Serverless 容器引擎 SCE

用户操作指南

天翼云科技有限公司

1 产品简介

1.1 产品定义

Serverless 容器引擎 SCE 是天翼云提出的 Serverless Kubernetes 容器服务。相比与传统 Kubernetes 集群，SCE 集群无需购买节点即可直接部署容器应用，同时无需对集群进行节点维护和容量规划，降低了 Kubernetes 使用门槛，让用户更专注于应用程序，而不是管理底层基础设施。SCE 集群提供完善的 Kubernetes 兼容能力，您可以轻松迁移已有的 Kubernetes 应用到 SCE 集群上，并且根据应用配置的 CPU 和内存资源量进行按需付费。

SCE 集群中的每个 Pod 都是基于天翼云弹性容器实例 ECI 运行，实现了安全隔离的容器运行环境。每个容器底层采用轻量级虚拟化安全沙箱技术，实现了容器实例间的强隔离，避免了容器实例间相互影响的问题。SCE 集群还具有易用性、灵活性和安全性等优点，能够满足不同规模应用场景的需求，并提供了完善的监控、日志和核心运维能力。

1.2 功能特性

虚拟节点

SCE 集群无需管理节点，但为了兼容原生 Kubernetes，以及提供应对突发业务流量的弹性能力，您仍会在集群中看到虚拟节点的存在。

Pod 配置

在 SCE 集群中创建 Pod 时，您可以通过添加 Annotation 来定制 Pod。

网络管理

SCE 集群支持 Service 和 Ingress 等对象的个性化定制，并且允许通过 CoreDNS 和弹性 IP 等办法提供服务发现功能。

存储管理

支持天翼云盘挂载，提供标准的 CSI，支持存储卷自动创建。

可观测性

SCE 集群支持安装相应组件启动监控功能。

镜像管理

SCE 集群支持使用 ImageCache 来加速创建 Pod，帮助您快速响应业务。

组件管理

SCE 集群提供多种类型的组件，以扩展集群的各种功能。根据业务需求，您可以随时部署、升级或卸载这些组件。

应用管理

SCE 集群支持灰度发布、蓝绿发布、应用监控以及应用弹性伸缩。同时，内置的模板市场支持 Helm 应用一键部署，大大简化云服务集成。

1.3 产品优势

开箱即用

一键创建 Kubernetes 集群、直接部署应用程序，无需管理 Kubernetes 节点和服务器。

原生兼容

提供完善的 Kubernetes 兼容能力，支持原生 Kubernetes 应用和生态，可以轻松迁移已有的 Kubernetes 应用程序。

安全隔离

Pod 基于 ECI 服务运行，实现了安全隔离的容器运行环境，避免了容器实例间相互影响的问题。

降低成本

提供按需创建、按量计费的模式，避免了不必要的资源浪费和成本费用，同时 Serverless 也大大降低了运维成本。

服务集成

支持与天翼云基础服务无缝集成，可以使用一体化控制台高效操作。

1.4 应用场景

应用托管

SCE 集群无需购买节点即可轻松部署容器应用，无需对集群进行节点维护和容量规划，大大降低业务的基础设施管理和运维成本，提供高效的应用托管服务。

突发业务

面对有明显波峰波谷特征的业务负载，SCE 集群的秒级伸缩能力能够以最低的成本代价平滑应对流量高峰，保证业务的高可用和性能。

数据计算

面对数据分析或机器学习训练等任务，SCE 集群可以提供快速、灵活的计算资源，快速启动大量 Pod 实例在短时间内运行特定的计算任务，计算结束后释放自动停止计费，极大降低了整体的计算成本。

CI/CD

SCE 集群可以轻松搭建各种持续集成环境，帮助用户快速构建测试环境和自动化的部署流程，同时为各种持续集成任务之间提供隔离性和安全性。

1.5 使用限制

使用 Serverless 容器引擎 SCE 前，需要注意以下使用限制：

- 不支持 DaemonSet 型的工作负载，如果想要使用，您可以通过将 DaemonSet 重新配置为 Pod 的 Sidecar 容器来运行。
- 不支持在 Pod 的 manifest 中指定 HostPath 和 HostNetwork。
- 不支持 Privileged 权限容器，您可使用 Security Context 为 Pod 添加 Capability。
- 不支持 NodePort 类型的 Service。

2 快速入门

2.1 Serverless 容器引擎 SCE 使用快速入门

前提条件

- 已开通并授权云容器引擎。
- 已登录弹性容器实例控制台开通 ECI 服务。

操作步骤

创建 SCE 集群

1. 登录云容器引擎 CCSE 控制台。

2. 在控制台的左侧导航栏中点击“集群”。

3. 在集群列表页面中，单击页面右上角的“创建 SCE 集群”，进入订购 Serverless 容器引擎界面。

4. 在订购 Serverless 容器引擎页面中，完成集群的基本配置。部分配置说明如下：

配置项	描述
计费模式	默认使用按需计费模式
虚拟私有云	集群所使用的 VPC，支持使用已有的 VPC 或创建虚拟私有云
所在子网	集群所使用的子网，支持使用已有的子网或创建子网

安全组	安全组是一种虚拟防火墙，能够控制实例的出入站流量，SCE 集群支持选择已有安全组，或创建新的安全组
启用 IPv6	启用 IPv6 双栈将创建双栈 SCE 集群
Service CIDR	设置 Service CIDR。Service 网段不能与 VPC 及 VPC 内已有集群使用的网段重复，并且 Service 地址段也不能和 Pod 地址段重复。集群创建成功后不能再修改该网段
Kubernetes 版本	显示当前 SCE 集群支持的 Kubernetes 版本
API Server 访问	API Server 的访问需要依赖 ELB 实例，您可根据需要选择合适的 ELB 规格，系统将根据该规格创建一个私网 ELB 实例。同时，您也可以选择是否使用 EIP 暴露 API Server，选择不开放时，则无法通过外网访问集群 API Server
实例名称	填写集群的实例名称
企业项目	按需选择企业项目
时区	集群所要使用的时区
部署模式	支持单可用区部署、多可用区部署
可用区	按需选择可用区
集群删除保护	设置是否启用集群删除保护，防止通过控制台或者 API 误删除集群
集群本地域名	填写集群的本地域名

5. 集群配置完成后，单击“下一步”。

6. 确定产品名称、基础配置以及费用无误后提交订单。

7. 集群创建成功后，您可以进入云容器引擎控制台，在集群选项卡中可以看到新创建的集群。点击集群名称进入集群详情界面，可以查看集群相关信息。

使用镜像创建应用

步骤参考

1. 在群管理页面的左侧导航栏中，选择“工作负载”。

2. 点击“无状态”，在无状态页面中，单击左上角的“创建 Deployment”。

3. 进入新建 Deployment 页面，设置应用的基本信息。

配置项	描述
Deployment 名称	工作负载的名称
数据卷（选填）	为容器提供存储，目前支持 configMap、临时目录、secret、使用已有 PVC 以及 downwardApi，此外数据卷还需挂载到容器的指定路径中
副本数量	工作负载的副本数
指标伸缩	为工作负载定义伸缩规格和伸缩范围，支持新增 Rersource 规则、Pod 规则以及 Object 规则
实例内容器	工作负载中的容器实例配置，可配置一个或多个

