   a. 登录容器服务管理控制台，在左侧导航栏选择路由与负载均衡 > 路由，选择集群k8s-piggymetrics-cluster及命名空间default，获取目标路由的端点为*39.98.8.112*。

2. 通过升级客户配置或代码方式，将*39.98.8.112*地址添加到客户端访问列表中。

3. 请重点观察线上生产流量访问Kubernetes集群是否正常。请参见确认应用日志。

   说明:

   您也可基于自身的监控体系，确认 Kubernetes 集群的流量和业务服务是否正常。

4. 观察一段时间，在确认生产流量访问Kubernetes集群正常后，请通过升级客户配置或代码方式，将Swarm集群的SLB IP地址从备案域名的后端IP地址列表中删除。

   本示例，将*39.98.98.230* 这个Swarm集群SLB IP地址从客户端访问列表中删除。

10.9 下线Swarm集群

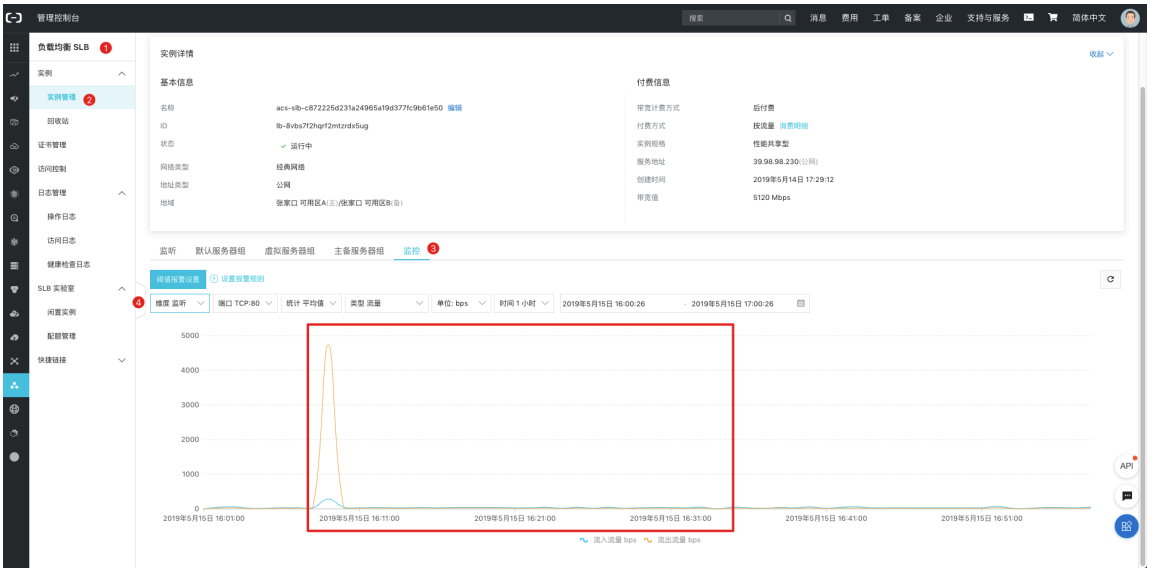
确认Swarm集群流量

1. 确认SLB流量。

   在下线Swarm集群前，需要重点确认，老的Swarm SLB是否还有流量进来，可以通过SLB提供的监控视图查看，其支持按SLB实例或监听规则视角查看，您可以通过如下操作进行查看。

   a. 登录负载均衡管理控制台，在实例管理页面，单击目标实例，进入实例详情页面。

   b. 在监控页签，设置监听规则等，可以看到监控视图。



2. 确认应用流量。

   建议通过客户自己的业务监控体系重点观察一段时间，确认老的Swarm集群是否还有业务流量进来；避免出现客户端升级未完成或DNS有缓存，导致有残余流量存在，进而影响线上业务。

