

ALIBABA CLOUD

阿里云

弹性容器实例
最佳实践

文档版本：20220705

 阿里云

法律声明

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

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

通用约定

格式	说明	样例
 危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 危险 重置操作将丢失用户配置数据。
 警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 警告 重启操作将导致业务中断，恢复业务时间约十分钟。
 注意	用于警示信息、补充说明等，是用户必须了解的内容。	 注意 权重设置为0，该服务器不会再接受新请求。
 说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击设置> 网络> 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击确定。
Courier字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
<i>斜体</i>	表示参数、变量。	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.使用并集成ECI Terminal	05
2.Redis集群应用	06
3.虚拟节点实现在线业务弹性伸缩	13
4.通过virtual-kubelet-autoscaler将Pod自动调度到虚拟节点	19
5.运行Job任务	24
6.使用抢占式实例运行Job任务	25
7.使用GitLab CI	29
8.在ASK上快速搭建Jenkins环境及执行流水线构建	30
9.搭建WordPress应用	35
10.搭建Spark应用	37
11.搭建TensorFlow应用	45
12.通过ASK对接RDS	49
13.常用数据源对接	57
13.1. 在ECI中访问HDFS的数据	57
13.2. 在ECI中访问OSS数据	68

1.使用并集成ECI Terminal

本文为您介绍如何使用ECI Terminal，并将ECI Terminal集成到自有系统中。

使用ECI Terminal

您在使用ECI的时候，如果想要进入到容器内部执行相关操作，可以调用`ExecContainerCommand`来获取WebSocket Uri。但是WebSocket Uri并不能在浏览器中直接打开使用，此时，您可以使用该接口返回的Http Url。

Http Url的有效时间为30秒，您需要在接口调用后的30秒内在浏览器中打开Http Url，即可进入到容器内部。

```
Welcome to Alibaba Cloud Elastic Container Instance!
This connection has been audited, you can view the audit log on the ECI console.
How to integrate the current page into your system: https://help.aliyun.com/document_detail/202846.html

ECI Terminal

eci information:
cn-shanghai::eci-uf65xfvuz0 3zdy::j: test::test1
```

通过这种方式进入容器的操作将会被审计，审计信息包括：

1. 获取Http Url的OpenAPI请求所对应的时间、RequestId、客户端IP和请求参数。
2. 通过浏览器打开Http Url所对应的客户端IP。

说明

审计信息最多保留一个月，您可以在[弹性容器实例控制台](#)的Terminal 使用日志页面查看相关的审计记录。

将ECI Terminal集成到自有系统

在自有系统中，调用`ExecContainerCommand`获取到Http Url后，您可以通过以下方式集成：

- 在独立窗口打开
可以由前端应用为Http Url打开一个新的窗口。
- 内嵌到当前页面
通过iframe的方式内嵌到自有系统的页面中。

2.Redis集群应用

本文介绍怎样以容器化方式部署Redis集群。

Redis集群模式说明

Redis支持以集群模式运行，在该模式下，Redis将所有存储空间分为16384个哈希槽，集群中的每个Master节点负责N个哈希槽（一个数据分片），当用户写入一条数据时，Redis计算其哈希槽，然后将数据写在负责该哈希槽的节点上。且每个Master节点可以添加一个或多个Slave节点，当某个Master节点不可用时，其Slave节点自动代替Master节点继续工作。

由此可见，在Redis集群模式下，我们可获得更高的吞吐量，和一定程度的可用性。需要注意的是，在集群模式下，Redis仍不能保证数据零丢失，详见[Redis官方文档](#)。

基本原理

本文以Redis官方镜像6.0.8版本作为示例，创建一个6个节点的Redis集群，其中3个Master节点，3个Slave节点。因为每个节点有自己的状态和标识，所以使用Statefulset来创建Pod，此外需要为每个节点挂载一个云盘，用以持久化节点数据。建议选择较新Redis版本，如果Redis版本低于5.0，则初始化集群所用的命令可能会有所不同。

开始构建

1. 创建一个Configmap对象，用来存储和管理Redis集群的配置。

```
kubectl create -f - <<<'
apiVersion: v1
kind: ConfigMap
metadata:
  name: redis-cluster
data:
  redis.conf: |
    bind 0.0.0.0
    port 6379
    cluster-announce-bus-port 16379
    cluster-enabled yes
    appendonly yes
    cluster-node-timeout 5000
    dir /data
    cluster-config-file /data/nodes.conf
    requirepass pass123
    masterauth pass123
'
```

字段dir为Redis节点数据的持久化目录，所以Pod的 `/data` 目录应当是一个PVC目录。字段cluster-config-file是Redis集群的节点信息，由Redis节点自动生成和修改，该位置也应当是一个持久化的目录，以便节点宕机恢复后能够找到集群中的其他节点，并继续工作。

2. 创建Statefulset资源需要依赖Service对象，我们提前创建一个headless类型的Service。

```
kubectl create -f - <<<'
apiVersion: v1
kind: Service
metadata:
  name: redis-cluster-svc
spec:
  clusterIP: None
  selector:
    app: redis-cluster
'
```

3. 创建Statefulset时引用前面的Service，将Configmap挂载到每一个Pod的 `/config`，并且为每一个Pod创建一个PVC，挂载到 `/data`。

```
kubectl create -f - <<<'
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: redis-cluster
spec:
  serviceName: redis-cluster-svc
  replicas: 6
  template:
    metadata:
      labels:
        app: redis-cluster
    spec:
      nodeName: virtual-node-eci-0
      terminationGracePeriodSeconds: 10
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - podAffinityTerm:
                labelSelector:
                  matchExpressions:
                    - key: app
                      operator: In
                      values:
                        - redis-cluster
                topologyKey: kubernetes.io/hostname
                weight: 100
      containers:
        - name: redis
          image: redis:6.0.8
          command: ["redis-server", "/config/redis.conf"]
          ports:
            - name: redis
              containerPort: 6379
              protocol: TCP
            - name: election
              containerPort: 16379
              protocol: TCP
      volumeMounts:
```

```

- name: redis-conf
  mountPath: /config
- name: pvc-essd-redis-data
  mountPath: /data
volumes:
- name: redis-conf
  configMap:
    name: redis-cluster
    items:
      - key: redis.conf
        path: redis.conf
volumeClaimTemplates:
- metadata:
    name: pvc-essd-redis-data
  spec:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: alicloud-disk-essd
    resources:
      requests:
        storage: 20Gi

```

接下来等待Statefulset中的所有Pod达到Ready状态。

```

kubectl get statefulset redis-cluster -o wide
NAME          READY  AGE   CONTAINERS  IMAGES
redis-cluster 6/6    77m   redis       redis:6.0.8

```

4. 初始化集群，目前Redis还不支持以hostname方式初始化集群，所以先获取每个节点的IP地址。

```

kubectl get pods -l app=redis-cluster -o wide

```

输出类似以下信息。

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED	READY	STATUS	RESTARTS	AGE	IP	NODE
redis-cluster-0	1/1	Running	0	83m	192.168.0.245	virtual-node-eci-0
<none>	<none>					
redis-cluster-1	1/1	Running	0	83m	192.168.0.246	virtual-node-eci-0
<none>	<none>					
redis-cluster-2	1/1	Running	0	81m	192.168.0.247	virtual-node-eci-0
<none>	<none>					
redis-cluster-3	1/1	Running	0	81m	192.168.0.248	virtual-node-eci-0
<none>	<none>					
redis-cluster-4	1/1	Running	0	80m	192.168.0.249	virtual-node-eci-0
<none>	<none>					
redis-cluster-5	1/1	Running	0	80m	192.168.0.250	virtual-node-eci-0
<none>	<none>					

登录到其中一个Redis节点。

```

kubectl exec -ti redis-cluster-0 bash

```


执行初始化命令，共有6个节点，当选项 `--cluster-replicas` 指定为1时，表示为每个Master节点分配一个Slave节点，这样集群中刚好3个Master节点和3个Slave节点。

```
redis-cli -a pass123 --cluster create \  
192.168.0.245:6379 \  
192.168.0.246:6379 \  
192.168.0.247:6379 \  
192.168.0.248:6379 \  
192.168.0.249:6379 \  
192.168.0.250:6379 \  
--cluster-replicas 1
```

输出类似以下信息表示初始化成功。

```
[OK] All nodes agree about slots configuration.  
>>> Check for open slots...  
>>> Check slots coverage...  
[OK] All 16384 slots covered.
```

5. 使用Redis，现在进入集群中的任意一个Pod中都可以访问Redis服务，前面我们创建了一个headless类型的Service，kubernetes集群会为该服务分配一个DNS A记录，格式为

`my-svc.my-namespace.svc.cluster-domain.example`，每次访问该服务名时，将随机解析到其中一

个Redis节点上。

我们进入redis-cluster-0节点中，尝试登录到Redis。

```
redis-cli -a pass123 -c -h redis-cluster-svc.default.svc.cluster.local -p 6379  
192.168.0.248> set k1 v1  
OK  
192.168.0.248> get k1  
"v1"
```

扩容

我们目前的部署方式不支持动态扩容集群，主要的问题是每次新增节点都需要为集群中的所有节点重新分配哈希槽，我们可以在Redis镜像中添加脚本，以便每个Pod启动时自动完成该操作，但是当集群中的数据非常多时，连续的重新分片会导致扩容操作非常缓慢，并且有可能在重新分片期间因为耗尽Redis集群带宽而导致依赖此服务的所有客户端超时。另一个问题是没有好的策略确定新启动的Pod应该作为Master节点还是Slave节点。所以我们接下来以手动分片演示集群扩容。

1. 修改Statefulset副本，现在我们添加一个Master节点和一个Slave节点，首先增加Statefulset的副本数，使其从6增加到8。

```
kubectl scale statefulsets redis-cluster --replicas=8
```

2. 获取新节点IP，等待所有Pod达到Ready状态后，我们获取新节点的IP地址。

```
kubectl get pods -l app=redis-cluster -o wide
```

输出类似以下信息。

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
NOMINATED NODE	READINESS	GATES				
redis-cluster-0	1/1	Running	0	83m	192.168.0.245	virtual-node-eci-0
<none>	<none>					
redis-cluster-1	1/1	Running	0	83m	192.168.0.246	virtual-node-eci-0
<none>	<none>					
redis-cluster-2	1/1	Running	0	81m	192.168.0.247	virtual-node-eci-0
<none>	<none>					
redis-cluster-3	1/1	Running	0	81m	192.168.0.248	virtual-node-eci-0
<none>	<none>					
redis-cluster-4	1/1	Running	0	80m	192.168.0.249	virtual-node-eci-0
<none>	<none>					
redis-cluster-5	1/1	Running	0	80m	192.168.0.250	virtual-node-eci-0
<none>	<none>					
redis-cluster-6	1/1	Running	0	90m	192.168.0.251	virtual-node-eci-0
<none>	<none>					
redis-cluster-7	1/1	Running	0	90m	192.168.0.252	virtual-node-eci-0
<none>	<none>					

3. 添加Master节点，现在我们登录到其中一个节点，执行以下命令将redis-cluster-6添加为集群的Master节点。

```
redis-cli -a pass123 --cluster add-node 192.168.0.251:6379 192.168.0.245:6379
```

说明

其中 `192.168.0.245:6379` 为任意一个旧节点IP，连接成功后，redis-cli将自动获取其他所有节点。

查看redis-cluster-6节点的ID。

```
redis-cli -a pass123 -c cluster nodes | grep 192.168.0.251:6379 | awk '{print $1}'
2748a28ec5db88aeb6b3326d06f6e56ee0dfdab5
```

4. 重新分配哈希槽，现在共有4个Master节点，共同分担16384个哈希槽，平均每个节点分担的哈希槽数量为 $16384 / 4 = 4096$ 。现在我们从之前的三个Master分出4096个哈希槽给新节点。执行以下命令开始分配哈希槽，其中 `192.168.0.245:6379` 为任意一个旧节点IP，连接成功后，redis-cli将自动获取其他所有节点。

```
redis-cli -a pass123 --cluster reshard 192.168.0.245:6379
```

以上命令是交互式的，根据提示依次输入：准备为新节点分配的哈希槽数量（4096）、新节点的ID（2748a28ec5db88aeb6b3326d06f6e56ee0dfdab5）、从所有节点平均抽取哈希槽给新节点（all）、确认分配（yes）。

```
How many slots do you want to move (from 1 to 16384)? 4096
What is the receiving node ID? 2748a28ec5db88aeb6b3326d06f6e56ee0dfdab5
Source node #1: all
Do you want to proceed with the proposed reshard plan (yes/no)? yes
```

接下来等待分配完毕，在此期间Redis集群可以正常提供服务。

5. 添加Slave节点，将redis-cluster-7添加为redis-cluster-6的Slave节点。

```
redis-cli -a pass123 \  
---cluster add-node 192.168.0.252:6379 192.168.0.245:6379 \  
--cluster-master-id 2748a28ec5db88aeb6b3326d06f6e56ee0dfdab5
```

说明

1. 其中 `192.168.0.245:6379` 为任意一个旧节点IP，连接成功后，redis-cli将自动获取其他所有节点。
2. `2748a28ec5db88aeb6b3326d06f6e56ee0dfdab5` 为该Slave节点要跟随的Master节点ID。

现在可以看到有4个Master节点，并且每个Master有一个副本节点。

```
redis-cli -a pass123 -c cluster nodes
```

缩容

集群缩容稍微麻烦，首先Statefulset只能以和创建Pod时相反的顺序逐个删除Pod，也就是说我们要删除一个Master节和一个Slave节点，就只能删除redis-cluster-7和redis-cluster-6。其次，删除这两个节点之前必须先该节点上所有哈希槽移动到其它节点，否则将会有数据丢失，并且Redis集群将拒绝服务。

1. 重新分配哈希槽，为了避免剩下的三个Master节点出现负载倾斜的情况，我们需要将redis-cluster-6的哈希槽平均分配给redis-cluster-0、redis-cluster-2和redis-cluster-4，所以需要执行三次重新分片的操作，其操作方法与扩容时执行的分片操作基本相同，但接受节点ID变为其它剩下三个节点中的一个，来源节点为redis-cluster-6的节点ID。

```
redis-cli -a pass123 --cluster reshard 192.168.0.245:6379
```

说明

其中 `192.168.0.245:6379` 为任意一个旧节点IP，连接成功后，redis-cli将自动获取其他所有节点。

2. 修改Statefulset副本，分片完成后将Statefulset的副本数从8改为6。

```
kubectl scale statefulsets redis-cluster --replicas=6
```

删除集群

删除集群时需要删除Statefulset，Service，Configmap。

```
kubectl delete statefulset redis-cluster  
kubectl delete svc redis-cluster-svc  
kubectl delete cm redis-cluster
```

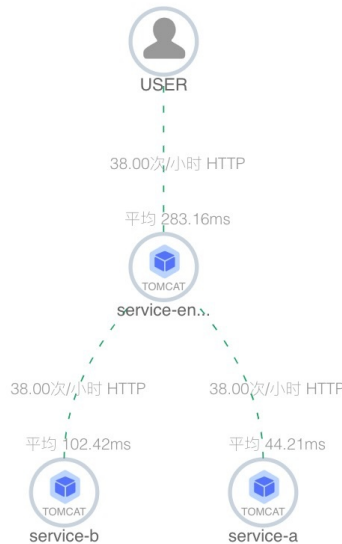
需要注意的是，Statefulset删除后，相关的PVC并不会被自动删除，需要单独删除PVC，PVC删除后，相应的云盘和数据也将被删除。

```
kubectl delete pvc -l app=redis-cluster
```

3. 虚拟节点实现在线业务弹性伸缩

本文介绍如何使用ACK和ECI完成在线业务的弹性伸缩=

准备业务镜像



整体业务请求链路如上图所示，您可以请求service-entry（模拟入口业务应用），service-entry会请求service-a（模拟耗CPU应用）和service-b（模拟耗内存应用），所以需要准备3个Docker镜像。

- service-entry: [aliyuneci/ack-vk-autoscaler-demo-service-entry:1.0](#)
- service-a: [aliyuneci/ack-vk-autoscaler-demo-service-a:1.0](#)
- service-b: [aliyuneci/ack-vk-autoscaler-demo-service-b:1.0](#)

创建ACK集群

Kubernetes 专有版
Kubernetes 托管版
Serverless Kubernetes
Kubernetes 边缘托管版（公测）

集群名称

名称为1-63个字符，可包含数字、汉字、英文字符，或“-”

资源组

未选择

地域

华北 2 (北京)	华北 3 (张家口)	华北 5 (呼和浩特)	华东 1 (杭州)	华东 2 (上海)
华南 1 (深圳)	中国 (香港)	日本 (东京)	新加坡	澳大利亚 (悉尼)
马来西亚 (吉隆坡)	印度尼西亚 (雅加达)	印度 (孟买)	美国 (弗吉尼亚)	美国 (硅谷)
英国 (伦敦)	德国 (法兰克福)			