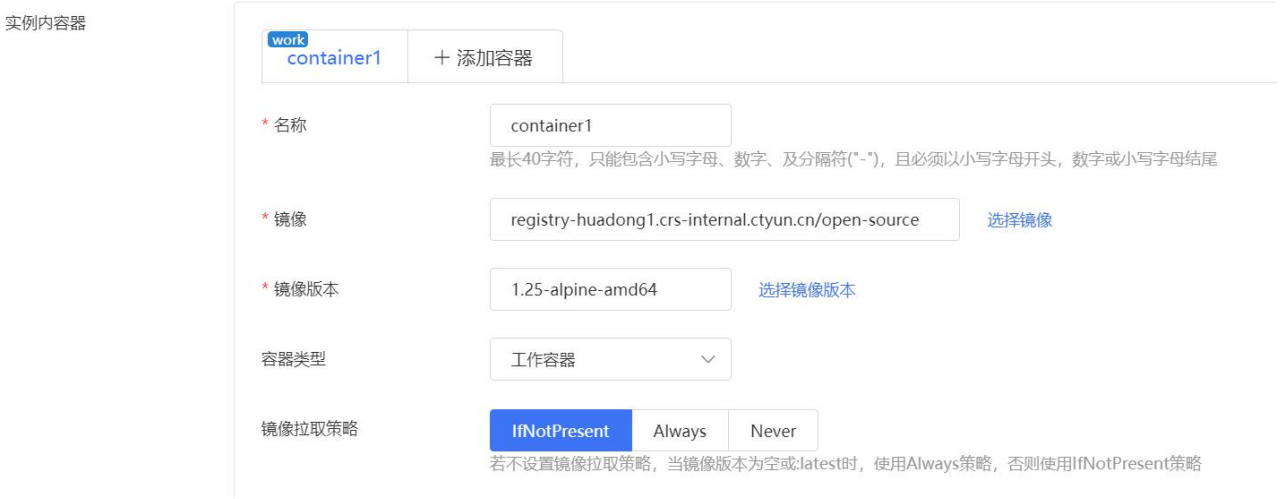
容器名称	填写容器的名称
镜像及镜像版本	支持在容器镜像服务或开源镜像中选择镜像以及镜像版本
容器类型	按需选择容器类型
镜像拉取策略	支持 IfNotPresent、Always、Never
挂载点（选填）	支持挂载数据卷到容器内的指定路径
CPU/内存限制	Request 用于预分配资源，当集群中的节点没有 request 所要求的资源数量时，容器会创建失败。Limit 用于设置容器使用资源的最大上限，避免异常情况下节点资源消耗过多
环境变量（选填）	支持配置容器的环境变量
启动执行（选填）	● 命令：对应镜像的 ENTRYPOINT 命令，将会覆盖镜像的 ENTRYPOINT 命令；每个输入框仅输入一个命令或参数启动执行 ● 参数：对应镜像的 CMD 命令，将会覆盖镜像的 CMD 命令；每个输入框仅输入一个命令或参数
启动后处理（选填）	容器启动后执行，注意由于是异步执行，无法保证一定在 ENTRYPOINT 之后运行；每个输入框仅输入一个命令或参数
停止前处理（选填）	容器停止前执行，常用于资源清理。每个输入框仅输入一个命令或参数
容器健康检查（选填）	● 存活检查：检查容器是否正常，不正常则重启实例 ● 就绪检查：检查容器是否就绪，不就绪则停止转

	发流量到当前实例
高级配置（选填）	按需配置负载标签、负载注解、Pod 标签、Pod 注解等信息
访问设置（选填）	配置 Service 访问负载

4. 信息填写完成后，点击“提交”。

示例

1. 在本次设置中，设置 Deployment 名称为 test，镜像选择 nginx。



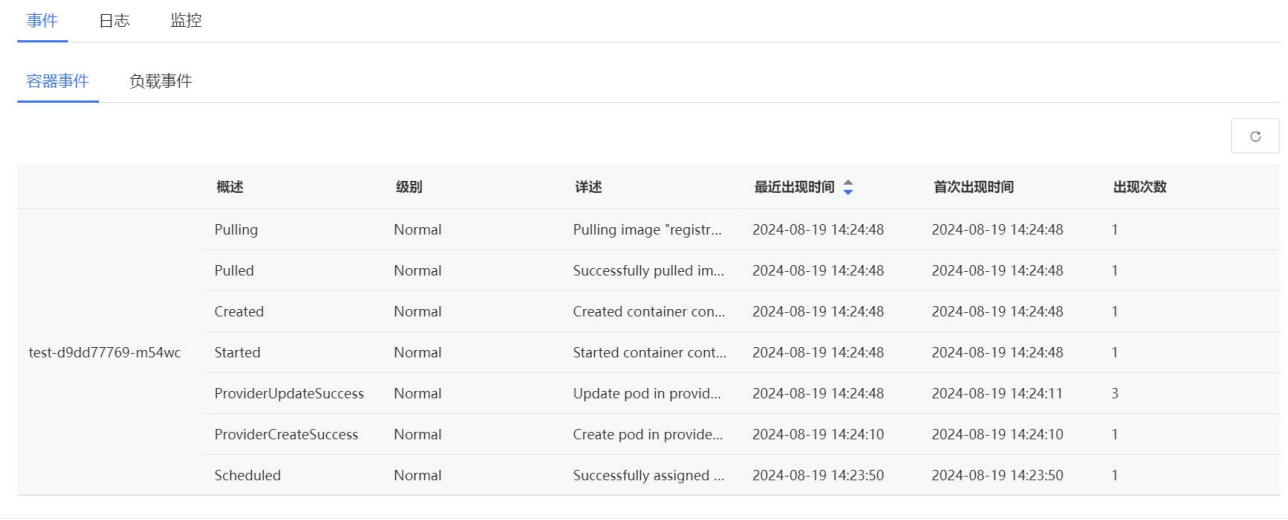
2. 在访问设置中，服务访问方式中选择虚拟集群 IP，协议选择 TCP，容器端口和服务端口分别配置为 80 和 30003。

3. 创建完成后，进入工作负载的无状态页面，可以看到新建的 nginx 应用出现在无状态列表下。

4. 在群管理页面的左侧导航栏中，选择“网络”下的“服务”页面，可以看到新建的 nginx 服务出现在服务列表下。

5. 在群管理页面的左侧导航栏中，选择“工作负载”，点击“容器组”，在容器组页面下可以查看所有创建的容器。

6. 点击想要查看的实例名称，即可进入查看该容器组的详细信息。



2.2 通过 SCE 快速部署 Nginx 应用

SCE 集群不需要用户管理维护节点，让您将精力放在具体应用的开发和维护上，而不是底层基础设施的管理。本文将介绍如何在 SCE 上实现在线 Web 应用的免运维托管。

前提条件

确保您已经成功创建了 SCE 集群。

背景信息

SCE 兼容原生 Kubernetes 语义和 API，您可以在 SCE 集群中轻松创建 Deployment、StatefulSet、Service、Ingress、ConfigMap 或 Secret 等资源。此外，您也可以使用 Helm 部部署和管理复杂的 Kubernetes 应用程序的生命周期。

操作步骤

1. 通过 kubectl 工具连接 SCE 集群。

2. 创建 Nginx 应用的 YAML 配置文件 nginx.yaml，内容示例如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 30003
      targetPort: 80
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: nginx-deployment
  namespace: default
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image:registry-huadong1.crs-internal.ctyun.cn/open-source/nginx:1.25-alpine
        resources:
          limits:
            cpu: "1"
            memory: "1Gi"
```