下线Swarm集群

在确认K8S集群正常对外提供服务，老的Swarm集群没有残余流量后，我们可以开始着手准备下线老Swarm集群；主要包括以下3个步骤：

1. 备份 Swarm 集群配置。

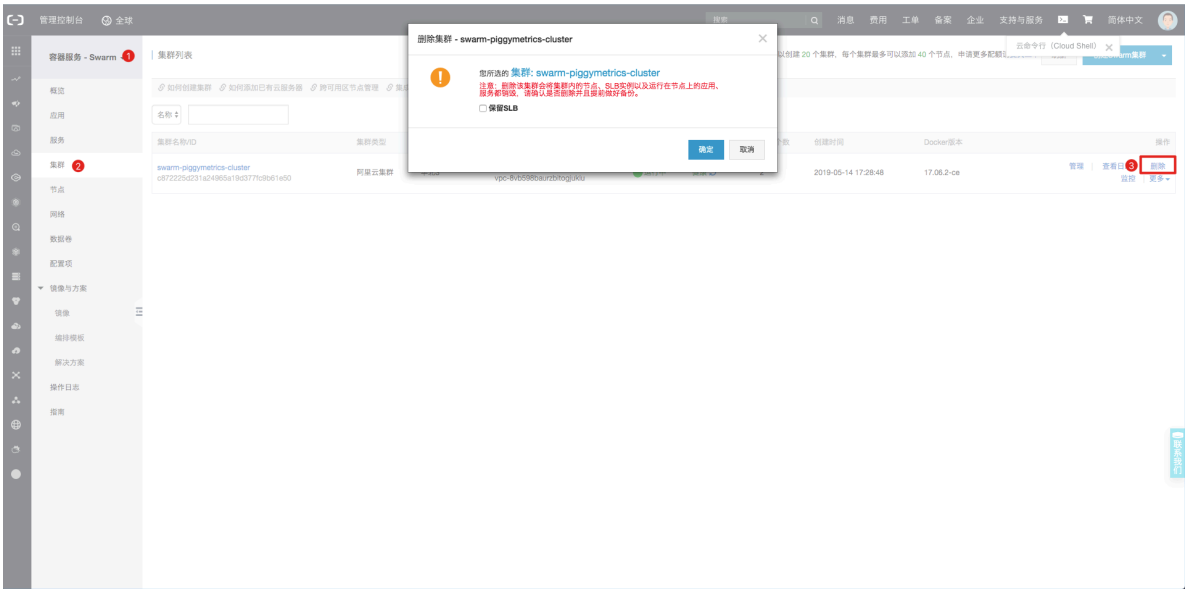
   为了确保 K8S 集群运行中出现未知问题时，能快速回溯原 Swarm集群配置，建议备份一份老 Swarm 集群的配置，避免配置丢失不可恢复，包括但不仅限于以下维度的配置：

   - 集群 Node 标签、网络、数据卷配置、配置项、SLB 配置等。
   - 各应用编排文件、日志等。

2. 下线Swarm集群资源。

   注意：
   在Swarm集群中，Node节点ECS、应用、服务、SLB等云产品都挂在Swarm集群上，集群被删除，其关联的云产品都会被删除，请慎重确认。

   登录容器服务管理控制台，在左侧导航栏选择集群 > 集群，在目标集群右侧单击删除，删除集群。



3. 下线云盘资源。

云盘只能挂载在单一ECS机器上，所以在数据卷迁移章节我们强调过，如果云盘是作为数据盘需要迁移的，则只能选择停机迁移到K8S集群。

如果云盘只是作为数据盘用于数据收集且无须带数据迁移的，则在K8S集群里面可以新建一个云盘作为数据存储；而在老的Swarm集群下线之后，其对应挂载的云盘并不会自动释放，只会变成待挂载状态。

> **说明：**
> 请按业务需要先收集或备份好云盘上的数据再释放云盘，避免数据丢失不可恢复。

若需释放Swarm集群云盘，则需到云盘控制台手动释放。

a. 登录ESC管理控制台，在左侧导航栏选择存储 > 云盘。

b. 在磁盘列表页面，在目标云盘右侧选择更多 > 释放，完成Swarm集群云盘的释放。