专有网络

eci-test (vpc-2zayfb7is3wm0g8dssvsn)

[创建专有网络](#) [VPC 下 Kubernetes 的网络地址规划](#)

虚拟交换机

选择 1-3 个虚拟交换机。为保证集群高可用，建议选择不同可用区的虚拟交换机。

名称	ID	可用区	CIDR
<input type="checkbox"/> eci-test-g2	vsw-2zeb4v1uycs3t880zc1cl	华北2 可用区G	192.168.32.0/19
<input type="checkbox"/> eci-test-g	vsw-2zeh7k0m251wpeh4150al	华北2 可用区G	192.168.2.0/24
<input type="checkbox"/> eci-test-f	vsw-2zed8gsar9x3sdc96xjm0	华北2 可用区F	192.168.1.0/24

[创建虚拟交换机](#)

⊙ 请选择至少 1 个虚拟交换机

当前配置

地域: 华北2

Worker

实例规格: 实例数量: 3

⚠ 注意 使用 EIP 暴露 API Server

♥ 推荐 在 ECS 节点上安装云监控插件

⚠ 注意 使用日志服务

创建集群

生成 [OpenAPI 请求参数](#)

开通按量付费的云服务器，现金账户余额不得少于100.00元，如账户金额少于100.00元，需充值后方可开通，否则创建集群会失败。去充值

开通 云服务器、负载均衡 需要先进行实名认证，马上认证>>

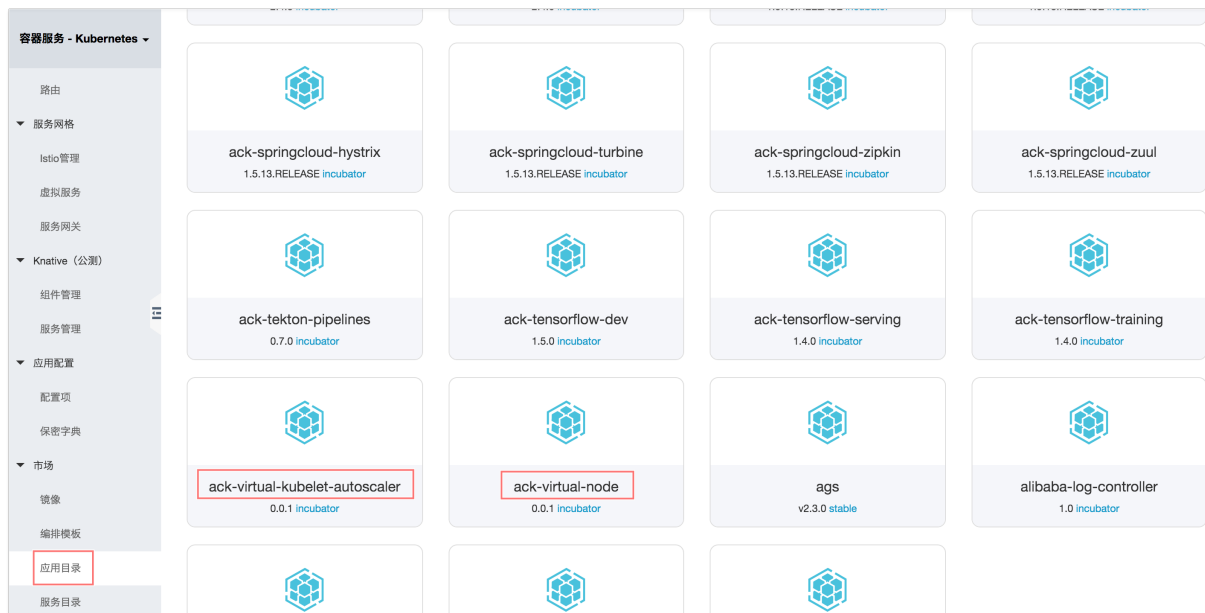
您目前可以创建 20 个集群，每个集群最多可以添加 100 个节点。如需更高配额，请提交工单申请

如需更多实例规格，请提交工单申请

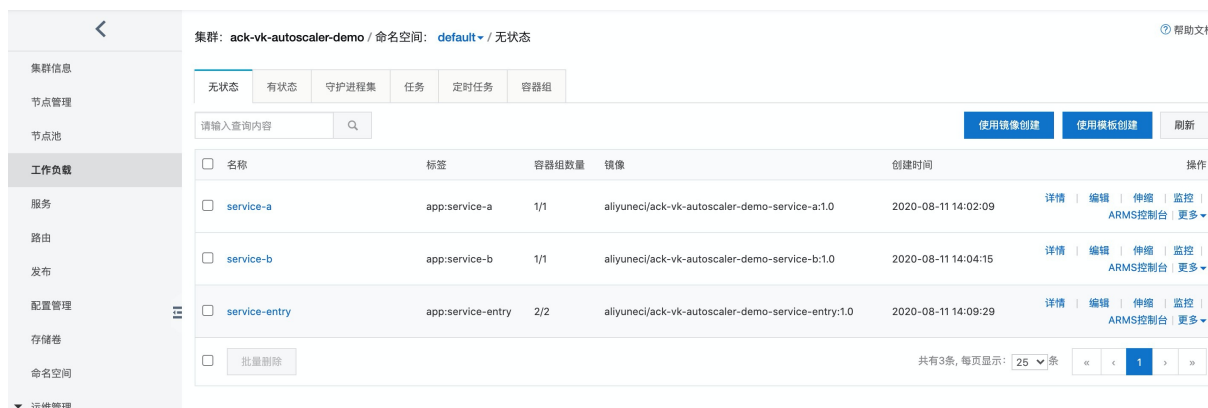
本文使用Kubernetes托管版类型的集群，使用ECS Node服务业务的正常水位，通过ack-virtual-kubelet-autoscaler来使用Virtual Node（底层资源是ECI），以服务业务的弹性水位。

部署VK和vk-autoscaler

在容器服务控制台的市场>应用目录中，依次安装ack-virtual-node和ack-virtual-kubelet-autoscaler。



部署业务应用



在容器服务控制台的应用中，根据业务形态选择对应的部署形式，本文采用无状态部署（Deployment）。填入相关内容，选择前面准备好的业务镜像，暴露相关的端口信息（service-entry为8080，service-a为8001，service-b为8002），配置service，即可创建完成。

创建service-entry的时候，需要传入2个环境变量（值来自service-a和service-b的service内部端点）。

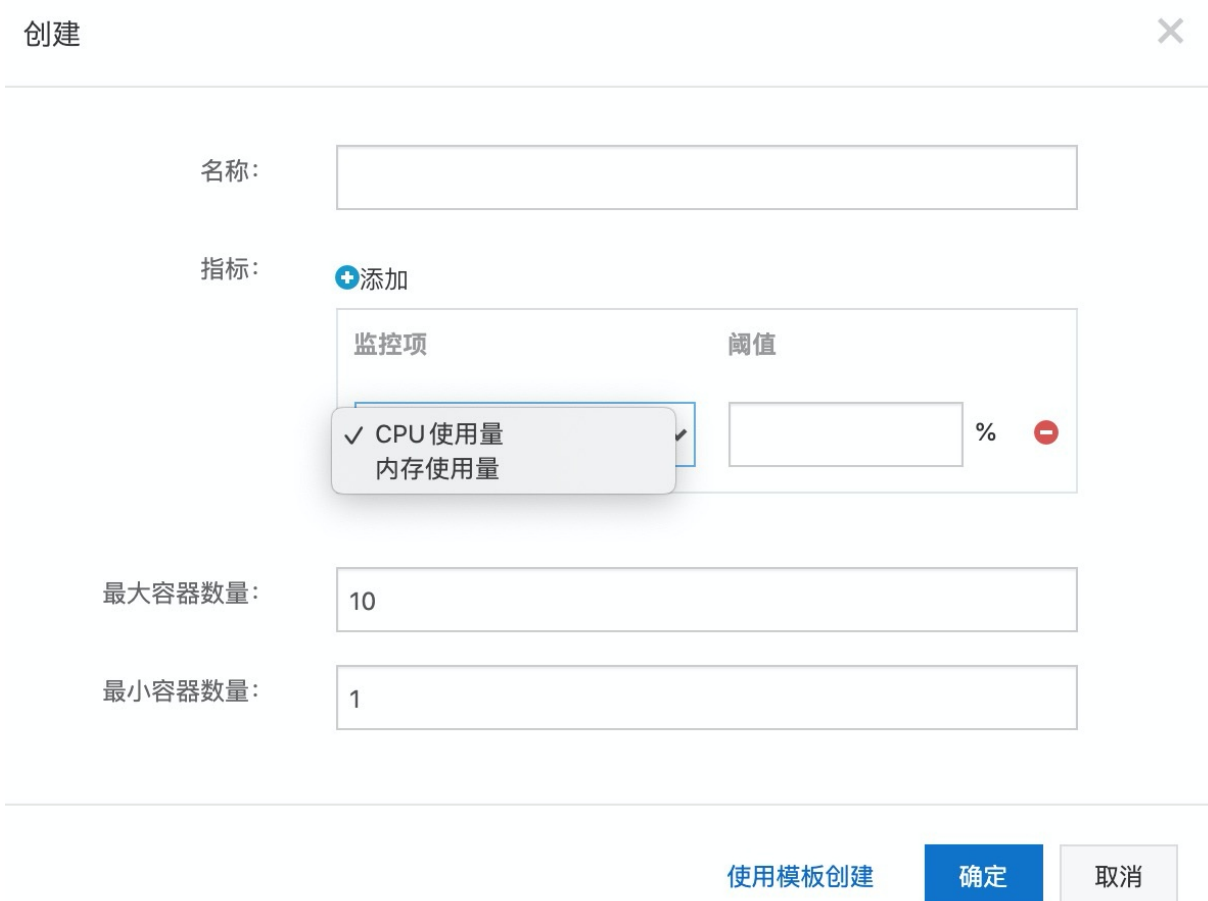


并且service-entry需要提供ingress路由，将业务能力暴露到外部环境。



配置弹性伸缩策略

在应用>无状态中，选择业务app，配置容器组水平伸缩器。



目前默认策略支持mem和cpu，也可以自己实现更多复杂的策略。

本文为上述提及的App均配置了CPU、内存维度的HPA（容器组水平伸缩器）。

容器组	访问方式	事件	容器组水平伸缩器	历史版本	日志	触发器
-----	------	----	----------	------	----	-----

HPA [创建](#)

名称	目标使用率	当前使用率	最小副本数	最大副本数	当前副本数	创建时间	操作
cpu	CPU : 50%	cpu : 1%	1	10	2	2020-08-11 16:22:33	状态 事件 编辑 删除
mem	MEMORY : 50%	memory : 71%	1	10	2	2020-08-11 16:49:33	状态 事件 编辑 删除

部署三方应用

在市场-应用目录中，可以根据业务需要，安装部署相应的App，本文中部署了以下App。

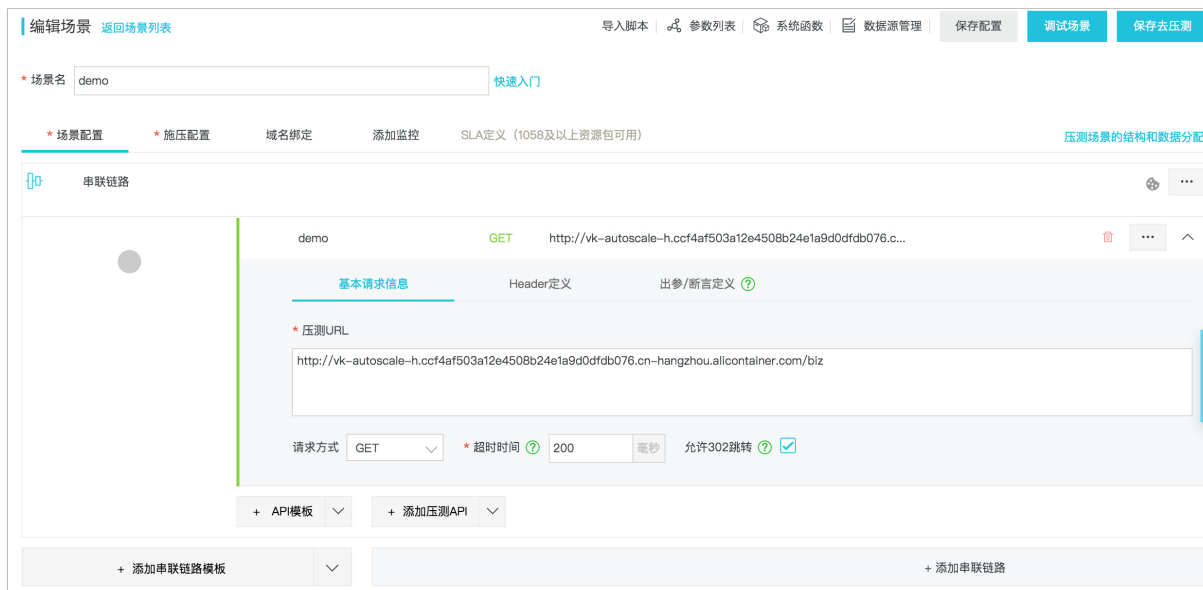
集群: ack-vk-autoscaler-demo / Helm [刷新](#)

发布名称	状态	命名空间	Chart 名称	Chart 版本	应用版本	更新时间	操作
ack-node-problem-detector	● 已部署	kube-system	ack-node-problem-detector	1.2.1	0.8.0	2020-08-11 12:19:49	详情 更新 删除
ack-virtual-kubelet-autoscaler	● 已部署	kube-system	ack-virtual-kubelet-autoscaler	0.0.1	0.0.1	2020-08-11 14:14:59	详情 更新 删除
ack-virtual-node	● 已部署	kube-system	ack-virtual-node	0.0.1	0.0.1	2020-08-11 14:14:13	详情 更新 删除
ahas	● 已部署	ahas	ack-ahas-pilot	1.8.0	1.8.0	2020-08-11 14:19:32	详情 更新 删除
arms-pilot	● 已部署	arms-pilot	ack-arms-pilot	0.1.2	1.0.2	2020-08-11 14:25:11	详情 更新 删除
arms-prometheus	● 已部署	arms-prom	ack-arms-prometheus	0.1.4	1.0.4	2020-08-11 12:19:49	详情 更新 删除

- ack-virtual-node: 通过vk来将pod创建在eci上。
- ack-virtual-kubelet-autoscaler: 通过vk动态的将pod创建在eci上（本文在eci上实现的弹性伸缩能力就是借助于此）。
- ahas: 提供应用架构自动探测，故障注入式高可用能力评测和一键流控降级等功能。
- arms-pilot: 自动发现应用拓扑、自动生成 3D 拓扑、自动发现并监控接口、捕获异常事务和慢事务。
- arms-prometheus: 对接开源 Prometheus 生态，支持类型丰富的组件监控，提供多种开箱即用的预置监控大盘，且提供全面托管的 Prometheus 服务。

PTS 压测

使用PTS对服务进行压测，来模拟业务低峰期到高峰期的一个过程。本文从5个并发施压，一直递增至40个并发。



观察效果

名称: service-b 命名空间: default
创建时间: 2020-08-11 14:04:15 标签: app:service-b
注解: deployment.kubernetes.io/revision:2 选择器: app:service-b
策略: RollingUpdate 滚动升级策略: 超过期望的Pod数量:25% 不可用Pod最大数量:25%
状态: 就绪: 12/12个, 已更新: 12个, 可用: 12个 [展开现状详情](#)

容器组	访问方式	事件	容器组水平伸缩器	历史版本	日志	触发器
类型 (全部)	对象 (全部)	信息	原因	时间		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 12	ScalingReplicaSet	2020-08-11 17:04:20		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 10	ScalingReplicaSet	2020-08-11 16:50:10		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 8	ScalingReplicaSet	2020-08-11 16:47:07		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 6	ScalingReplicaSet	2020-08-11 16:44:18		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 5	ScalingReplicaSet	2020-08-11 16:41:15		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 4	ScalingReplicaSet	2020-08-11 16:39:13		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 3	ScalingReplicaSet	2020-08-11 16:23:11		
Normal	Deployment service-b	Scaled up replica set service-b-745fdf9cdd to 2	ScalingReplicaSet	2020-08-11 16:21:09		

可以看到随着施压并发的递增，各个App（如截图中的service-b）根据自身的负载情况进行了动态的扩容。

创建时间:	2020-08-11 14:04:15	标签:	app:service-b
注解:	deployment.kubernetes.io/revision:2	选择器:	app:service-b
策略:	RollingUpdate	滚动升级策略:	超过期望的Pod数量:25% 不可用Pod最大数量:25%
状态:	就绪: 12/12个, 已更新: 12个, 可用: 12个 展开现状详情		