```
        requests:
          cpu: "0.5"
          memory: "500Mi"
      ports:
        - containerPort: 80
```

3. 通过该配置文件部署 Nginx 应用。

```
kubectl apply -f nginx.yaml
```

预期返回为：

```
service/nginx-service created
deployment.apps/nginx-deployment created
```

4. 查看创建 Pod 和 Serivce 的状态，查看 Pod 的状态信息：

```
kubectl get pod
```

预期输出结果为：

```
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-7d4df6ffc8-52dr8   1/1     Running   0          12m
nginx-deployment-7d4df6ffc8-gshq4   1/1     Running   0          12m
```

查看 Service 的状态信息：

```
kubectl get svc
```

预期输出结果为：

```
NAME          TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)     AGE
nginx-service   ClusterIP   10.96.75.52   <none>        30003/TCP   28m
```

5. 访问 Nginx 应用：

```
curl 10.96.75.52:30003
```

预期输出结果为：

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome to nginx!</title>
...
</body>
</html>
```

2.3 基于 SCE 集群快速部署 FastChat 应用

前提条件

已开通 SCE 集群，并且能通过公网访问集群。

背景信息

SCE 兼容原生 Kubernetes 语义和 API，您可以在 SCE 集群中轻松创建 Deployment、StatefulSet、Service、Ingress、PersistentVolume、

ConfigMap 或 CRD 等资源。此外，您也可以使用 Helm 部部署和管理复杂的 Kubernetes 应用程序的生命周期。

FastChat 介绍

FastChat 是一个用于训练、部署和评估基于大型语言模型的聊天机器人的开放平台。其核心功能包括：最先进模型的权重、训练代码和评估代码（例如 Vicuna、FastChat-T5）；基于分布式多模型的服务系统，具有 Web 界面和与 OpenAI 兼容的 RESTful API。

操作步骤

创建 FastChat 应用

通过控制台部署 FastChat 应用，也可以通过 kubectl 工具连接 sce 集群来创建 FastChat 应用。

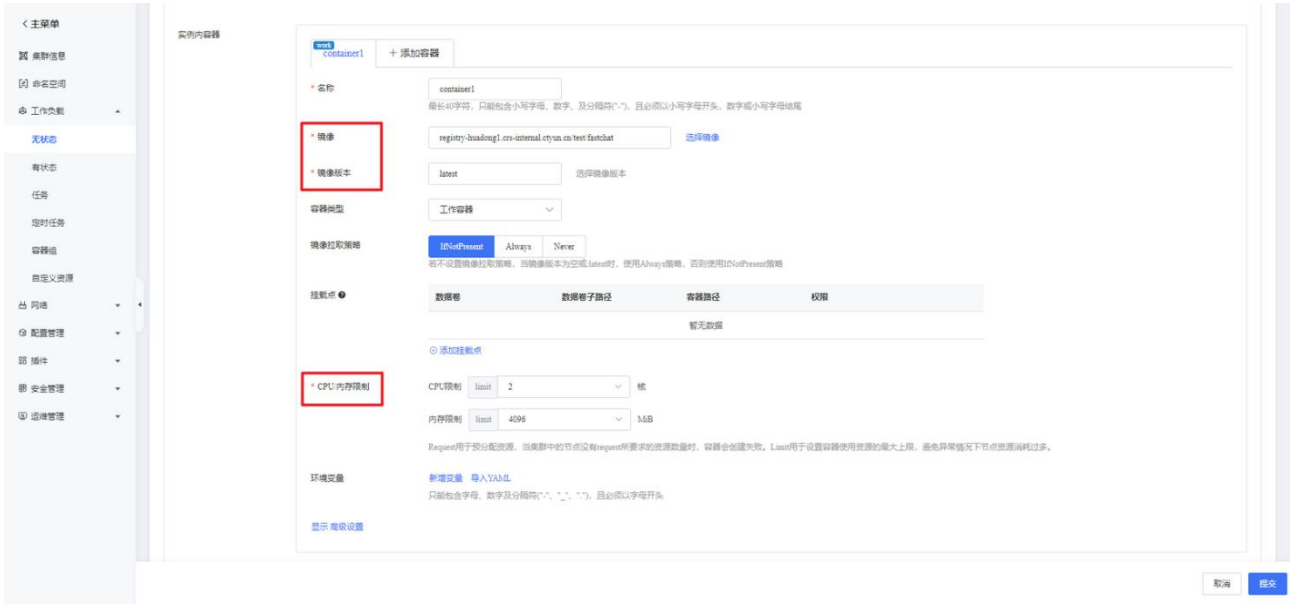
1. 登录管理控制台，在左侧菜单栏选择“集群”。

2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏，选择“工作负载”下的“无状态”，选择“创建 Deployment”。