名称	镜像	状态 (全部)	监控	重启次数	Pod IP	节点	创建时间	操作
service-b-745fdf9cdd-5nmnk	aliyuneci/ack-vk-autoscaler-demo-service-b:1.0	Running	R	0	192.168.0.164	virtual-node-eci-0-172.20.0.14	2020-08-11 16:44:19	详情 日志 更多
service-b-745fdf9cdd-64pqt	aliyuneci/ack-vk-autoscaler-demo-service-b:1.0	Running	R	0	192.168.0.166	virtual-node-eci-0-172.20.0.14	2020-08-11 16:47:07	详情 日志 更多
service-b-745fdf9cdd-92jkn	aliyuneci/ack-vk-autoscaler-demo-service-b:1.0	Running	R	0	192.168.0.159	virtual-node-eci-0-172.20.0.14	2020-08-11 16:24:41	详情 日志 更多
service-b-745fdf9cdd-9s2jx	aliyuneci/ack-vk-autoscaler-demo-service-b:1.0	Running	R	0	192.168.0.167	virtual-node-eci-0-172.20.0.14	2020-08-11 16:47:07	详情 日志 更多
service-b-745fdf9cdd-ktv5b	aliyuneci/ack-vk-autoscaler-demo-service-b:1.0	Running	R	0	192.168.0.168	virtual-node-eci-0-172.20.0.14	2020-08-11 16:50:10	详情 日志 更多
service-b-745fdf9cdd-lkg55	aliyuneci/ack-vk-autoscaler-demo-service-b:1.0	Running	R	0	172.20.0.16	cn-beijing.192.168.0.154-192.168.0.154	2020-08-11 14:37:33	详情 日志 更多

当ECS Node上资源不足的时候，Pod被自动调度到了Virtual Node，实现了ACK+ECI的在线业务弹性伸缩的效果。

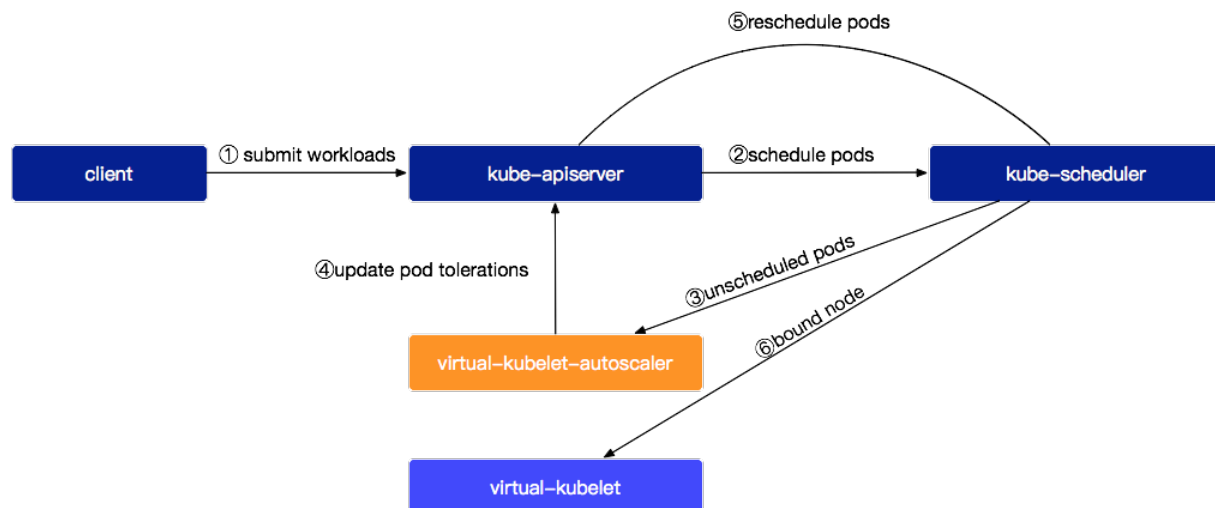
更多关于ACK+ECI的在线业务弹性伸缩的细节，请参见[在线业务弹性扩缩容最佳实践](#)。

4.通过virtual-kubelet-autoscaler将Pod自动调度到虚拟节点

本文介绍当Kubernetes集群中真实节点计算资源不足时，如何通过virtual-kubelet-autoscaler插件将Pod创建调度到虚拟节点，让您的业务享受到极致弹性的体验。

背景信息

virtual-kubelet-autoscaler插件将Pod创建调度到虚拟节点如下图所示。



前提条件

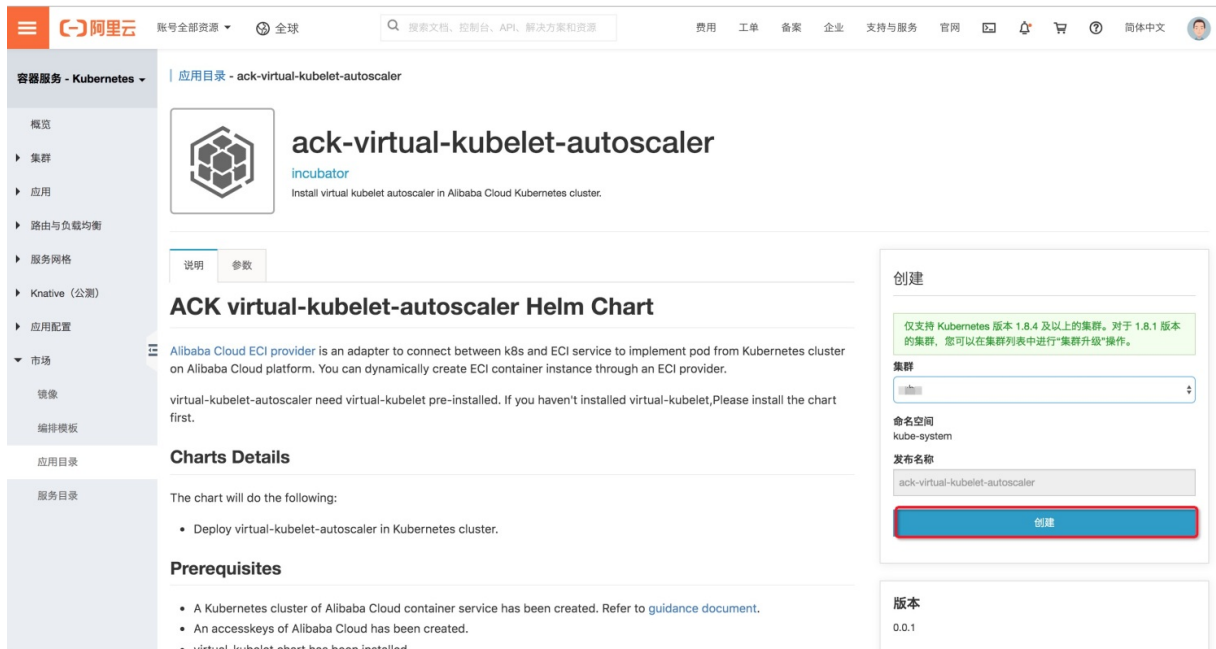
您已经创建了集群。具体操作，请参见[创建Kubernetes集群](#)。

操作步骤

1. 登录阿里云[容器服务控制台](#)查看您的集群。
2. [安装ack-virtual-node插件](#)。
3. 安装virtual-kubelet-autoscaler插件。
 - a. 选择市场>应用目录，在右上角搜索virtual-kubelet-autoscaler，单击ack-virtual-kubelet-autoscaler图标。



b. 选择您要安装的集群，单击创建。



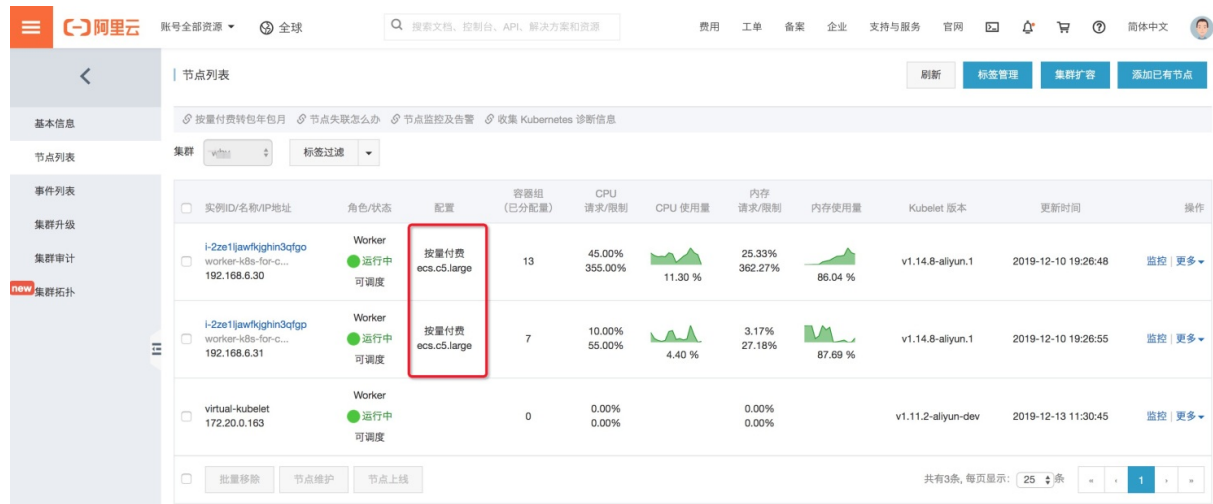
4. 您可以在集群中执行以下命令确认 ack-virtual-kubelet-autoscaler 运行状况。

```
kubectl get deploy -n kube-system
```

```
shell@Alicloud:~$ kubectl get deploy -n kube-system
NAME                                     READY   UP-TO-DATE   AVAILABLE   AGE
ack-virtual-node-affinity-admission-controller 1/1     1             1           4d
ack-virtual-node-controller              1/1     1             1           4d
alicloud-application-controller          1/1     1             1           6d16h
alicloud-disk-controller                 1/1     1             1           6d16h
alicloud-monitor-controller              1/1     1             1           6d16h
aliyun-acr-credential-helper             1/1     1             1           6d16h
coredns                                   2/2     2             2           6d16h
metrics-server                           1/1     1             1           6d16h
nginx-ingress-controller                  2/2     2             2           6d16h
tiller-deploy                            1/1     1             1           6d16h
virtual-kubelet-autoscaler                1/1     1             1           101m
```

查看真实节点资源使用状况

登录 容器服务 Kubernetes 控制台 查看您的集群节点，可以看到下图有两台ECS做为真实节点，ECS的规格是ecs.c5.large（2 vCPU 4 GiB），规格详细信息请参见计算型实例规格族c5。此时虚拟节点（virtual-kubelet）未分配容器组（Pod）。



部署deployment

准备好您的yaml文件，以下面deployment-autoscaler.yaml文件为例，配置副本数（replicas）为10，每个副本的容器声明了2 vCPU 4 GiB的计算资源。

```

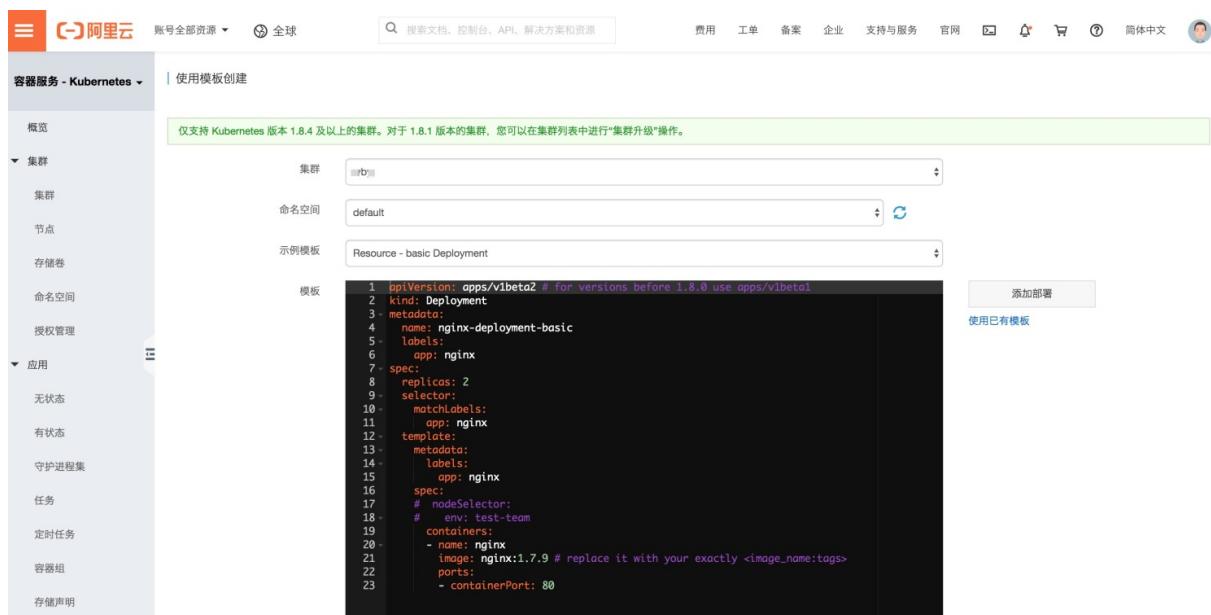
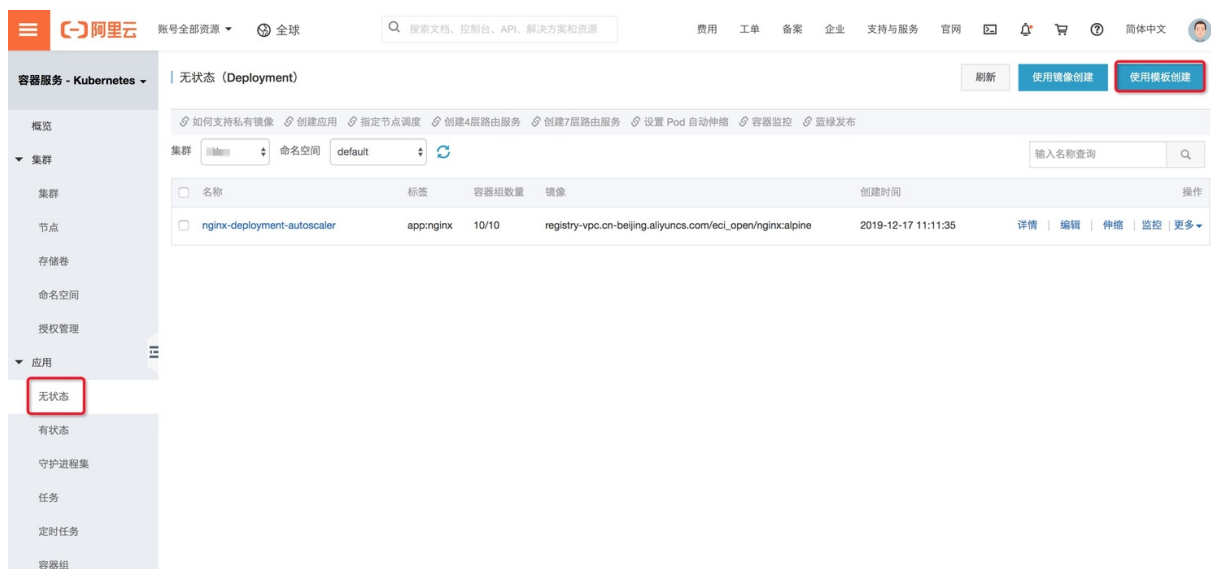
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment-autoscaler
  labels:
    app: nginx
spec:
  replicas: 10
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: registry-vpc.cn-beijing.aliyuncs.com/eci_open/nginx:alpine
        ports:
        - containerPort: 80
      resources:
        requests:
          cpu: 2
          memory: 4Gi

```

您可以在集群中执行以下命令进行创建。

```
kubectl create -f deployment-autoscaler.yaml
```

您也可以在 容器服务 Kubernetes 控制台 选择无状态>使用模板创建。



确认Pod调度情况

您可以通过以下命令确认Pod运行状况。

```
kubectl get pods
```



```

shell@Alicloud:~$ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
ack-node-pod                            1/1     Running   0           24h
nginx-deployment-autoscaler-786876b6b-5qtw4 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-5w7kg 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-84t6w 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-9b7xj 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-9r155 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-jwrjd 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-jzhq6 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-prcm7 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-t9xzq 0/1     Pending   0           15s
nginx-deployment-autoscaler-786876b6b-w6xnj 0/1     Pending   0           15s
shell@Alicloud:~$ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
ack-node-pod                            1/1     Running   0           24h
nginx-deployment-autoscaler-786876b6b-5qtw4 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-5w7kg 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-84t6w 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-9b7xj 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-9r155 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-jwrjd 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-jzhq6 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-prcm7 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-t9xzq 1/1     Running   0           32s
nginx-deployment-autoscaler-786876b6b-w6xnj 1/1     Running   0           32s

```

您可以通过以下命令查询单个Pod的事件信息。

```
kubectl describe pod nginx-deployment-autoscaler-786876b6b-5qtw4
```

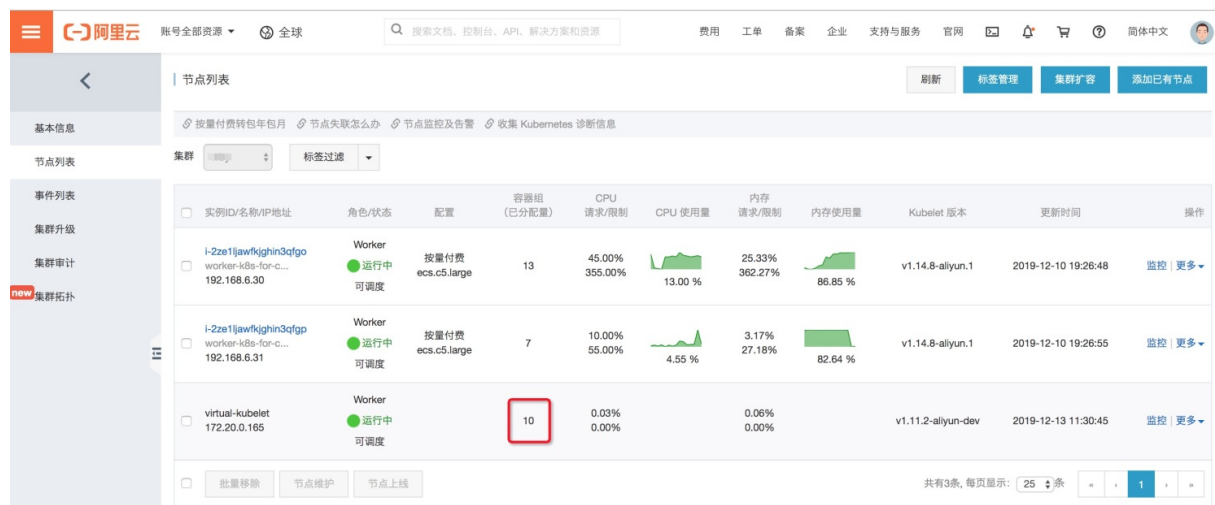
```

Events:
  Type     Reason            Age   From              Message
  ----     -
  Warning  FailedScheduling  2m37s default-scheduler 0/3 nodes are available: 1 node(s) had taints that the pod didn't tolerate, 2 Insufficient
  Normal   Scheduled         2m30s default-scheduler Successfully assigned default/nginx-deployment-autoscaler-786876b6b-5qtw4 to virtual-kube
  Normal   Pulling          2m14s kubelet, eci      pulling image "registry-vpc.cn-beijing.aliyuncs.com/eci_open/nginx:alpine"
  Normal   SuccessfulMountVolume 2m14s kubelet, eci      MountVolume.SetUp succeeded for volume "default-token-lwzln"
  Normal   Pulled           2m12s kubelet, eci      Successfully pulled image "registry-vpc.cn-beijing.aliyuncs.com/eci_open/nginx:alpine"
  Normal   Created          2m12s kubelet, eci      Created container
  Normal   Started          2m12s kubelet, eci      Started container

```

可以看到由于真实节点上计算资源不足，kube-scheduler通过virtual-kubelet-autoscaler把Pod创建二次调度到了虚拟节点上。

此时在控制台上查询集群的节点信息，可以看到虚拟节点上已经分配了10个容器组（Pod）。



5. 运行Job任务

对于短时间运行的Job任务，使用ECI来运行可以避免资源闲置浪费，降低计算成本。本文介绍如何使用ECI来运行Job任务。

对于很多Kubernetes集群，通常需要同时支撑在线和离线的多种负载，在线负载流量的波动性和离线计算任务的时间不确定性，导致在不同时刻下，负载的资源需求呈波峰波谷状，比如很多企业需要在周末、月中和月末进行大量的数据计算，在特定的时间点需要大量的计算力，以应对突发的计算资源需求。

目前Kubernetes通常的方法是通过autoscaler来自动扩容节点（约2分钟启动一个新节点），直到Pod被成功调度运行，当Pod执行完成后会自动回收临时节点。使用这种扩容方式，Pod一般需要等待2分钟甚至更多时间才能被调度运行。

针对上述场景，推荐您使用ECI来运行Job任务。ECI通过虚拟节点的方式接入Kubernetes集群，支持秒级启动，按需扩容，能够很好地提升集群的弹性能力。使用ECI来运行Job任务，您无需提前预估业务流量，预留闲置资源，在满足业务需求的同时能够有效降低使用和运维成本。

- 如果您使用阿里云ACK集群，需要部署虚拟节点，以便使用ECI来运行Job任务。
具体操作，请参见[基于ECI运行Job任务](#)。
- 如果您使用阿里云ASK集群，可以直接使用ECI来运行Job任务。
具体操作，请参见[通过ASK运行Job任务](#)。
- 如果你在云上或者线下IDC自建了Kubernetes集群，可以通过部署虚拟节点的方式接入ECI，然后将Job任务调度到ECI来运行。关于自建集群如何使用ECI，请参见[自建Kubernetes集群对接ECI](#)。

② 说明

您也可以结合使用抢占式ECI实例来节约实例使用成本。更多信息，请参见[抢占式实例运行Job任务](#)。

6.使用抢占式实例运行Job任务

抢占式ECI实例特别适合于Job任务或者无状态应用，可以有效地节约实例使用成本。本文介绍如何使用抢占式ECI实例来运行Job任务。

背景信息

使用抢占式实例前，请了解以下相关信息：

- 抢占式实例的市场价格随供需变化而浮动，您需要在创建实例时指定出价模式，当市场价格低于出价且资源库存充足时，就能成功创建抢占式实例。
- 抢占式实例创建成功后有一个小时的保护期，超出保护期后，如果出价低于市场价格或者资源库存不足，实例会被释放。在释放前三分钟左右，ECI会通过Kubernetes Events事件通知的方式告知您。在此期间，您可以做一定的处理来确保业务不受实例释放所影响（如切断该实例的业务入口流量）。
- 抢占式实例由于竞价失败而释放后数据会随之一起释放，但实例信息会保留（状态变更为Failed，失败原因为BidFailed）。如果您有重要数据需要保留，建议配置持久化存储，如云盘，NAS等。

更多信息，请参见[抢占式实例概述](#)和[创建抢占式实例](#)。

配置方法

您可以在Pod的声明中设置Annotation来指定创建抢占式ECI实例。相关配置项如下：

- `k8s.aliyun.com/eci-spot-strategy`：设置抢占式实例的出价策略。取值如下：
 - `SpotAsPriceGo`：系统自动出价，跟随当前市场实际价格。
 - `SpotWithPriceLimit`：自定义设置抢占实例价格上限。
- `k8s.aliyun.com/eci-spot-price-limit`：设置抢占式实例的每小时价格上限，最多支持精确到小数点后三位。当`k8s.aliyun.com/eci-spot-strategy`配置为`SpotWithPriceLimit`时必须设置。

说明

创建抢占式ECI实例时，建议您设置`k8s.aliyun.com/eci-use-specs`来指定多个规格，可以有效提升实例创建的成功率。

配置示例如下：

- `SpotAsPriceGo`

该模式下，系统跟随当前市场价格自动出价。

```
apiVersion: v1
kind: Pod
metadata:
  name: spot-as-price-go
  annotations:
    k8s.aliyun.com/eci-use-specs: ecs.sn1ne.large,2-4Gi
    # SpotAsPriceGo不需要设置价格上限；系统自动出价，跟随当前市场实际价格
    k8s.aliyun.com/eci-spot-strategy: SpotAsPriceGo
```

- `SpotWithPriceLimit`

该模式下，系统根据您设置的价格上限来出价，可能会出现以下几种情况：

- 出价 < 当前市场价格：实例会处于Pending状态，并每5分钟自动进行一次出价，直到出价等于或高于市场价格时，开始自动创建实例。
- 出价 ≥ 当前市场价格：如果库存充足，则自动创建实例。

```
apiVersion: v1
kind: Pod
metadata:
  name: spot-with-price-limit
  annotations:
    k8s.aliyun.com/eci-use-specs: ecs.sn1ne.large
    k8s.aliyun.com/eci-spot-strategy: SpotWithPriceLimit
    # SpotWithPriceLimit模式必须设置价格上限，如果设置的出价高于ECI实例当前的按量价格，使用按量价格来创建实例。
    k8s.aliyun.com/eci-spot-price-limit: "0.05"
```

配置示例

在Kubernetes中，Job负责批量处理短暂的一次性任务。使用抢占式实例运行Job任务的配置示例如下：

1. 准备Job的配置文件，命名为spot_job.yaml。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
      annotations:
        k8s.aliyun.com/eci-use-specs: ecs.t5-c1m2.large,2-4Gi
        k8s.aliyun.com/eci-spot-strategy: SpotAsPriceGo #系统自动出价，跟随当前市场实际价格
    spec:
      containers:
        - name: pi
          image: registry-vpc.cn-beijing.aliyuncs.com/ostest/perl
          command: ["perl", "-Mbignum=bpi", "-wle", "print bpi(2000)"]
          restartPolicy: Never
```

2. 创建Job。

```
kubectl create -f spot_job.yaml
```

3. 查看运行情况。

```
kubectl get pod
```

返回信息中可以看到对应Pod已经成功创建，示例如下：

NAME	READY	STATUS	RESTARTS	AGE
pi-frmr8	1/1	Running	0	5s

抢占式实例创建成功后，在一个小时的保护期内，即使市场价格高于出价，抢占式实例也不会被释放，您可以在该实例上正常运行业务。超出保护期后，当出价低于市场价格或者资源库存不足，实例将被释放。释放有以下几种方式通知方式：

- 释放前事件通知

抢占式实例在释放前三分钟左右，会产生释放事件，ECI会将释放事件同步到Kubernetes Events中，您可以通过以下命令查看：

- 查看Pod详细信息

```
kubectl describe pod pi-frmr8
```

返回信息的Events中可以看到释放事件。示例如下：

```
Events:
  Type            Reason             Age   From              Message
  ----            -
  Warning         SpotToBeReleased  3m32s kubelet, eci     Spot ECI will be released in 3 minutes
```

- 查看事件信息

```
kubectl get events
```

返回信息中可以看到释放事件。示例如下：

```
LAST SEEN   TYPE      REASON             OBJECT          MESSAGE
3m39s       Warning   SpotToBeReleased   pod/pi-frmr8   Spot ECI will be released in 3 minutes
```

- 释放后状态展示

抢占式实例释放后，实例信息仍会保留，状态变更为Failed，Failed原因为BidFailed。您可以通过以下命令查看：

- 查看Pod信息

```
kubectl get pod
```

返回信息中可以看到Pod的STATUS为BidFailed。示例如下：

```
NAME        READY   STATUS    RESTARTS   AGE
pi-frmr8   1/1    BidFailed  0          3h5m
```

- 查看Pod详细信息

```
kubectl describe pod pi-frmr8
```

返回信息中可以看到Pod的Status为Failed，Reason为BidFailed。示例如下：

```
Status:          Failed
Reason:          BidFailed
Message:         The pod is spot instance, and have been released at 2020-04-08T12:36Z
```

实例释放后，Kubernetes中的Job Controller会自动创建新的实例来继续运行任务。您可以通过

`kubectl get pod` 命令查看Pod，返回示例如下：

NAME	READY	STATUS	RESTARTS	AGE
pi-frmr8	1/1	BidFailed	0	4h53m
pi-kp5zx	1/1	Running	0	3h45m

7.使用GitLab CI

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。具体操作，请参见[使用GitLab CI运行GitLab Runner并执行Pipeline](#)。

8.在ASK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（ASK）上快速搭建Jenkins持续集成环境，并基于提供的应用示例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

- 已创建ASK集群。具体操作，请参见[创建Serverless Kubernetes集群](#)。
- 已通过kubectl连接Kubernetes集群。具体操作，请参见[通过kubectl连接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。

您可以挂载 `nfs volume`，修改 `serverless-k8s-jenkins-deploy.yaml` 文件，取消以下字段注释并配置您的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。

- i. 登录[容器服务管理控制台](#)。
- ii. 在控制台左侧导航栏中，单击[集群](#)。
- iii. 在[集群列表](#)页面中，单击目标集群名称或者目标集群右侧操作列下的详情。
- iv. 在[集群管理](#)页左侧导航栏中，选择[网络 > 服务](#)。
- v. 单击Jenkins服务的外部端点登录Jenkins。



名称	标签	类型	创建时间	集群 IP	内部端点	外部端点	操作
jenkins	appjenkins componentjenkins-master	LoadBalancer 监控信息	2020-08-17 14:41:16		jenkins:8080 TCP jenkins:30783 TCP	:8080	详情 更新 查看YAML 删除
jenkins-agent	appjenkins componentjenkins-master	LoadBalancer 监控信息	2020-08-17 14:41:16		jenkins-agent:50000 TCP jenkins-agent:30530 TCP	:50000	详情 更新 查看YAML 删除

vi. 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。

 **注意** 为了保证Jenkins系统安全，请登录后修改密码。

5. 配置Jenkins的新手入门。

- i. 在新手入门配置页面，单击安装推荐的插件。
- ii. 插件安装完成后，在新手入门的实例配置页面，单击保存并完成。
- iii. 实例配置保存完成后，单击开始使用Jenkins。

6. 获取Token的Secret。

- i. 执行以下命令查看Token。

```
kubectl get secret
```

预期输出：

NAME	TYPE	DATA	AGE
ack-jenkins-sa-token-q****	kubernetes.io/service-account-token	3	28m
default-token-b****	kubernetes.io/service-account-token	3	27h

- ii. 执行以下命令获取ack-jenkins-sa-token-q****的Secret。

```
kubectl get secret ack-jenkins-sa-token-q**** -o jsonpath={.data.token} |base64 -d
```

预期输出：