3. 在创建 Deployment 页面，填写 Deployment 名称、副本数量等。

4. 在实例内容器项填写容器名称、镜像、镜像版本、cpu/内存限制等。

注：fastchat 镜像要提前上传到 CRS 容器镜像服务，点击选择镜像选择 fastchat 镜像即可。

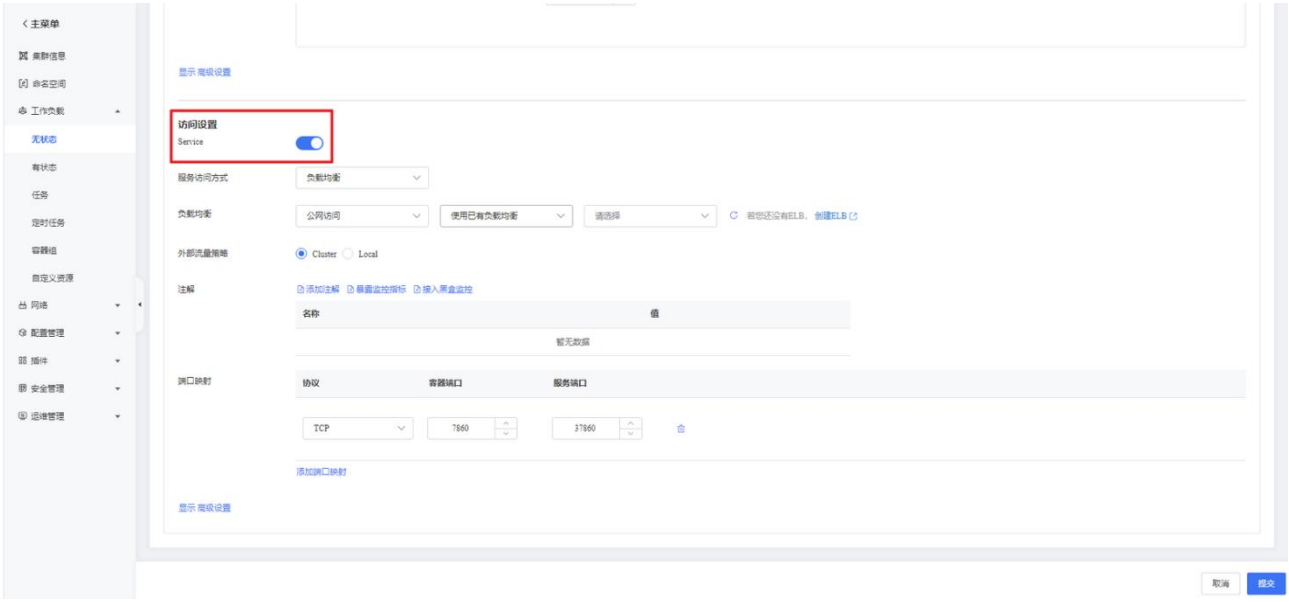


5. 在实例内容器项内点击“显示高级设置”，添加启动执行命令。

6. 开启“容器健康检查”，勾选“就绪检查”。

7．在访问设置项，点击“开启 service”，设置服务相关参数，通过该服务公开 fastchat 应用。

注：需要提前手工创建 ELB。



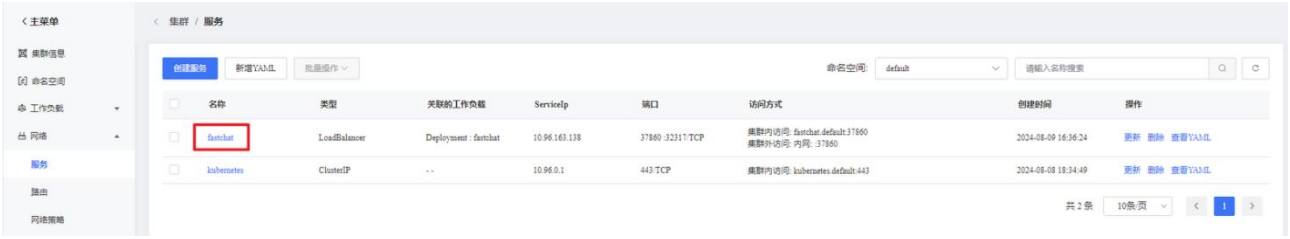
8．点击“提交”，返回到如下页面表示创建成功，等待 deployment 的副本 pod 运行起来即可。。

访问服务

1. 登录管理控制台，在左侧菜单栏选择“集群”。

2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏，选择“网络”下的服务“服务”。

3. 在访问方式看到可以通过集群内访问或者集群外访问。



3 用户指南

3.1 ECI Pod

3.1.1 ECI 实例概述

前提条件

- 已开通并授权云容器引擎。
- 已登录弹性容器实例控制台开通 ECI 服务。

Kubernetes 应用限制

借助 Kubernetes 社区的 Virtual Kubelet 技术，天翼云 ECI 可以完美连接到 Kubernetes，实现真正意义上的无缝连接。ECI 实例并不会在一个集中式的真实节点上运行，而是分布在整个天翼云的资源池中。由于公共云的安全性和虚拟节点本身的限制，ECI 目前还不支持 Kubernetes 中的一些功能，如 DaemonSet。具体功能限制请参见下表：

不支持的功能	说明
HostPath	允许将宿主机（Node）上的文件或目录挂载到 Pod 中
HostNetWork	允许 Pod 使用宿主机的网络命名空间，而不是使用 Kubernetes 的网络隔离

DaemonSet	用于确保每个集群节点上自动运行一个指定的 Pod 副本
Privileged 权限	允许容器几乎拥有宿主机级别的权限
type=NodePort 的 Service	通过在集群的节点上暴露一个静态端口，使得外部可以通过指定 IP 地址和该端口访问服务

核心功能

功能项	说明
安全隔离	提供虚拟机级别的安全和资源隔离能力，每个容器实例都运行在独占内核中，不与其它负载和 Pod 共享基础设施资源。同时针对容器运行环境进行了深度优化，具备比虚拟机更快的启动速度和运行效率
CPU/Memory 资源或规格配置	ECI Pod 默认使用按需计费模式进行费用收取。支持指定 CPU 和 Memory 资源或者指定 ECS 规格创建实例
镜像拉取与缓存	• 镜像拉取：ECI Pod 在每次启动时，会自动从远程仓库获取容器镜像。对于公共镜像的获取，建议配置 VPC 的 NAT 网关或为 ECI Pod 配置弹性公网 IP (EIP)。为优化镜像拉取效率，我们推荐您使用天翼云容器镜像服务，加速镜像的下载 • 镜像缓存：ECI 提供镜像缓存功能，镜像缓存是为了加速拉取镜像以减少 ECI 启动时间而设计的。镜像拉取是容器实例启动的主要耗时，而制作镜像缓存可以通过预先获取、存储和管理已经拉取的镜像，实现对容器实例启动时间的显著减少。考虑到网络和镜像大小等因素的影响，构建镜像缓存可以通过连续使用相同的镜像实现快速部署，从而加速容器实例的启动并提高系统的可用性

存储	支持使用多种存储方式： • CSI：CSI 插件是目前 Kubernetes 社区推荐的插件实现方案，SCE 集群所提供的 CSI 存储插件与社区 CSI 特性兼容。该插件由以下两个组件组成：CSI-Plugin：提供挂载和卸载数据卷的能力，SCE 默认支持云硬盘和弹性文件服务两种存储服务；CSI-Provisioner：提供自动挂载数据卷的能力 • PV/PVC：PV 提供长期存储资源，而 PVC 允许用户以抽象的方式请求这些存储资源，实现存储的分配和管理 • EmptyDir：该数据卷是一种用于容器实例中临时存放数据的目录，以便于容器之间共享数据。但是需要注意的是，当容器实例被删除时，EmptyDir 数据卷中的数据也会被清空
网络	ECI Pod 默认采用 Host 网络模式，并会占用交换机 vSwitch 的一个弹性网卡 ENI 资源。在 Kubernetes 集群环境中，ECI Pod 与云主机节点上的 Pod 可以相互访问。具体方法如下： • 创建类型为 LoadBalancer 的 Service 对象，并与 ECI Pod 进行关联；也支持 Service 同时关联 ECI Pod 和云主机上的 Pod • 创建类型为 ClusterIP 的 Service 对象，ECI Pod 可以直接访问集群中的 clusterIP 地址 • 配置相应的 NAT 网关或弹性公网 EIP，并为 ECI Pod 绑定指定 EIP，或者将 NAT 网关绑定到 ECI 实例所属的 VPC 网络中
日志采集	通过安装日志采集服务插件，一般情况无需再额外部署 sidecar 容器

3.1.2 通过指定 CPU 和内存创建 ECI Pod

您可以通过指定 vCPU 和内存来创建 ECI Pod，系统会尝试使用多种云主机规格来支撑您的实例，以提供比单一云主机规格更好的弹性和资源供应能力。本文将分别介绍如何指定 ECI 实例的容器规格和 ECI Pod 的规格。

规格说明

在创建 ECI Pod 时，如果指定的 vCPU 和内存大小不符合 ECI 支持的规格要求，系统将会进行自动规整。在规整时，系统会将申请的实例规格向最接近的可支持的规格进行调整，同时需要确保所需的资源量不超过 ECI 的规格限制，以获得最佳的性能和资源利用率。例如：在创建 ECI 实例时声明了 7 vCPU，13 GiB 内存，则实际创建的 ECI 实例为 8 vCPU，16 GiB 内存。