```
sdgdrh****
```

7. 创建Secret Text类型凭证。

- i. 在Jenkins系统左侧导航栏，选择系统管理。
- ii. 在管理Jenkins页面，系统配置下单击节点管理。
- iii. 在节点列表页面左侧导航栏，选择Configure Clouds。
- iv. 在配置集群页面，单击Kubernetes Cloud details...
- v. 在凭据字段，选择添加 > Jenkins。

vi. 在Jenkins 凭据提供者: Jenkins对话框, 添加Secret Text类型的凭证。

凭证的参数说明如下所示:

参数	说明
Domain	表示凭据的域。默认为全局凭据 (unrestricted)
类型	表示凭据的类型。本示例选择Secret Text。
范围	表示凭据的范围, 可选择全局或系统。本示例选择全局 (Jenkins, nodes, items, all child items, etc)。
Secret	表示凭据的Secret。本示例输入上个步骤获取的Secret。
ID	表示凭据的名称。本示例为ask-jenkins-token。
描述	表示凭据的补充说明。

vii. 单击添加。

8. 在配置集群页面配置相关参数。更多信息, 请参见Kubernetes Cloud的配置说明。

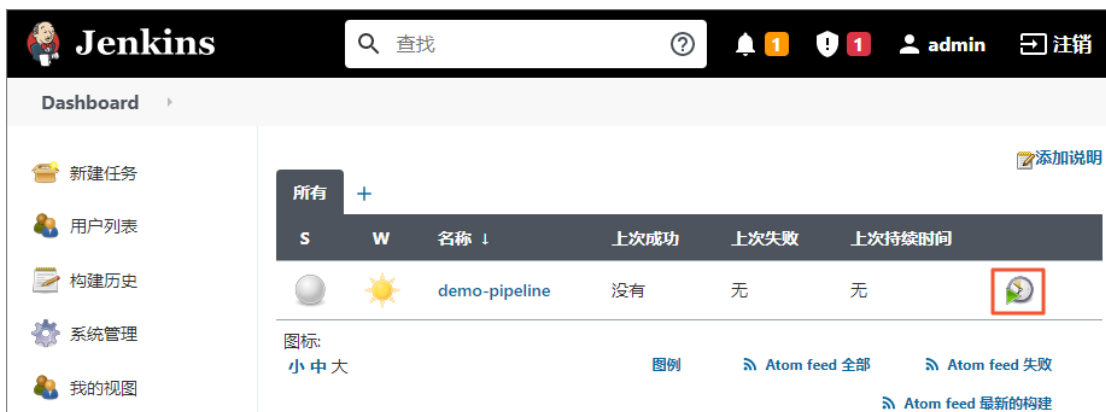
i. 配置Kubernetes地址, 凭据选择为ask-jenkins-token, 单击连接测试验证连接是否正常。

ii. 配置Jenkins地址和Jenkins通道。

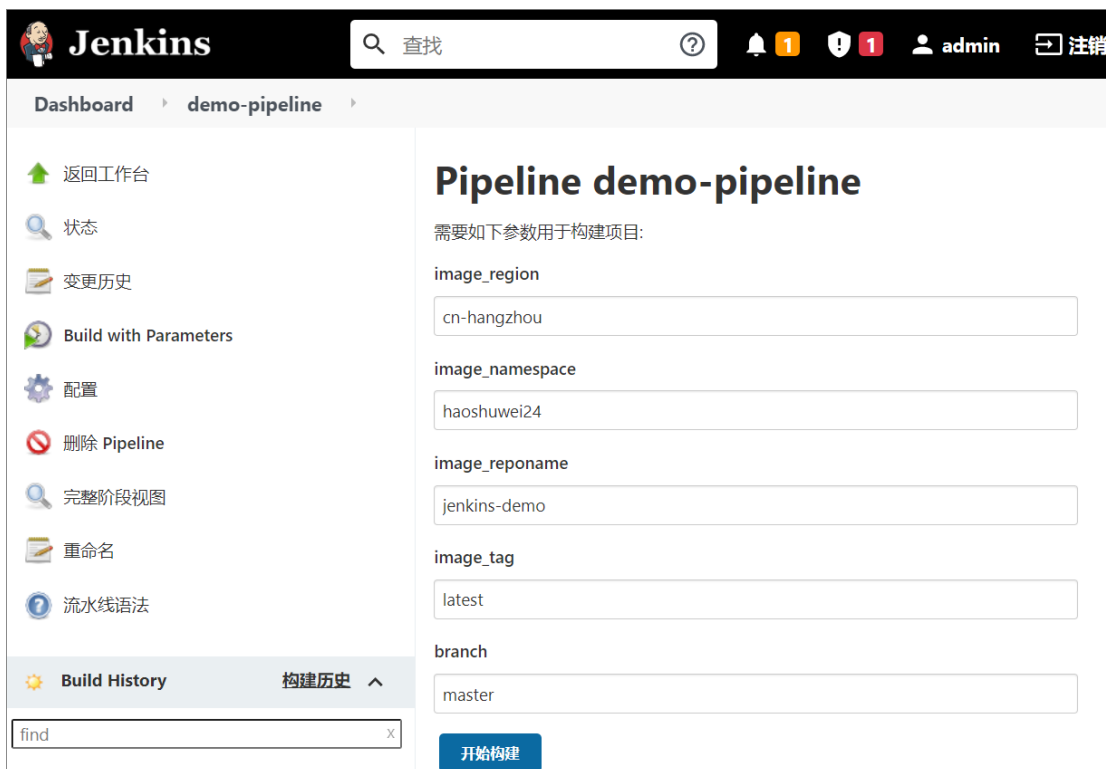
iii. 单击Save。

9. 构建demo-pipeline并访问应用服务。

i. 在Jenkins首页, 单击demo-pipeline的图标。



- ii. 根据您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/ack-cicd/ack-jenkins-demo:latest`。

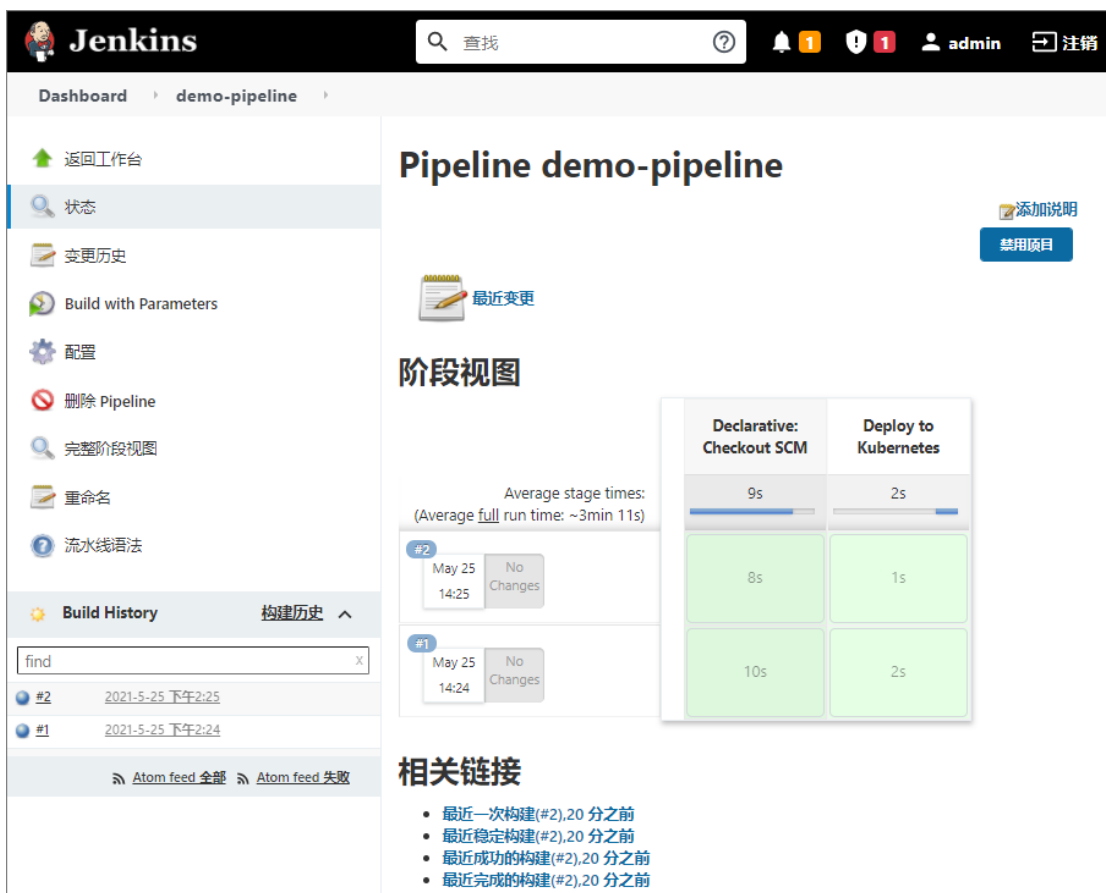


- iii. 单击开始构建。

测试Kubernetes集群动态分配的Jenkins Slave Pod与Jenkins Master是否连接正常。

执行构建后，Jenkins从Kubernetes集群动态创建一个Slave Pod运行本次构建任务。关于示例应用代码，请参见[jenkins-demo-GitHub](#)或[jenkins-demo](#)。

iv. 查看状态，若构建成功则表示Jenkins on Kubernetes运行正常。



后续步骤

- 关于如何为Slave Pod配置Maven缓存，请参见[为Slave Pod配置Maven缓存](#)。
- 关于如何使用kaniko构建和推送容器镜像，请参见[使用kaniko构建和推送容器镜像](#)。

9. 搭建WordPress应用

本文介绍如何使用Cloud Shell来快速搭建基于阿里云Serverless Kubernetes（ASK）和弹性容器实例（ECI）的WordPress应用。

背景信息

WordPress是使用PHP语言开发的博客平台，在支持PHP和MySQL数据库的服务器上，您可以用WordPress架设自己的网站，也可以用作内容管理系统（CMS）。

Clos

本教程已在Cloud Shell中集成，您也可以打开[Cloud Shell](#),

创建ASK集群

1. 打开[Cloud Shell](#)。
2. 输入以下命令切换地域。

```
switch-region cn-beijing
```

3. （可选）修改集群配置文件。
4. 执行以下命令创建ASK集群。
5. 根据需要执行以下命令查看集群信息。

- 查看集群的属性

```
aliyun cs GET /clusters/<YOUR-CLUSTER-ID>
```

- 查看集群的配置信息

安装WordPress

访问WordPress

打开Cloud Shell

说明

以下步骤均已在Cloud Shell中集成，可以通过打开上面的链接来快速体验通过Cloud Shell操作ECI。

创建Serverless Kubernetes集群

请参见[创建Serverless Kubernetes集群](#)。

安装WordPress

注意

请确保上一步中创建的 Serverless Kubernetes 集群，已完成初始化（一般需要3~5分钟），再开始以下的操作。

使用Cloud Shell来管理上一步中创建中的Serverless Kubernetes集群。

```
source use-k8s-cluster ${集群ID}
```

执行WordPress安装YAML文件。

```
kubectl apply -f wordpress-all-in-one-pod.yaml
```

观察安装进度，直到STATUS为Running。

```
kubectl get pods
```

查询EIP地址。

```
kubectl get -o json pod wordpress |grep "k8s.aliyun.com/allocated-eipAddress"
```

由于安全组默认没有开放 80 端口的访问，需要给安全组添加 80 端口的 ACL。

首先获取Serverless Kubernetes集群自动创建的安全组，找到最近创建的以 `alicloud-cs-auto-created` 为命名前缀的安全组ID。

```
aliyun ecs DescribeSecurityGroups|grep -B 1 'alicloud-cs-auto-created'|head -1
```

对安全组进行授权操作。

```
aliyun ecs AuthorizeSecurityGroup --RegionId cn-chengdu --SecurityGroupId ${安全组ID} --IpProtocol tcp --PortRange 80/80 --SourceCidrIp 0.0.0.0/0 --Priority 100
```

使用WordPress

在浏览器起输入上一步获取到的EIP地址，即可开始使用WordPress。

10. 搭建Spark应用

本文介绍如何使用阿里云Serverless Kubernetes（ASK）和弹性容器实例（ECI），快速搭建Spark应用。

背景信息

Apache Spark是一个在数据分析领域广泛使用的开源项目，它常被应用于众所周知的大数据和机器学习工作负载中。从Apache Spark 2.3.0版本开始，您可以在Kubernetes上运行和管理Spark资源。

Spark Operator是专门针对Spark on Kubernetes设计的Operator，开发者可以通过使用CRD的方式，提交Spark任务到Kubernetes集群中。使用Spark Operator有以下优势：

- 能够弥补原生Spark对Kubernetes支持不足的部分。
- 能够快速和Kubernetes生态中的存储、监控、日志等组件对接。
- 支持故障恢复、弹性伸缩、调度优化等高阶Kubernetes特性。

准备工作

1. 创建ASK集群。

在[容器服务管理控制台](#)上创建ASK集群。更多信息，请参见[创建Serverless Kubernetes集群](#)。

说明

如果您需要从公网拉取镜像，或者训练任务需要访问公网，请配置NAT网关。

您可以通过kubectl管理和访问ASK集群，相关操作如下：

- 如果您需要从本地计算机管理集群，请安装并配置kubectl客户端。具体操作，请参见[通过kubectl连接Kubernetes集群](#)。
- 您也可以直接在CloudShell上通过kubectl管理集群。具体操作，请参见[在CloudShell上通过kubectl管理Kubernetes集群](#)。

2. 创建OSS存储空间。

您需要创建一个OSS存储空间（Bucket）用来存放测试数据、测试结果和测试过程中的日志等。关于如何创建OSS Bucket，请参见[创建存储空间](#)。

安装Spark Operator

1. 安装Spark Operator。

- i. 在[容器服务管理控制台](#)的左侧导航栏，选择市场>应用市场。
- ii. 在应用目录页签，找到并单击ack-spark-operator。
- iii. 单击创建，在弹出面板中完成相关配置。

参数配置中，`sparkJobNamespace` 为需要部署Spark作业的命名空间，默认为 `default`，如果设置为空字符串则表示所有命名空间都监听。

2. 创建ServiceAccount、Role和Rolebinding。

Spark作业需要一个ServiceAccount来获取创建Pod的权限，因此需要创建ServiceAccount、Role和Rolebinding。YAML示例如下，请根据需要修改三者的Namespace。

```

apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: default
  name: spark-role
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["*"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
subjects:
- kind: ServiceAccount
  name: spark
  namespace: default
roleRef:
  kind: Role
  name: spark-role
  apiGroup: rbac.authorization.k8s.io

```

构建Spark作业镜像

您需要编译Spark作业的JAR包，使用Dockerfile打包镜像。

以阿里云容器服务的Spark基础镜像为例，设置Dockerfile内容如下：

```

FROM registry.aliyuncs.com/acs/spark:ack-2.4.5-latest
RUN mkdir -p /opt/spark/jars
# 如果需要使用OSS（读取OSS数据或者离线Event到OSS），可以添加以下JAR包到镜像中
ADD https://repo1.maven.org/maven2/com/aliyun/odps/hadoop-fs-oss/3.3.8-public/hadoop-fs-oss-3.3.8-public.jar $SPARK_HOME/jars
ADD https://repo1.maven.org/maven2/com/aliyun/oss/aliyun-sdk-oss/3.8.1/aliyun-sdk-oss-3.8.1.jar $SPARK_HOME/jars
ADD https://repo1.maven.org/maven2/org/aspectj/aspectjweaver/1.9.5/aspectjweaver-1.9.5.jar $SPARK_HOME/jars
ADD https://repo1.maven.org/maven2/org/jdom/jdom/1.1.3/jdom-1.1.3.jar $SPARK_HOME/jars
COPY SparkExampleScala-assembly-0.1.jar /opt/spark/jars

```

建议您使用阿里云Spark基础镜像。阿里云提供了Spark2.4.5的基础镜像，针对Kubernetes场景（调度、弹性）进行了优化，能够极大提升调度速度和启动速度。您可以通过设置Helm Chart的变量

`enableAlibabaCloudFeatureGates: true` 的方式开启，如果想要达到更快的启动速度，可以设置

`enableWebhook: false` 。

```
1 operatorImageName: registry.aliyuncs.com/acs/spark-operator
2 operatorImageVersion: ack-2.4.5-latest
3 operatorVersion: v2.4.5-v1beta2
4 imagePullPolicy: IfNotPresent
5
6 rbac:
7   create: true
8
9 serviceAccounts:
10  spark:
11    create: true
12    name: spark
13  sparkoperator:
14    create: true
15    name: ack-spark-operator
16
17 sparkJobNamespace: "default"
18
19 enableWebhook: false
20 enableMetrics: true
21 enableAlibabaCloudFeatureGates: false
22
```

构建ImageCache

Spark镜像如果较大，则拉取需要较长时间，您可以通过ImageCache加速镜像拉取。更多信息，请参见[管理ImageCache](#)和[使用ImageCache加速创建Pod](#)。

编写作业模板并提交作业

创建一个Spark作业的YAML配置文件，并进行部署。

1. 创建spark-pi.yaml文件。

一个典型的作业模板示例如下。更多信息，请参见[spark-on-k8s-operator](#)。

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: default
spec:
  type: Scala
  mode: cluster
  image: "registry.aliyuncs.com/acs/spark:ack-2.4.5-latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///opt/spark/examples/jars/spark-examples_2.11-2.4.5.jar"
  sparkVersion: "2.4.5"
  restartPolicy:
    type: Never
  driver:
    cores: 2
    coreLimit: "2"
    memory: "3g"
    memoryOverhead: "1g"
    labels:
      version: 2.4.5
    serviceAccount: spark
    annotations:
      k8s.aliyun.com/eci-kube-proxy-enabled: 'true'
      k8s.aliyun.com/eci-image-cache: "true"
    tolerations:
      - key: "virtual-kubelet.io/provider"
        operator: "Exists"
  executor:
    cores: 2
    instances: 1
    memory: "3g"
    memoryOverhead: "1g"
    labels:
      version: 2.4.5
    annotations:
      k8s.aliyun.com/eci-kube-proxy-enabled: 'true'
      k8s.aliyun.com/eci-image-cache: "true"
    tolerations:
      - key: "virtual-kubelet.io/provider"
        operator: "Exists"
```

2. 部署一个Spark计算任务。

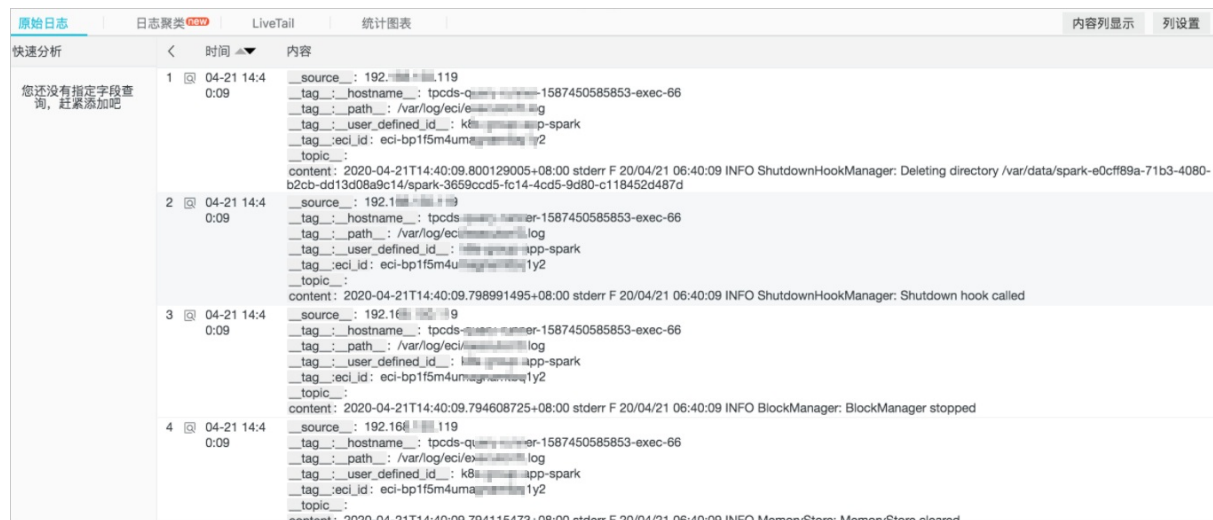
```
kubectl apply -f spark-pi.yaml
```

配置日志采集

以采集Spark的标准输出日志为例，您可以在Spark driver和Spark executor的envVars字段中注入环境变量，实现日志的自动采集。更多信息，请参见[自定义配置ECI日志采集](#)。


```
envVars:
  aliyun_logs_test-stdout_project: test-k8s-spark
  aliyun_logs_test-stdout_machinegroup: k8s-group-app-spark
  aliyun_logs_test-stdout: stdout
```

提交作业时，您可以按上述方式设置driver和executor的环境变量，即可实现日志的自动采集。



配置历史服务器

历史服务器用于审计Spark作业，您可以通过在Spark Application的CRD中增加SparkConf字段的方式，将event写入到OSS，再通过历史服务器读取OSS的方式进行展现。配置示例如下：

```
sparkConf:
  "spark.eventLog.enabled": "true"
  "spark.eventLog.dir": "oss://bigdatastore/spark-events"
  "spark.hadoop.fs.oss.impl": "org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem"
  # oss bucket endpoint such as oss-cn-beijing.aliyuncs.com
  "spark.hadoop.fs.oss.endpoint": "oss-cn-beijing.aliyuncs.com"
  "spark.hadoop.fs.oss.accessKeySecret": ""
  "spark.hadoop.fs.oss.accessKeyId": ""
```

阿里云也提供了spark-history-server的Chart，您可以在容器服务管理控制台的[市场>应用市场](#)页面，搜索ack-spark-history-server进行安装。安装时需在参数中配置OSS的相关信息，示例如下：

```
oss:
  enableOSS: true
  # Please input your accessKeyId
  alibabaCloudAccessKeyId: ""
  # Please input your accessKeySecret
  alibabaCloudAccessKeySecret: ""
  # oss bucket endpoint such as oss-cn-beijing.aliyuncs.com
  alibabaCloudOSSEndpoint: "oss-cn-beijing.aliyuncs.com"
  # oss file path such as oss://bucket-name/path
  eventsDir: "oss://bigdatastore/spark-events"
```

安装完成后，您可以在集群详情页面的服务中看到ack-spark-history-server的对外地址，访问对外地址即可查看历史任务归档。

The screenshot shows the Spark History Server interface. At the top, it says 'History Server' and 'Event log directory: oss://bigdatastore/spark-events'. Below that, it indicates 'There are 18 event log(s) currently being processed which may result in additional applications getting listed on this page. Refresh the page to view updates.' and 'Client local time zone: Asia/Shanghai'. There is a search bar and a 'Show 20 entries' dropdown. The main part of the image is a table with columns: App ID, App Name, Started, Completed, Duration, Spark User, Last Updated, and Event Log. Each row represents a Spark application with its ID, name (Spark Pi), start and end times, duration (e.g., 17 min), user (root), and last updated time. Each row also has a 'Download' button for the event log.

查看作业结果

1. 查看Pod的执行情况。

```
kubectl get pods
```

预期返回结果：

NAME	READY	STATUS	RESTARTS	AGE
spark-pi-1547981232122-driver	1/1	Running	0	12s
spark-pi-1547981232122-exec-1	1/1	Running	0	3s

2. 查看实时Spark UI。

```
kubectl port-forward spark-pi-1547981232122-driver 4040:4040
```

3. 查看Spark Application的状态。

```
kubectl describe sparkapplication spark-pi
```

预期返回结果：

```
Name: spark-pi
Namespace: default
Labels: <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
  {"apiVersion":"sparkoperator.k8s.io/v1alpha1","kind":"SparkApplication",
  "metadata":{"annotations":{},"name":"spark-pi","namespace":"defaul...
API Version: sparkoperator.k8s.io/v1alpha1
Kind: SparkApplication
Metadata:
  Creation Timestamp: 2019-01-20T10:47:08Z
  Generation: 1
  Resource Version: 4923532
  Self Link: /apis/sparkoperator.k8s.io/v1alpha1/namespaces/default/sparkappl
  ications/spark-pi
  UID: bbe7445c-1ca0-11e9-9ad4-062fd7c19a7b
```

```

spec:
  Deps:
  Driver:
    Core Limit: 200m
    Cores: 0.1
    Labels:
      Version: 2.4.0
    Memory: 512m
    Service Account: spark
    Volume Mounts:
      Mount Path: /tmp
      Name: test-volume
  Executor:
    Cores: 1
    Instances: 1
    Labels:
      Version: 2.4.0
    Memory: 512m
    Volume Mounts:
      Mount Path: /tmp
      Name: test-volume
  Image: gcr.io/spark-operator/spark:v2.4.0
  Image Pull Policy: Always
  Main Application File: local:///opt/spark/examples/jars/spark-examples_2.11-2.4.0.jar
  Main Class: org.apache.spark.examples.SparkPi
  Mode: cluster
  Restart Policy:
    Type: Never
  Type: Scala
  Volumes:
    Host Path:
      Path: /tmp
      Type: Directory
    Name: test-volume
  Status:
    Application State:
      Error Message:
      State: COMPLETED
    Driver Info:
      Pod Name: spark-pi-driver
      Web UI Port: 31182
      Web UI Service Name: spark-pi-ui-svc
    Execution Attempts: 1
    Executor State:
      Spark - Pi - 1547981232122 - Exec - 1: COMPLETED
    Last Submission Attempt Time: 2019-01-20T10:47:14Z
    Spark Application Id: spark-application-1547981285779
    Submission Attempts: 1
    Termination Time: 2019-01-20T10:48:56Z
  Events:
    Type Reason Age From Message
    ---- -
    Normal SparkApplicationAdded 55m spark-operator SparkApplicati
    on spark-pi was added. Enqueuing it for submission

```

```

On spark-pi was added, Enqueuing it for submission
Normal SparkApplicationSubmitted 55m spark-operator SparkApplicati
on spark-pi was submitted successfully
Normal SparkDriverPending 55m (x2 over 55m) spark-operator Driver spark-p
i-driver is pending
Normal SparkExecutorPending 54m (x3 over 54m) spark-operator Executor spark
-pi-1547981232122-exec-1 is pending
Normal SparkExecutorRunning 53m (x4 over 54m) spark-operator Executor spark
-pi-1547981232122-exec-1 is running
Normal SparkDriverRunning 53m (x12 over 55m) spark-operator Driver spark-p
i-driver is running
Normal SparkExecutorCompleted 53m (x2 over 53m) spark-operator Executor spark
-pi-1547981232122-exec-1 completed

```

4. 查看日志获取结果。

NAME	READY	STATUS	RESTARTS	AGE
spark-pi-1547981232122-driver	0/1	Completed	0	1m

当Spark Applicaiton的状态为Succeed或者Spark driver对应的Pod状态为Completed时，可以查看日志获取结果。

```

kubectl logs spark-pi-1547981232122-driver
Pi is roughly 3.152155760778804

```

11. 搭建TensorFlow应用

本文介绍如何使用阿里云Serverless Kubernetes（ASK）和弹性容器实例（ECI），快速完成基于GPU的TensorFlow训练任务。

背景信息

人工智能与机器学习已经被广泛应用到各个领域，近些年来各种各样的训练模型被提出，更多的训练任务运行到云上。然而上云之后，想要轻松、持久地运行训练任务，仍有一些痛点，例如：

- 环境搭建麻烦：您需要购买GPU实例并安装GPU驱动，即使您已经把训练任务容器化，还需要安装GPU runtime hook。
- 使用缺乏弹性：运行完任务后，为了节约成本，您可能需要释放闲置资源，但在下次启动任务时您需要重新创建实例并配置环境；或者在计算节点资源不够的情况下，扩容需要您再次创建实例并配置环境。

针对上述痛点，推荐您使用ASK+ECI的方案来运行训练任务，可以帮助您轻松地创建和启动训练任务。该方案还具备以下优势：

- 按需付费，免运维。
- 一次配置，无限次复用。
- 镜像缓存功能加速Pod创建，训练任务启动快速。
- 数据与训练模型解耦，数据可以持久化存储。

准备工作

1. 准备好训练模型的容器镜像和训练数据。

本文以Github的一个TensorFlow训练任务为例，相关示例镜像（eci/tensorflow）您可以从阿里云容器镜像仓库中获取。更多信息，请参见[TensorFlow训练任务](#)。

2. 创建ASK集群。

在[容器服务管理控制台](#)上创建ASK集群。更多信息，请参见[创建Serverless Kubernetes集群](#)。

🔍 说明

如果您需要从公网拉取镜像，或者训练任务需要访问公网，请配置NAT网关。

您可以通过kubect管理并访问ASK集群，相关操作如下：

- 如果您需要从本地计算机管理集群，请安装并配置kubect客户端。具体操作，请参见[通过kubect连接Kubernetes集群](#)。
- 您也可以直接在CloudShell上通过kubect管理集群。具体操作，请参见[在CloudShell上通过kubect管理Kubernetes集群](#)。

3. 创建NAS文件系统，并添加挂载点。

在[NAS文件系统控制台](#)上创建文件系统，并添加挂载点。更多信息，请参见[管理文件系统](#)和[管理挂载点](#)。

🔍 说明

NAS文件系统需和ASK集群处于同一VPC。

创建镜像缓存

镜像缓存功能已经集成到Kubernetes CRD中，可以加速镜像的拉取。更多信息，请参见[使用镜像缓存CRD加速创建Pod](#)。

操作如下：

1. 准备YAML文件。

示例imagecache.yaml的内容如下：

```
apiVersion: eci.alibabacloud.com/v1
kind: ImageCache
metadata:
  name: tensorflow
spec:
  images:
  - registry-vpc.cn-beijing.aliyuncs.com/eci/tensorflow:1.0 # 训练任务的镜像，建议您上传到
    阿里云容器镜像仓库中。此处使用VPC私网地址，请确保对应VPC为集群所属的VPC。
```

2. 创建镜像缓存。

```
kubectl create -f imagecache.yaml
```

创建镜像缓存时需要拉取镜像，受镜像大小影响，需要一定的时间。您可以通过以下命令查询镜像缓存的创建进度。

```
Kubectl get imagecache tensorflow
```

返回如下结果时，表示镜像缓存已经创建成功。

NAME	AGE	CACHEID	PHASE	PROGRESS
tensorflow	11m	imc-2ze1xczztv7tgesg****	Ready	100%

创建训练任务

您可以使用镜像缓存来创建训练任务。

1. 准备YAML文件。

示例gpu_pod.yaml的内容如下：

```

apiVersion: v1
kind: Pod
metadata:
  name: tensorflow
  annotations:
    k8s.aliyun.com/eci-use-specs: "ecs.gn6i-c4g1.xlarge" # 指定GPU规格创建ECI实例
    k8s.aliyun.com/eci-image-cache: "true" # 开启镜像缓存自动匹配
spec:
  containers:
  - name: tensorflow
    image: registry-vpc.cn-beijing.aliyuncs.com/eci/tensorflow:1.0 # 训练任务的镜像
    command:
      - "sh"
      - "-c"
      - "python models/tutorials/image/imagenet/classify_image.py" # 触发训练任务的脚本
  resources:
    limits:
      nvidia.com/gpu: "1" # 容器所需的GPU个数
    volumeMounts:
      - name: nfs-pv
        mountPath: /tmp/imagenet
  volumes:
  - name: nfs-pv #训练结果持久化，保存到NAS
    flexVolume:
      driver: alicloud/nas
      fsType: nfs
      options:
        server: 16cde4****-ijv**.cn-beijing.nas.aliyuncs.com #NAS文件系统挂载点
        path: / #挂载目录
    restartPolicy: OnFailure

```

2. 执行命令创建Pod。

```
kubectl create -f gpu_pod.yaml
```

3. 查看执行情况。

您可以根据需要查看事件或日志。

- o 查看事件

```
kubectl describe pod tensorflow
```

- o 查看日志

```
kubectl logs tensorflow
```

查看结果

您可以在控制台上查看训练任务的运行结果。

- 在NAS文件系统控制台，您可以看到训练完成的数据已占用存储容量。

文件系统 ID/名称	标签	文件系统类型	存储规格	使用量/总容量	可用区	计费方式	数据生命周期管理	创建时间	状态	协议类型	操作
+ 16cde4-16cde4		通用型NAS	容量型	176.84 MiB / 10 PiB	华北2 可用区 H	按量付费	已启用 配置策略	2021年2月2日 17:55:39	✓ 运行中	NFS	监控 管理 更多

? 说明

训练结果存储在配置的NAS文件系统中，挂载NAS后，您可以在对应的路径下获取结果数据。

- 在弹性容器实例控制台，您可以看到运行成功的ECI实例。

容器组ID/名称	状态	事件	规格	所在可用区	IP地址	时间	安全组/虚拟交换机	操作
eci-2zeblhp0f.../n4dmd	运行成功	5	4 vCpu 15 GiB	华北2 可用区 H	192...2 (内)	实例创建: 2021年2月3日 18:05:55 执行完成: 2021年2月3日 18:07:11	sg-2ze5f3...hghgr1p vsw-2zer4...j66bdn5wu	删除 重启 修改

? 说明

运行成功表示实例中的容器已经运行终止，此时系统会回收底层计算资源，不再对Pod进行计费。

12.通过ASK对接RDS

本文将介绍如何通过ECI对接RDS。

目前RDS支持五种数据库引擎，您可以根据业务需求来选择。更多信息，请参见[数据库引擎介绍](#)。

本文使用RDS MySQL实例为例。

前提条件

- 已创建RDS实例，并且在实例上配置数据库用户账号。具体操作，请参见[创建RDS MySQL实例](#)。
- Kubernetes集群中已正确部署virtual-kubelet。（Serverless Kubernetes默认已经安装，关于如何创建ASK集群，请参见[创建Serverless Kubernetes集群](#)）

注意

建议RDS实例与Kubernetes集群属于同一VPC网络下，当然也可以跨VPC网络通过公网方式建立连接；本次示例使用同一VPC网络下。

设置白名单

说明

RDS的白名单包括两种类型，IP白名单和安全组，如下：

- IP白名单

添加IP地址，允许这些IP地址访问该RDS实例。默认的IP白名单只包含默认IP地址127.0.0.1，表示任何设备均无法访问该RDS实例。

- 安全组

安全组是一种虚拟防火墙，用于控制安全组中的ECI实例的出入流量。在RDS白名单中添加安全组后，该安全组中的ECI实例就可以访问RDS实例。

- 内网方式

1、在RDS管理控制台中单击数据库连接获取内网地址。



内网地址为数据库的host。

2、设置白名单

添加创建Pod时绑定的vpc实例或者vsw实例的网段，也可以直接添加安全组实例。



● 公网方式

1、申请外网地址

公网也叫外网，通过公网访问RDS就是使用RDS实例的外网地址进行访问。RDS实例默认不提供外网地址，如果需要通过公网访问，请在RDS控制台>数据库连接页面申请外网地址。



说明

- 外网地址会降低实例的安全性，请谨慎使用。
- 为了获得更快的传输速率和更高的安全性，建议您将应用迁移到与您的RDS实例在同一地域且网络类型相同的ECI实例，然后使用内网地址。您也可以使用公网访问，更多信息，请参见[连接RDS数据库](#)。

有了外网地址之后，就可以使用外网地址连接到RDS实例。

2、设置白名单

- ECI实例所属vsw绑定到NAT网关，获取它的公网IP地址添加到RDS白名单中。
- 添加ECI绑定EIP实例的弹性公网地址。

说明

- 当RDS开通外网地址访问，ECI实例若连接RDS需要具备外网访问能力，具体可通过以下两种方式获取。
 - 方式一：VPC绑定NAT网关+EIP；把ECI所绑定的VSW实例公网IP地址添加到白名单中（建议使用）。
 - 方式二：实例直接绑定EIP；直接添加ECI绑定的弹性公网IP地址到白名单中。
- 白名单可以让RDS实例得到高级别的访问安全保护，建议您定期维护白名单。设置白名单不会影响RDS实例的正常运行。具体操作，请参见[设置IP白名单](#)。

Kubernetes方式

以下使用内网方式连接RDS数据库。

创建Pod所使用的vsw实例网段添加到RDS白名单中。

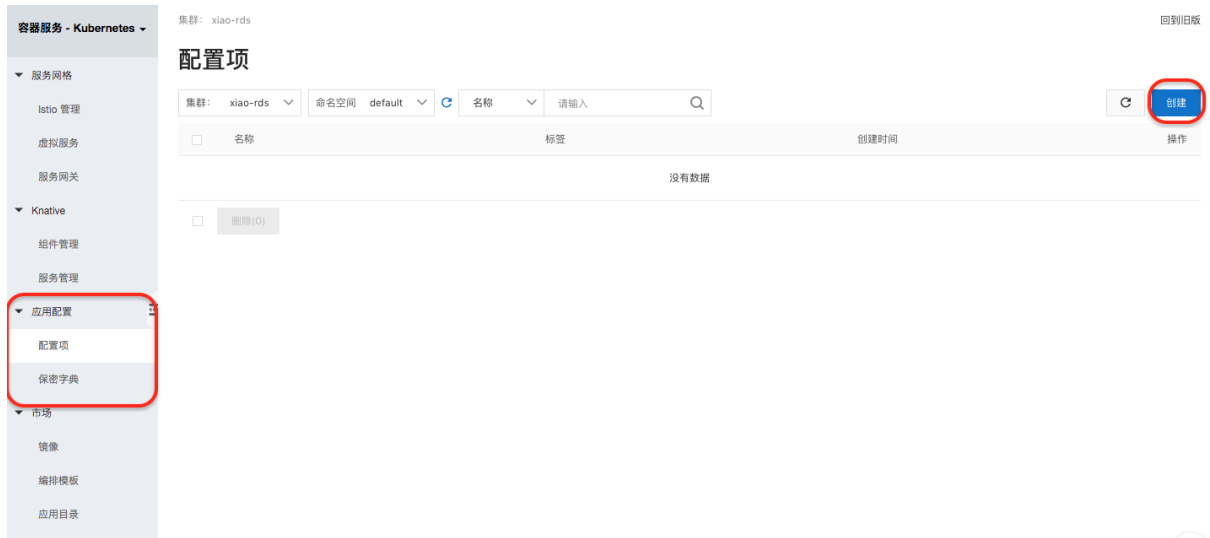
The screenshot shows the AWS Management Console interface. The top part displays a list of VSW (Virtual Switch) instances under the '交换机' (Switches) section. The table below shows details for several instances, with the IP address '192.168.64.0/24' highlighted in a red box.