注：如果没有指定 vCPU 和内存规格，系统将默认使用 2 vCPU 和 4 GiB 内存的规格来创建 ECI Pod。

使用示例

在创建 ECI Pod 时，通过定义容器中的 limits，可以指定 Pod 内容器的 vCPU 和内存。

注：在 Serverless 集群中，requests 会被忽略。

您可以通过直接定义容器的 limits 来指定该容器的 vCPU 和内存。具体的配置示例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
  name: nginx-test
  namespace: default
  labels:
    app: nginx-test
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-test
  template:
    metadata:
      labels:
        app: nginx-test
    spec:
      containers:
      - name: nginx
        image:registry-huadong1.crs-internal.ctyun.cn/open-
source/nginx:1.25-alpine
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: "1"
```

```
            memory: "2Gi"
```

3.1.3 使用 GPU 实例

ECI GPU 实例预装了显卡和 CUDA 驱动程序，这意味着在使用 ECI GPU 实例时，您只需选择集成了 CUDA Toolkit 等工具的标准镜像，无需额外安装任何驱动。本文将指导您如何有效利用 ECI GPU 实例

使用方法

- 在 Pod metadata 中添加 k8s.ctyun.cn/eci-use-specs 的 annotaions，选择合适的 ECI 支持 GPU 规格，目前 ECI 支持的 GPU 规格请见指定 ECS 规格创建实例。
- 在 Container 的 resources 中声明 GPU 资源，即 ctyun.cn/gpu，用于指定该容器使用的 GPU 个数。注意，容器使用 GPU 个数总和不能超过指定规格所包含的 GPU 数量。

具体实例如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox-gpu
  namespace: default
  labels:
    app: busybox-gpu
spec:
  selector:
    matchLabels:
      app: busybox-gpu
```

```
template:
  metadata:
    labels:
      app: busybox-gpu
    annotations:
      k8s.ctyun.cn/eci-use-specs: pi7.4xlarge.4
  spec:
    containers:
    - name: busybox
      image: "registry-huadong1.crs-internal.ctyun.cn/open-source/busybox:1.36"
      resources:
        limits:
          ctyun.cn/gpu: '1'
```

3.1.4 创建多可用区的 Pod

在面临突增流量时，您可能需要迅速水平扩展业务或启动众多实例以处理任务。这种情况下，您可能遭遇特定可用区内实例规格库存短缺或耗竭等问题，导致 ECI 实例创建不成功。借助 SCE 的多可用区功能，可以有效提高 ECI 实例的创建成功率。

使用方法

SCE 集群目前支持单可用区部署和多可用区部署。

- 单可用区部署：在订购 Serverless 容器引擎页面中，在部署模式下选择单可用区部署，接着需要在可用区里面手动指定可用区。
- 多可用区部署：在订购 Serverless 容器引擎页面中，在部署模式下选择多可用区部署。多可用区部署系统会自动将控制节点以及工作节点平均分配至各可用区，无需您手动指定。

3.1.5 为 Pod 配置时区

本文将介绍如何为 ECI Pod 配置不同的时区，确保您的应用程序、日志和时间戳记录遵循正确的时间和日期。

操作步骤

为 Kubernetes Pod 设置时区的最简单方法是在 Pod 中添加一个 Volume，然后将该 Volume 挂载到 Pod 中的某个目录。该目录可以包含代表时区的一个或多个文件。这种方法的优点是可以在 Pod 内的多个容器中重用时区设置，而无需在每个容器中都复制一遍。

您想要创建一个 configmap，并导入所需的时区信息。为了指定时区，需要进行相应的配置，请选择/usr/share/zoneinfo/Asia/目录下的配置文件进行导入。以下是一个示例：

1. 创建应用的 YAML 配置文件 timezone.yaml，内容示例如下：

```
apiVersion: v1
kind: Pod
metadata:
  name: timezone-pod-initcontainer
spec:
  initContainers:
```

```
- name: timezone-setup
  image: "registry-huadong1.crs-internal.ctyun.cn/open-source/nginx:1.25-alpine"
  command: ["/bin/sh", "-c"]
  args:
  - |
    cp /usr/share/zoneinfo/Asia/Shanghai /timezone/localtime
  volumeMounts:
  - name: timezone-config
    mountPath: /timezone
containers:
- name: main-container
  image: busybox
  command:
   - "tail"
   - "-f"
   - "/dev/null"
  volumeMounts:
  - name: timezone-config
    mountPath: /etc/localtime
    subPath: localtime
volumes:
- name: timezone-config
  emptyDir: {}
```

2.通过该配置文件部署 busybox 应用。

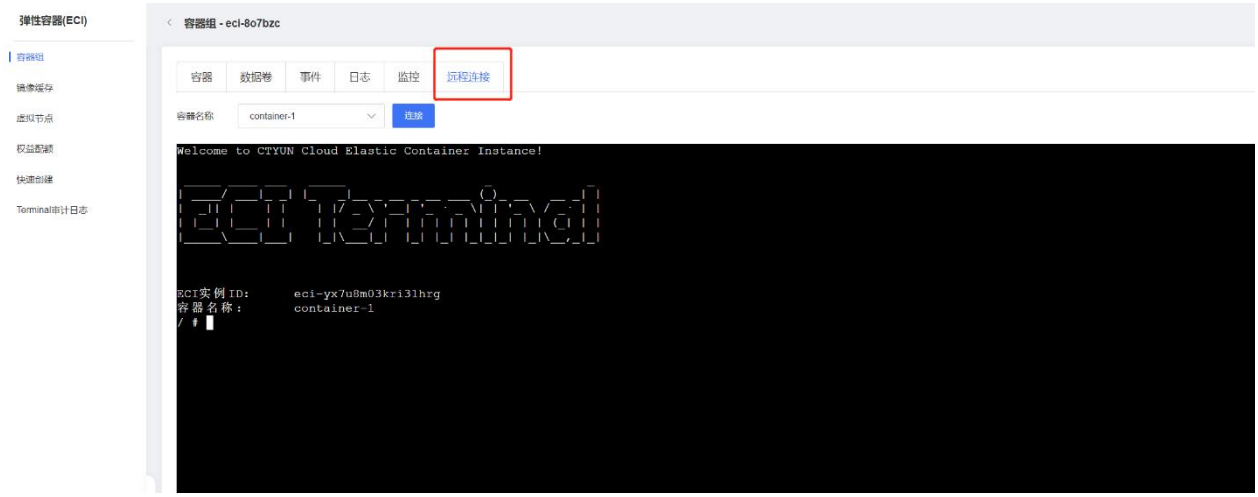
```
kubectl apply -f timezone.yaml
```

预期返回：

```
NAME              READY     STATUSRESTARTS     AGE
timezone          1/1       Running0           1m30s
```

3.进入指定容器。

通过弹性容器实例 ECI 控制台，选择指定的容器组，点击“远程连接”连接容器。



4.在容器中运行 date -R 命令，显示当前的日期和时间。如果返回的时间与您设置的时区信息相符，则表示设置成功。以下是设置成功后的示例返回结果：

```
/ # date -R
Fri, 26 Jan 2024 10:08:00 +0800
```

3.2 集群

3.2.1 SCE 集群概述

产品简介

Serverless 容器引擎 SCE 是天翼云提出的 Serverless Kubernetes 容器服务。相比与传统 Kubernetes 集群，SCE 集群无需购买节点即可直接部署容器应用，同时无需对集群进行节点维护和容量规划，降低了 Kubernetes 使用门槛，让用户更专注于应用程序，而不是管理底层基础设施。SCE 集群提供完善的 Kubernetes 兼容能力，您可以轻松迁移已有的 Kubernetes 应用到 SCE 集群上，并且根据应用配置的 CPU 和内存资源量进行按需付费。