实例ID/名称	专有网络	标签	状态	IPv4网段	可用IP数	默认交换机	可用区	资源	操作
vsw-bp1e2dz...	eci-test-hangzhou		可用	192.168.4.0/22	1017	否	杭州 可用区B	默认	管理 删除 创建
vsw-bp1sx40...	eci-test-hangzhou		可用	192.168.18.0/24	246	否	杭州 可用区I	默认	管理 删除 创建
vsw-bp1xsf...	eci-test-hangzhou		可用	192.168.0.0/24	252	否	杭州 可用区E	默认	管理 删除 创建
vsw-bp1...	eci-test-hangzhou		可用	192.168.64.0/24	238	否	杭州 可用区H	默认	管理 删除 创建
vsw-bp...	eci-test-hz-g		可用	192.168.29.0/24	237	否	杭州 可用区G	默认	管理 删除 创建

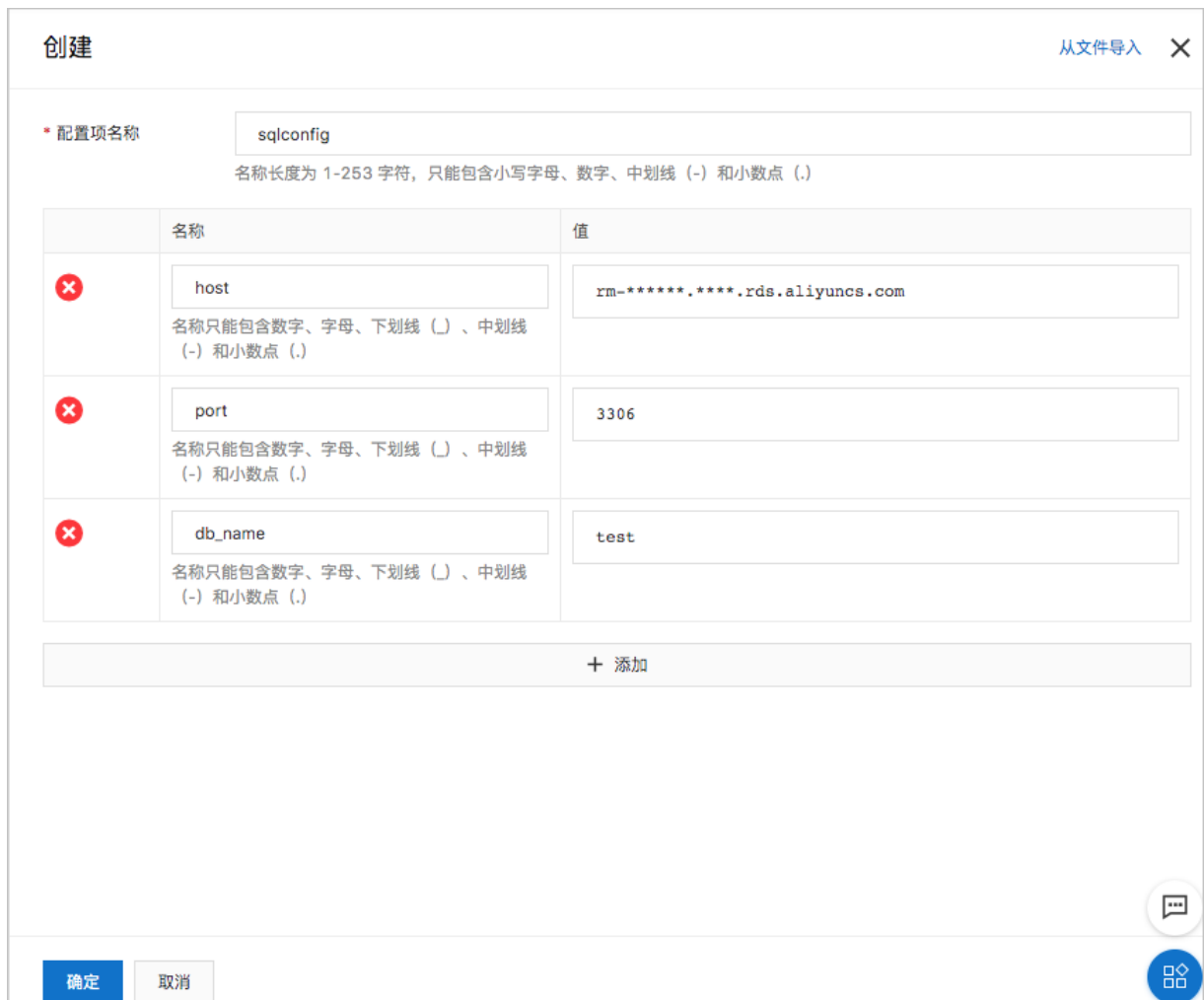
The bottom part of the screenshot shows the '数据安全' (Data Security) settings for a database instance, specifically the '白名单设置' (Whitelist Settings) section. The network isolation mode is set to '通用白名单模式' (General Whitelist Mode). A list of whitelisted IP addresses is shown, with '192.168.64.0/24' added. A red box highlights this IP address in the screenshot.

1、创建ConfigMap

ConfigMap 将您的环境配置信息和容器镜像解耦，便于应用配置的修改，在容器服务管理控制台中单击应用配置>配置项创建。



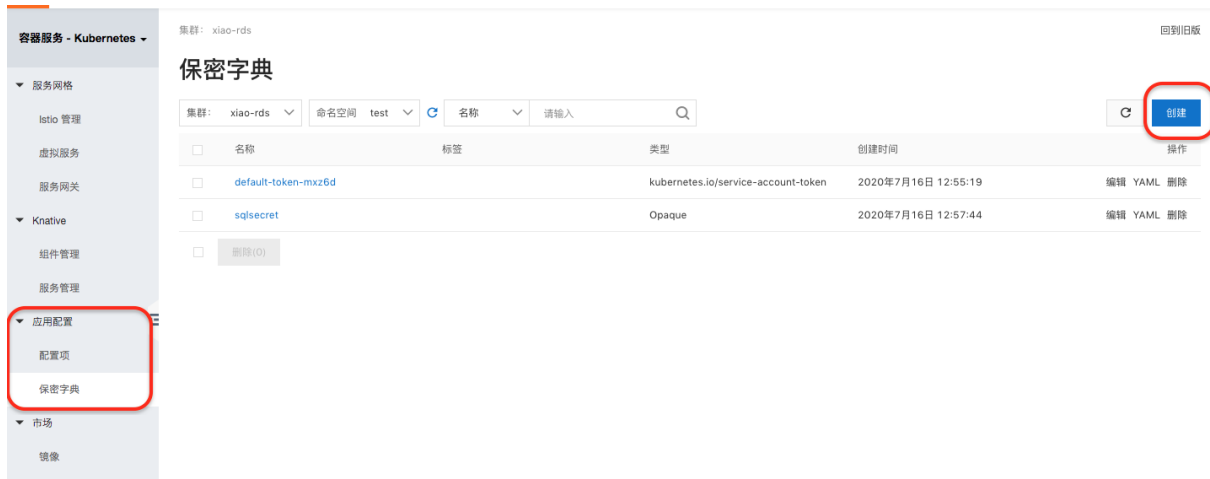
连接云数据库需要的信息写入configMap中。



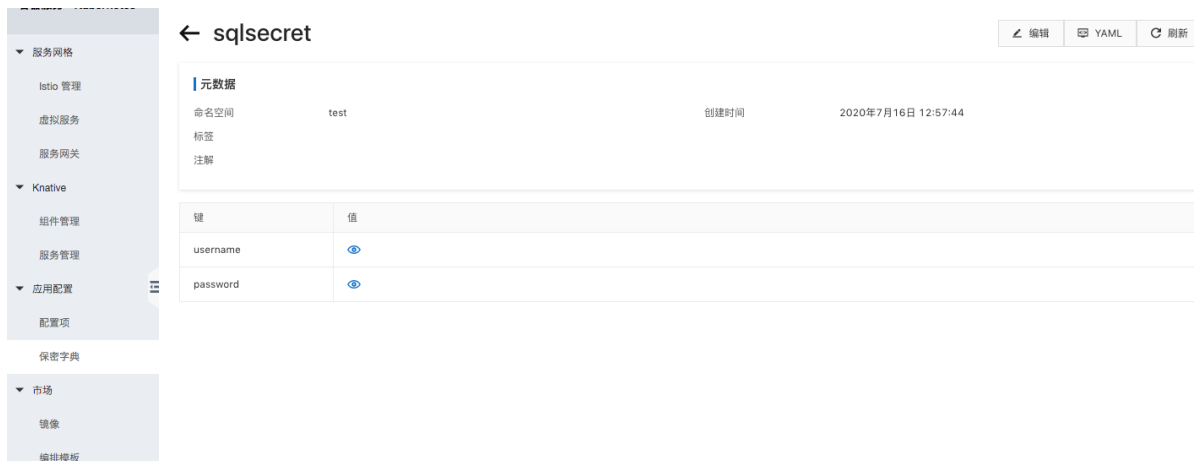


2、创建Secret

当您需要储存机密信息时可以使用 Secret 将配置详细信息传递给应用的安全方式，例如您的数据库名称、用户和密码等详细信息，您可以将其作为环境变量注入到应用中。



添加RDS数据库用户和密码。



3、创建Pod对接RDS

在容器服务管理控制台单击应用>无状态使用模板创建Pod。

🔍 说明

由于RDS与原生的数据库服务完全兼容，所以您可以使用任何通用的数据库客户端连接到RDS实例，且连接方法类似。如下示例使用python连接数据库。

```

apiVersion: v1
kind: Pod
metadata:
  labels:
    name: rds-test
  name: rds-test
spec:
  containers:
    - name: test-rds
      image: registry.cn-hangzhou.aliyuncs.com/eci_open/sqlclient:latest # 一个python job用来
      连接云数据库的镜像
      imagePullPolicy: IfNotPresent
      command: [ "/bin/bash", "-c", "python3 /testapp/mysqlclient.py" ]
      env:
        - name: MYSQL_HOST # 请注意这里和 ConfigMap 中的键名是不一样的
          valueFrom:
            configMapKeyRef:
              name: sqlconfig # 这个值来自 ConfigMap
              key: host # 需要取值的键
        - name: MYSQL_PORT
          valueFrom:
            configMapKeyRef:
              name: sqlconfig
              key: port
        - name: MYSQL_DB
          valueFrom:
            configMapKeyRef:
              name: sqlconfig
              key: database
        - name: MYSQL_USERNAME
          valueFrom:
            secretKeyRef:
              name: sqlsecret
              key: username
        - name: MYSQL_PWD
          valueFrom:
            secretKeyRef:
              name: sqlsecret
              key: password
      restartPolicy: Never

```

本示例镜像使用一个python脚本连接云数据库并插入两条数据。如下：

```
import pymysql
import os

config = {
    'host': str(os.getenv('MYSQL_HOST')),
    'port': int(os.getenv('MYSQL_PORT')),
    'user': str(os.getenv('MYSQL_USERNAME')),
    'password': str(os.getenv('MYSQL_PWD')),
    'database': str(os.getenv('MYSQL_DB')),
}

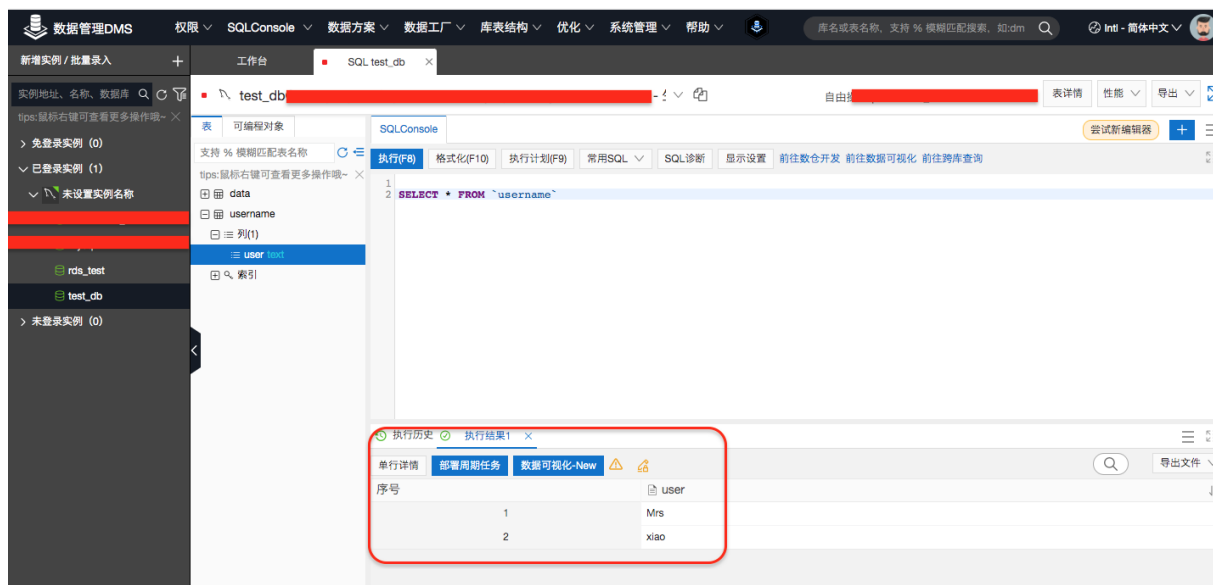
def mysqlClient():
    db = pymysql.connect(**config)
    try:
        cursor = db.cursor()
        cursor.execute("INSERT INTO username(user) VALUES('Mrs')")
        cursor.execute("INSERT INTO username(user) VALUES('xiao')")
        cursor.execute("SELECT*FROM username")
        result = cursor.fetchall()
    except Exception as e:
        print('System Error: ',e)
    finally:
        db.commit()
        cursor.close()
        db.close()
    return result

if __name__ == '__main__':
    mysqlClient()
```

🔍 说明

连接RDS数据库的方式有公网访问和内网访问两种，以上所用方式为内网访问，建议使用内网访问的方式保证传输速率和安全性。

4、成功对接RDS数据库效果



常见问题

数据库无法连接的排查方法请根据当前环境的实际情况，选择对应的排查方法。

- 内网无法访问RDS实例
- 外网无法访问RDS实例

更多信息，请参见[解决无法连接RDS实例的问题](#)。

数据库无法连接的常见场景包括：

- 网络类型不同
- 专有网络不同
- 域名解析失败或错误
- 地域不同
- IP白名单设置有误
- 只读实例未设置白名单
- 内外网地址使用错误
- 连接数已满
- 用户名或密码错误
- 无法解析地址