使用场景

- 互联网企业：大规模业务上线生产环境，对管控的稳定性、可观测性和安全性有较高要求。
- 大数据计算企业：大规模数据计算、高性能数据处理、高弹性需求等类型业务，对集群稳定性、性能和效率有较高要求。

3.2.2 创建 SCE 集群

前提条件

- 已开通并授权云容器引擎。
- 已登录弹性容器实例控制台开通 ECI 服务。

步骤一：登录云容器引擎控制台

1.登录云容器引擎控制台。

2.在控制台的左侧导航栏中点击“集群” 。

3.在集群列表页面中，单击页面右上角的“创建 SCE 集群”，进入订购 Serverless 容器引擎界面。

步骤二：配置集群

在订购 Serverless 容器引擎页面中，完成集群的基本配置。部分配置说明如下：

配置项	描述
计费模式	默认使用按需计费模式
虚拟私有云	集群所使用的 VPC，支持使用已有的 VPC 或创建虚拟私有云
所在子网	集群所使用的子网，支持使用已有的子网或创建子网
安全组	安全组是一种虚拟防火墙，能够控制实例的出入站流量，SCE 集群支持选择已有安全组，或创建新的安全组
Service CIDR	设置 Service CIDR。Service 网段不能与 VPC 及 VPC 内已有集群使用的网段重复，并且 Service 地址段也不能和 Pod 地址段重复。集群创建成功后不能再修改该网段
Kubernetes 版本	显示当前 SCE 集群支持的 Kubernetes 版本

API Server 访问	API Server 的访问需要依赖 ELB 实例，您可根据需要选择合适的 ELB 规格，系统将根据该规格创建一个私网 ELB 实例。同时，您也可以选择是否使用 EIP 暴露 API Server，选择不开放时，则无法通过外网访问集群 API Server
实例名称	填写集群的实例名称
企业项目	按需选择企业项目
部署模式	支持单可用区部署、多可用区部署
可用区	按需选择可用区
集群删除保护	设置是否启用集群删除保护，防止通过控制台或者 API 误删除集群
集群本地域名	填写集群的本地域名

步骤三：确认配置

单击页面右下方“下一步” ，进行各种配置信息的确认 。 确认产品名称、基础配置以及费用无误后提交订单 。

集群创建成功后，您可以进入云容器引擎控制台，在集群选项卡中可以看到新创建的集群。点击集群名称进入集群详情界面，可以查看集群相关信息。

说明：一个 SCE 集群的创建时间通常约为十分钟。

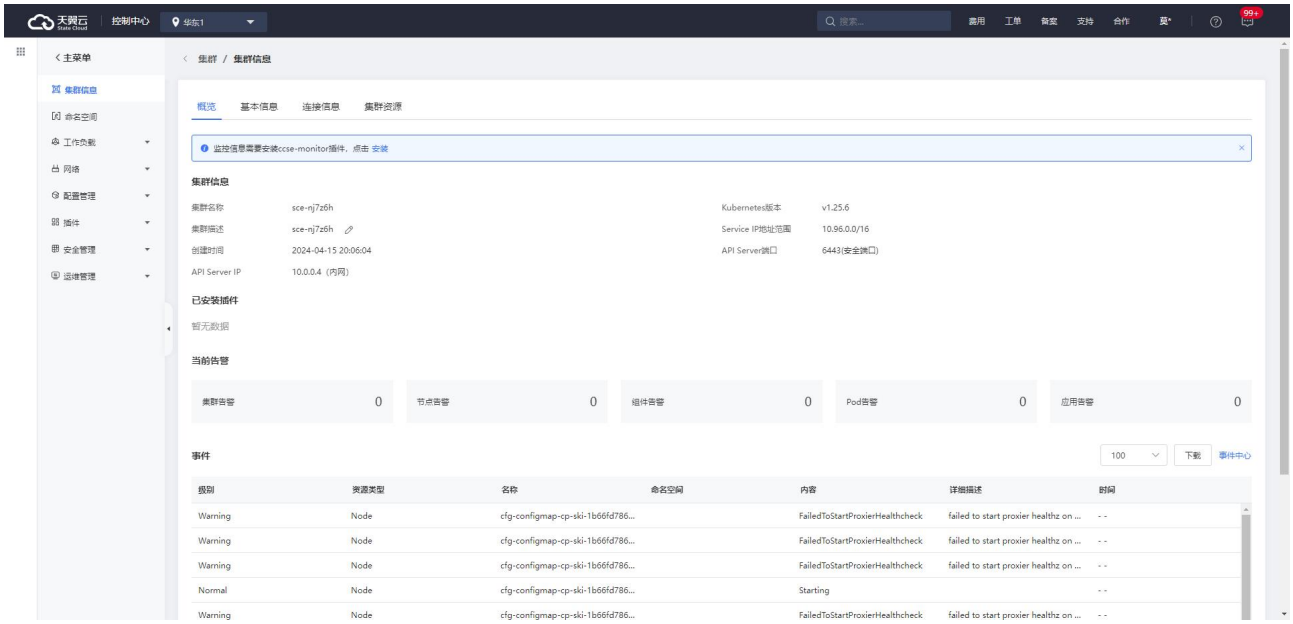
3.2.3 查看集群信息

SCE 集群提供了细致全面的集群概览展示页，其中包括组件状态、集群资源使用情况等功能板块，可以快速准确地了解集群的健康状态，并提供了集群的基本信息、连接信息、集群资源等相关信息。

查看集群概览

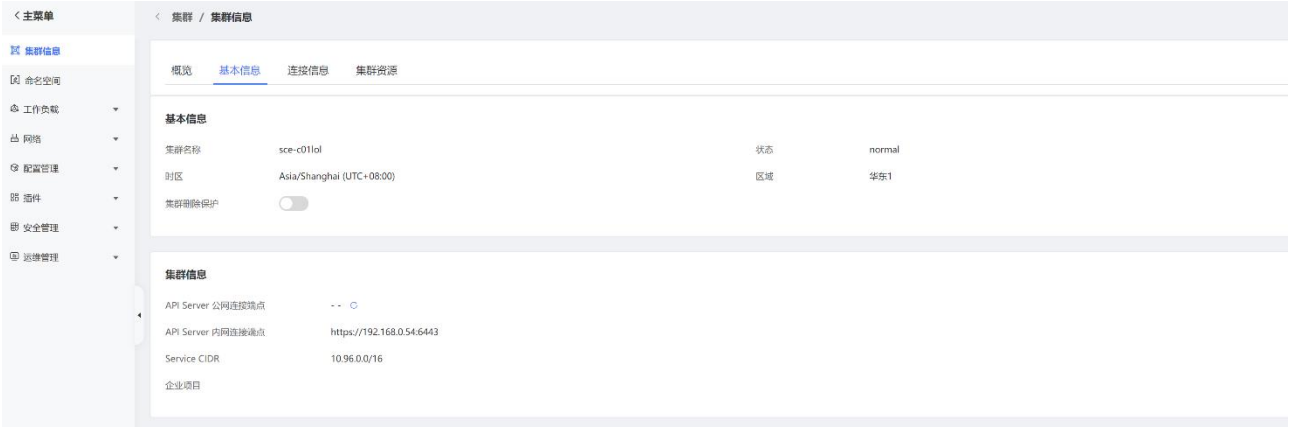
集群概览页面内查看组件状态、集群资源使用情况等信息。具体操作步骤如下：

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群” 。
3. 在集群列表页面中，单击目标集群的名称进入集群详情界面。
4. 单击概览页签，进去集群概览页面，查看集群信息、已安装插件、当前告警、事件等信息。
   a) 集群信息： 显示当前集群包括集群名称、Kubernetes 版本、创建时间、API Server 端口以及 API Server IP 等基础信息。
   b) 已安装插件： 展示集群当前已经安装的插件和插件状态。
   c) 当前告警： 自动扫描集群，提供集群告警、节点告警以及组件告警等信息， 及时提示您存在的潜在风险以便避免业务受损。
   d) 事件： 显示当前集群的日志信息。



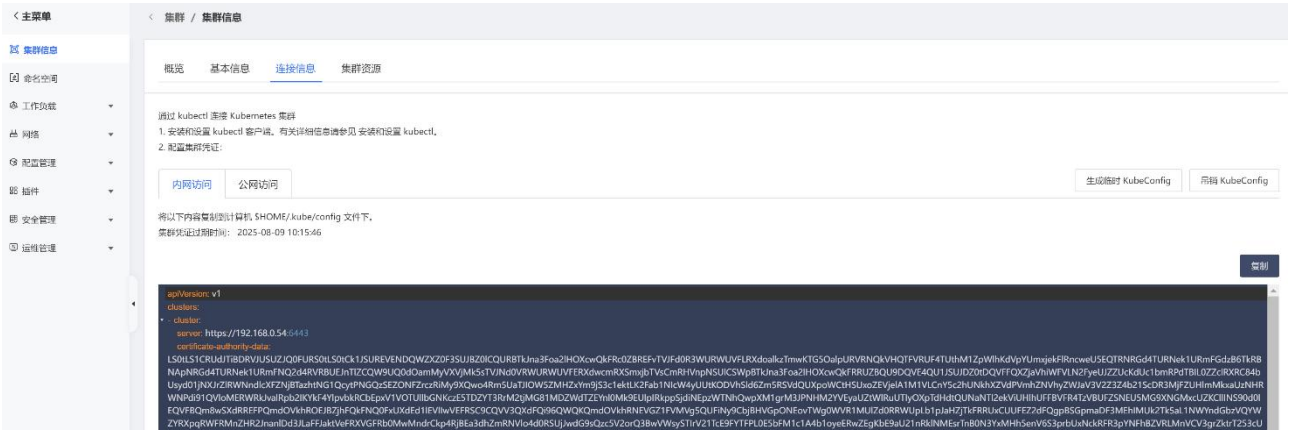
查看基本信息

单击基本信息页签，可以查看集群名称、状态、时区、区域以及是否开启集群删除保护等基本信息。同时可以查看 API Server 公网连接端点、API Server 内网连接端点以及 Service CIDR 等集群信息。



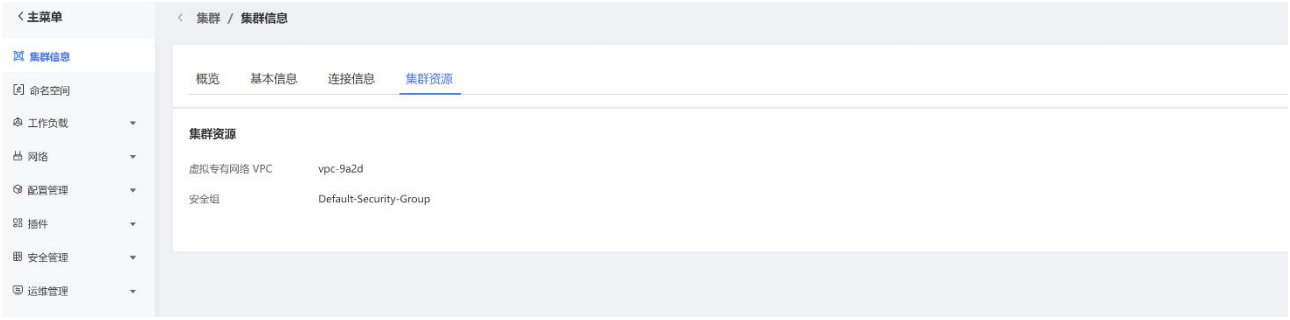
查看连接信息

单击连接信息页签，可以获取公网以及内网环境下的 KubeConfig 文件的配置内容，用于配置 Kubectl 客户端对集群的访问。



查看集群资源

单击集群资源页签，可以查看集群的虚拟专有网络 VPC 以及安全等信息。



3.2.4 删除集群

通过控制台可以方便退订不再需要的集群，避免资源浪费。具体操作步骤如下:

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群” 。
3. 在集群列表页面中，在目标集群选项卡中点击“退订” ，对集群实例进行退订操作。

3.2.5 管理和访问集群

3.2.5.1 通过 kubectl 连接 Kubernetes 集群

前提条件

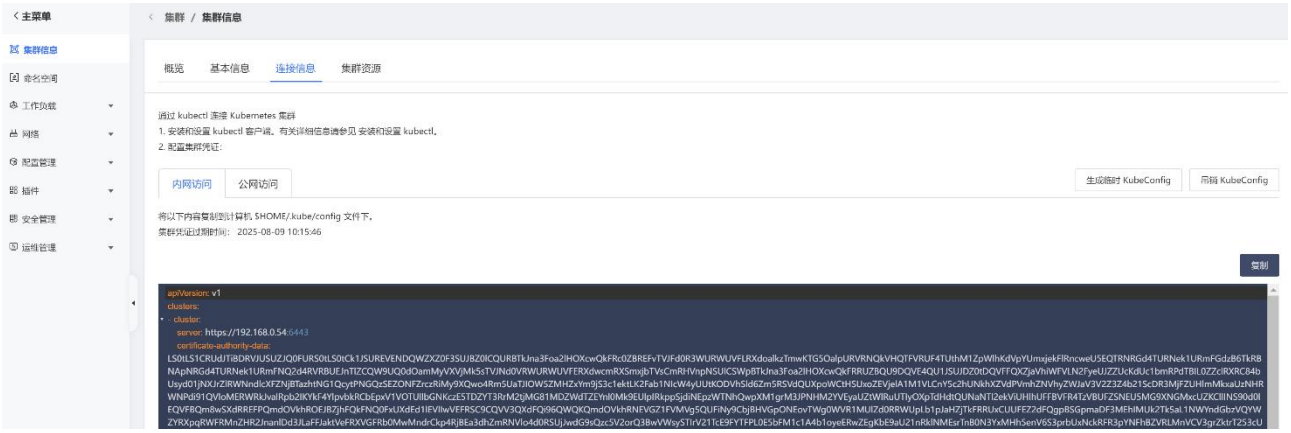
- 已经下载并安装最新的 kubectl 客户端。
- 完成 kubectl 的相关配置。

操作步骤

1. 登录云容器引擎控制台，在控制台的左侧导航栏中点击“集群” 。

2. 在集群列表页面中单击目标集群的名称 。

3. 在集群详情界面中选择连接信息页签，复制集群凭据到本地文件中。您可以在$HOME/.kube/config（kubectl 默认寻找凭据的路径）下创建并保存集群凭据。