13. 常用数据源对接

13.1. 在ECI中访问HDFS的数据

数据准备

HDFS是Hadoop/Spark批处理作业可选的数据存储之一。本文将演示在HDFS中创建一个文件，并在Spark中进行访问。

1. 开通HDFS服务，并创建文件系统

创建文件系统



区域

- 区域

cn-hangzhou

- 可用区

cn-hangzhou-e

同一地域不同可用区之间文件系统与计算节点互通。若要获取最高性能，请使用同一可用区内的文件系统与计算节点。

基本配置

- 文件系统名称

spark-test

6-64个字符,只允许英文字母,数字,或'_'、'-'

文件系统描述

测试

2/32

描述备注信息,长度不超过32个字符

存储配置

- 协议类型

HDFS

确定

取消



2. 设置权限组

1. 创建权限组。

创建权限组



● 区域

cn-hangzhou

● 名称

spark-on-k8s|

长度为6-64个字符,允许英文字母、数字,或'_','-'

网络类型

VPC

描述

0/32

描述备注信息,长度不超过32个字符

确定

取消



2. 设置权限组的规则。

创建规则

授权地址 <small>↓↑ ?</small>	访问类型 <small>↓↑ ?</small>	优先级 <small>↓↑ ?</small>	操作
0.0.0.0/0	可读写	100	编辑 删除

3. 为挂载点添加权限组。

修改挂载点权限组



文件系统ID

4b1fcae5-36fd-43fb-a92f-3bd8f38b2c7e

挂载点类型

VPC

权限组

spark-on-k8s ∨



至此HDFS文件系统就准备完毕。

3. 安装Apache Hadoop Client

HDFS文件系统准备就绪后，就是存入文件。我们采用HDFS client的方式。

Apache Hadoop下载地址：[官方链接](#)。建议选用的Apache Hadoop版本不低于2.7.2，本文中使用的Apache Hadoop版本为Apache Hadoop 2.7.2。

1、执行如下命令解压Apache Hadoop压缩包到指定文件夹。


```
tar -zxvf hadoop-2.7.2.tar.gz -C /usr/local/
```

2、执行如下命令打开core-site.xml配置文件。

```
vim /usr/local/hadoop-2.7.2/etc/hadoop/core-site.xml
```

修改core-site.xml配置文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>dfs://f-4b1fcae5dvexx.cn-hangzhou.dfs.aliyuncs.com:10290</value>
    <!-- 该地址填写自己的HDFS挂载点地址 -->
  </property>
  <property>
    <name>fs.dfs.impl</name>
    <value>com.alibaba.dfs.DistributedFileSystem</value>
  </property>
  <property>
    <name>fs.AbstractFileSystem.dfs.impl</name>
    <value>com.alibaba.dfs.DFS</value>
  </property>
  <property>
    <name>io.file.buffer.size</name>
    <value>8388608</value>
  </property>
  <property>
    <name>alidfs.use.buffer.size.setting</name>
    <value>false</value>
    <!-- 建议不开启，亲测开启后会严重降低iosize，进而影响数据吞吐 -->
  </property>
  <property>
    <name>dfs.usergroupservice.impl</name>
    <value>com.alibaba.dfs.security.LinuxUserGroupService.class</value>
  </property>
  <property>
    <name>dfs.connection.count</name>
    <value>256</value>
  </property>
</configuration>
```

 **注意** 由于我们是on k8s，所以yarn相关的配置项不用配置，只用配置HDFS相关的几个配置项。修改后的core-site.xml文件后在面很多地方会用到。

3、执行如下命令打开/etc/profile配置文件。

```
vim /etc/profile
```

添加环境变量。

```
export HADOOP_HOME=/usr/local/hadoop-2.7.2
export HADOOP_CLASSPATH=/usr/local/hadoop-2.7.2/etc/hadoop:/usr/local/hadoop-2.7.2/share/hadoop/common/lib/*:/usr/local/hadoop-2.7.2/share/hadoop/hdfs/lib/*:/usr/local/hadoop-2.7.2/share/hadoop/yarn/lib/*:/usr/local/hadoop-2.7.2/share/hadoop/mapreduce/lib/*:/usr/local/hadoop-2.7.2/contrib/capacity-scheduler/*.jar
export HADOOP_CONF_DIR=/usr/local/hadoop-2.7.2/etc/hadoop
```

执行如下命令使配置生效。

```
source /etc/profile
```



注意 我们只需要一个HDFS client即可，不需要部署HDFS集群。

4、添加阿里云HDFS依赖

```
cp aliyun-sdk-dfs-1.0.3.jar /usr/local/hadoop-2.7.2/share/hadoop/hdfs
```

下载地址：[此处](#)下载文件存储HDFS的SDK。

4. 上传数据

```
#创建数据目录
[root@liumi-hdfs ~]# $HADOOP_HOME/bin/hadoop fs -mkdir -p /pod/data
#将本地准备的文件（一本小说文本）上传到hdfs
[root@liumi-hdfs ~]# $HADOOP_HOME/bin/hadoop fs -put ./A-Game-of-Thrones.txt /pod/data/A-Game-of-Thrones.txt
#查看，文件大小为30G
[root@liumi-hdfs local]# $HADOOP_HOME/bin/hadoop fs -ls /pod/data
Found 1 items
-rwxrwxrwx  3 root root 33710040000 2019-11-10 13:02 /pod/data/A-Game-of-Thrones.txt
```

至此HDFS数据准备部分就已经ready。

在spark应用中读取HDFS的数据

1. 开发应用

应用开发上跟传统的部署方式没有区别。


```
SparkConf conf = new SparkConf().setAppName(WordCount.class.getSimpleName());
JavaRDD<String> lines = sc.textFile("dfs://f-4b1fcae5dvxxx.cn-hangzhou.dfs.aliyuncs.com:10290/pod/data/A-Game-of-Thrones.txt", 250);
...
wordsCountResult.saveAsTextFile("dfs://f-4b1fcae5dvxxx.cn-hangzhou.dfs.aliyuncs.com:10290/pod/data/A-Game-of-Thrones-Result");
sc.close();
```

2. HDFS的配置

有两种常见的配置方式：1) 静态配置文件。2) 提交应用的时候动态设置

1) 将前面的core-site.xml放入应用项目的resources目录

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <!-- HDFS 配置-->
  <configuration>
    <property>
      <name>fs.defaultFS</name>
      <value>dfs://f-4b1fcae5dvexx.cn-hangzhou.dfs.aliyuncs.com:10290</value>
      <!-- 该地址填写自己的HDFS挂载点地址 -->
    </property>
    <property>
      <name>fs.dfs.impl</name>
      <value>com.alibaba.dfs.DistributedFileSystem</value>
    </property>
    <property>
      <name>fs.AbstractFileSystem.dfs.impl</name>
      <value>com.alibaba.dfs.DFS</value>
    </property>
    <property>
      <name>io.file.buffer.size</name>
      <value>8388608</value>
    </property>
    <property>
      <name>alidfs.use.buffer.size.setting</name>
      <value>false</value>
      <!-- 建议不开启，亲测开启后会严重降低iosize，进而影响数据吞吐 -->
    </property>
    <property>
      <name>dfs.usergroupservice.impl</name>
      <value>com.alibaba.dfs.security.LinuxUserGroupService.class</value>
    </property>
    <property>
      <name>dfs.connection.count</name>
      <value>256</value>
    </property>
  </configuration>
</configuration>
```

2) 以spark为例，也可以在提交应用时设置

```
hadoopConf:
  # HDFS
  "fs.defaultFS": "dfs://f-4b1fcae5dvexx.cn-hangzhou.dfs.aliyuncs.com:10290"
  "fs.dfs.impl": "com.alibaba.dfs.DistributedFileSystem"
  "fs.AbstractFileSystem.dfs.impl": "com.alibaba.dfs.DFS"
```

3. 打包的jar文件需要包含所有依赖

```
mvn assembly:assembly
```

附应用的pom.xml:

```
1<?xml version="1.0" encoding="UTF-8"?>
2<project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5  <modelVersion>4.0.0</modelVersion>
6
7  <groupId>com.aliyun.liumi.spark</groupId>
8  <artifactId>SparkExampleJava</artifactId>
9  <version>1.0-SNAPSHOT</version>
10
11  <dependencies>
12    <dependency>
13      <groupId>org.apache.spark</groupId>
14      <artifactId>spark-core_2.12</artifactId>
15      <version>2.4.3</version>
16    </dependency>
17
18    <dependency>
19      <groupId>com.aliyun.dfs</groupId>
20      <artifactId>aliyun-sdk-dfs</artifactId>
21      <version>1.0.3</version>
22    </dependency>
23
24  </dependencies>
25
26  <build>
27    <plugins>
28      <plugin>
29        <groupId>org.apache.maven.plugins</groupId>
30        <artifactId>maven-assembly-plugin</artifactId>
31        <version>2.6</version>
32        <configuration>
33          <appendAssemblyId>>false</appendAssemblyId>
34          <descriptorRefs>
35            <descriptorRef>jar-with-dependencies</descriptorRef>
36          </descriptorRefs>
37          <archive>
38            <manifest>
39              <mainClass>com.aliyun.liumi.spark.example.WordCount</mainClass>
40            </manifest>
```

```
41         </archive>
42     </configuration>
43     <executions>
44         <execution>
45             <id>make-assembly</id>
46             <phase>package</phase>
47             <goals>
48                 <goal>assembly</goal>
49             </goals>
50         </execution>
51     </executions>
52 </plugin>
53 </plugins>
54 </build>
55</project>
```

4. 编写Dockerfile

```
# spark base image
FROM registry.cn-hangzhou.aliyuncs.com/eci_open/spark:2.4.4
# 默认的kubernetes-client版本有问题，建议用最新的
RUN rm $SPARK_HOME/jars/kubernetes-client-*.jar
ADD https://repo1.maven.org/maven2/io/fabric8/kubernetes-client/4.4.2/kubernetes-client-4.4
    .2.jar $SPARK_HOME/jars
# 拷贝本地的应用jar
RUN mkdir -p /opt/spark/jars
COPY SparkExampleJava-1.0-SNAPSHOT.jar /opt/spark/jars
```

5. 构建应用镜像

```
docker build -t registry.cn-beijing.aliyuncs.com/liumi/spark:2.4.4-example -f Dockerfile .
```

6. 推到阿里云ACR

```
docker push registry.cn-beijing.aliyuncs.com/liumi/spark:2.4.4-example
```

至此，镜像都已经准备完毕。接下来就是在kubernetes集群中部署Spark应用了。

13.2. 在ECI中访问OSS数据

OSS是Hadoop/Spark批处理作业可选的数据存储之一。本文将演示在OSS中创建一个文件，并在Spark中进行访问。

OSS数据准备

1. 创建bucket。

创建 Bucket ① 创建存储空间 ×

注意： Bucket 创建成功后，您所选择的 **存储类型、区域** 不支持变更。

Bucket 名称: spark-on-k8s 12/63 ✓

区域: 华东1 (杭州) ▼

Endpoint: oss-cn-hangzhou.aliyuncs.com

存储类型: 标准存储 低频访问 归档存储

读写权限: 私有 公共读 公共读写

服务器端加密: 无 AES256 KMS

提示： 将文件上传至 OSS 后，自动对其进行落地加密存储，KMS 加密方式需要进行权限设置，当前 KMS 仅支持 OSS 默认托管的 CMK，如需试用 KMS 单独创建的 CMK 加密 (BYOK)，请和

确定 取消

2. 上传文件

上传文件可以通过OSS SDK，也可以[通过HDP2.6 Hadoop读取和写入OSS数据](#)，我们直接以控制台为例：



记住这个bucket的地址：oss://liumi-hust/test.txt，endpoint：oss-cn-hangzhou-internal.aliyuncs.com，至此OSS数据准备部分就已经完成。

在spark应用中读取OSS的数据

1. 开发应用

应用开发上跟传统的部署方式没有区别。

```
SparkConf conf = new SparkConf().setAppName(WordCount.class.getSimpleName());
JavaRDD<String> lines = sc.textFile("oss://liumi-hust/test.txt", 250);
...
wordsCountResult.saveAsTextFile("oss://liumi-hust/test-result");
sc.close();
```

2. 有两种常见的配置方式：1) 静态配置文件。2) 提交应用的时候动态设置

1) 将前面的core-site.xml放入应用项目的resources目录

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
    http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->
<!-- Put site-specific property overrides in this file. -->
<configuration>
  <!-- OSS配置 -->
  <property>
    <name>fs.oss.impl</name>
    <value>org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem</value>
  </property>
  <property>
    <name>fs.oss.endpoint</name>
    <value>oss-cn-hangzhou-internal.aliyuncs.com</value>
  </property>
  <property>
    <name>fs.oss.accessKeyId</name>
    <value>{临时AK_ID}</value>
  </property>
  <property>
    <name>fs.oss.accessKeySecret</name>
    <value>{临时AK_SECRET}</value>
  </property>
  <property>
    <name>fs.oss.buffer.dir</name>
    <value>/tmp/oss</value>
  </property>
  <property>
    <name>fs.oss.connection.secure.enabled</name>
    <value>>false</value>
  </property>
  <property>
    <name>fs.oss.connection.maximum</name>
    <value>2048</value>
  </property>
</configuration>
```

2) 以spark为例，也可以在提交应用时设置

```
hadoopConf:
  # OSS
  "fs.oss.impl": "org.apache.hadoop.fs.aliyun.oss.AliyunOSSFileSystem"
  "fs.oss.endpoint": "oss-cn-hangzhou-internal.aliyuncs.com"
  "fs.oss.accessKeyId": "your ak-id"
  "fs.oss.accessKeySecret": "your ak-secret"
```

3. 打包的jar文件需要包含所有依赖

```
mvn assembly:assembly
```

附应用的pom.xml:

```
1<?xml version="1.0" encoding="UTF-8"?>
2<project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xs
d/maven-4.0.0.xsd">
5  <modelVersion>4.0.0</modelVersion>
6
7  <groupId>com.aliyun.liumi.spark</groupId>
8  <artifactId>SparkExampleJava</artifactId>
9  <version>1.0-SNAPSHOT</version>
10
11  <dependencies>
12    <dependency>
13      <groupId>org.apache.spark</groupId>
14      <artifactId>spark-core_2.12</artifactId>
15      <version>2.4.3</version>
16    </dependency>
17
18    <dependency>
19      <groupId>com.aliyun.dfs</groupId>
20      <artifactId>aliyun-sdk-dfs</artifactId>
21      <version>1.0.3</version>
22    </dependency>
23
24  </dependencies>
25
26  <build>
27    <plugins>
28      <plugin>
29        <groupId>org.apache.maven.plugins</groupId>
30        <artifactId>maven-assembly-plugin</artifactId>
31        <version>2.6</version>
32        <configuration>
33          <appendAssemblyId>>false</appendAssemblyId>
34          <descriptorRefs>
35            <descriptorRef>jar-with-dependencies</descriptorRef>
36          </descriptorRefs>
37          <archive>
38            <manifest>
39              <mainClass>com.aliyun.liumi.spark.example.WordCount</mainClass>
```

```
40         </manifest>
41     </archive>
42 </configuration>
43 <executions>
44     <execution>
45         <id>make-assembly</id>
46         <phase>package</phase>
47         <goals>
48             <goal>assembly</goal>
49         </goals>
50     </execution>
51 </executions>
52 </plugin>
53 </plugins>
54 </build>
55</project>
```

4. 编写Dockerfile

OSS:

```
# spark base image
FROM registry.cn-beijing.aliyuncs.com/eci_open/spark:2.4.4
RUN rm $SPARK_HOME/jars/kubernetes-client-*.jar
ADD https://repo1.maven.org/maven2/io/fabric8/kubernetes-client/4.4.2/kubernetes-client-4.4
.2.jar $SPARK_HOME/jars
RUN mkdir -p /opt/spark/jars
COPY SparkExampleJava-1.0-SNAPSHOT.jar /opt/spark/jars
# oss 依赖jar
COPY aliyun-sdk-oss-3.4.1.jar /opt/spark/jars
COPY hadoop-aliyun-2.7.3.2.6.1.0-129.jar /opt/spark/jars
COPY jdom-1.1.jar /opt/spark/jars
```

oss 依赖jar下载地址请参见[通过HDP2.6 Hadoop读取和写入OSS数据](#)。

5. 构建应用镜像

```
docker build -t registry.cn-beijing.aliyuncs.com/liumi/spark:2.4.4-example -f Dockerfile .
```

6. 推到阿里云ACR

```
docker push registry.cn-beijing.aliyuncs.com/liumi/spark:2.4.4-example
```

至此，spark应用镜像都已经准备完毕。接下来就是在kubernetes集群中部署Spark应用了。