4. 执行以下命令，确认集群连接情况。

```
kubectl get namespace
```

预期输出：

```
NAME                   STATUS     AGE
default                Active     5d5h
kube-node-lease        Active     5d5h
kube-public            Active     5d5h
kube-system            Active     5d5h
```

5. 配置完成后，您可以通过 kubectl 命令行工具，从本地计算机访问 Kubernetes 集群，实现对集群的管理和操作。

3.2.5.2 控制集群 API Server 的公网访问能力

您可以选择通过弹性公网 IP 暴露 Kubernetes 集群的 API Server。开启后，SCE 将为内网 ELB 实例绑定一个 EIP，获得从公网访问集群 API Server 的能力。

操作步骤

您可以在创建集群时绑定弹性公网 IP。

1. 登录云容器引擎控制台，在左侧菜单栏选择“集群”。
2. 在集群列表页面中，单击页面右上角的“创建 SCE 集群”，进入订购 Serverless 容器引擎界面。
3. 在“API Server 访问”配置项中勾选“使用 EIP 暴露 API Server”，系统默认为您创建 50Mbps 独享带宽的 EIP 并绑定，若您想修改带宽大小，可进入 EIP 控制台对已创建的 EIP 进行修改。

3.3 应用

3.3.1 通过命令行管理应用

前提条件

在本地使用命令之前，需要先通过 kubectl 连接 Kubernetes 集群。

通过命令创建应用

可以通过以下命令直接运行一个简单的 Nginx 容器。

```
kubectl run nginx-pod --image=nginx
```

如果需要在集群内部暴露相关服务，可以通过以下命令进行操作：

```
kubectl expose pod nginx-pod --name=nginx-service --port=8080 --target-port=80 --type=ClusterIP
```

通过命令查看容器

以下命令用于获取 Kubernetes 集群中所有运行的 Pod 对象的状态信息。

```
kubectl get pods
```

预期输出为：

```
NAME                        READY       STATUS      RESTARTS     AGE
nginx-pod                   1/1         Running     1            9h
```

以下命令用于获取 Kubernetes 集群中所有 Service 对象的状态信息。

```
kubectl get svc
```

预期输出为：

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
nginx-service	ClusterIP	10.233.10.115	<none>	8080/TCP	9h

3.3.2 使用镜像创建应用

前提条件

确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。

步骤一：配置应用基本信息

1. 在集群管理页面的左侧导航栏中，选择“工作负载” 。
2. 点击“无状态” ，在无状态页面中，单击“创建 Deployment” 。
3. 进入创建 Deployment 页面，设置应用的基本信息。

步骤二：配置容器

在实例内容器中，配置容器的名称、镜像、类型以及资源等。

说明：在实例内容器的页签中，可以单击上方的“+添加容器”来为您的应用创建多个容器。

1. 设置容器的基本信息。

配置项	描述

容器名称	填写容器的名称
镜像及镜像版本	支持在容器镜像服务或开源镜像中选择镜像以及镜像版本
容器类型（选填)	支持初始容器和工作容器
挂载点（选填)	支持挂载数据卷到容器内的指定路径
镜像拉取策略(选填)	支持 IfNotPresent、Always、Never
CPU/ 内存限制	Request 用于预分配资源，当集群中的节点没有 request 所要求的资源数量时，容器会创建失败。Limit 用于设置容器使用资源的最大上限，避免异常情况下节点资源消耗过多
环境变量（选填)	可以通过键值对的方式为 Pod 配置环境变量，这可以用于为 Pod 添加环境标志或传递配置信息： 类型：环境变量的类型包括 keyValue、fieldRef、configMapKeyRef 以及 secretKeyRef。其中 configMapRef 和 secretKeyRef 支持全部文件的引用；filedRef 目前仅支持 podIP 以及 nodeName。 自定义、配置项、保密字典、变量/变量引用和资源引用。配置项和保密字典支持对全部文件的引用。以保密字典为例，选择变量时，默认会引用整个 Secret 变量名：填写环境变量名称 变量值：填写变量引用的值
启动执行（选填)	对应镜像的 ENTRYPOINT 命令，将会覆盖镜像的 ENTRYPOINT 命令；每个输入框仅输入一个命令或参数 启

	动执行-参数：对应镜像的 CMD 命令，将会覆盖镜像的 CMD 命令；每个输入框仅输入一个命令或参数
启动后处理(选填)	容器启动后执行，注意由于是异步执行，无法保证一定在 ENTRYPOINT 之后运行；每个输入框仅输入一个命令或参数
停止前处理(选填)	容器停止前执行，常用于资源清理；每个输入框仅输入一个命令或参数
容器健康检查(选填)	存活检查：检查容器是否正常，不正常则重启实例 就绪检查：检查容器是否就绪，不就绪则停止转发流量到当前实例
集群删除保护	设置是否启用集群删除保护，防止通过控制台或者 API 误删除集群
集群本地域名	填写集群的本地域名

2. 访问设置 。开启 Service 选项后，可以设置暴露后端应用的方式。

说明：根据应用的实际需求，您可以根据以下方式进行服务访问方式设置：

- 虚拟集群 IP ：这适用于只在集群内部工作的应用，方便应用之间进行内部通信。
- 负载均衡 ：对于需要暴露到公网的应用，您可以采用负载均衡类型的服务，并通过天翼云提供的负载均衡服务 ELB，使得该服务获得公网访问能力。

配置项	描述
服务访问方式	目前支持虚拟集群 IP 和负载均衡两种服务访问方式： 虚拟集群 IP ：即 ClusterIP，是一种通过集群内部 IP 暴露服务的方式。选择这个值意味着服务只能在集群内部访问 负载均衡 ：即 LoadBalancer，通过天翼云 ELB 提供服务支持，可以根据实际需要选择公网访问或者私网访问。支持使用已有 ELB 或者新建 ELB
注解	为服务添加注解，即 Annotation
端口映射	支持指定协议、容器端口以及服务端口。确保容器端口与后端 Pod 中暴露的容器端口一致

步骤三：查看应用

1. 本例中，镜像选择 nginx。在访问设置中，协议选择 TCP，服务端口和容器端口分别配置为 30002 和 80。
2. 创建完成后，进入工作负载的无状态页面，可以看到新建的 nginx 应用出现在无状态列表下。

3. 在群管理页面的左侧导航栏中，选择“网络”下的“服务”页面，可以看到新建的 nginx 服务出现在服务列表下。

3.3.3 创建服务

前提条件

确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。

背景信息

Kubernetes 中每一个工作负载会有一个或多个实例（Pod），每个实例（Pod）的 IP 地址由网络插件动态随机分配（Pod 重启后 IP 地址会改变）。为屏蔽这些后端实例的动态变化和对多实例的负载均衡，引入了 Service 这个资源对象。Service 是一种资源，提供了我们访问单个或多个容器应用的

能力。每个服务在其生命周期内，都拥有一个固定的 IP 地址和端口。每个服务对应了后台的一个或多个 Pod，通过这种方式，客户端就不需要关心 Pod 所在的位置，方便后端进行 Pod 的扩容、缩容等操作，更多详细原理可参阅 Kubernetes 官网的 Service 部分。

步骤一：创建 Deplyoment

使用镜像创建一个 Deployment，详细操作步骤请参阅使用镜像创建应用 。

步骤二：创建服务

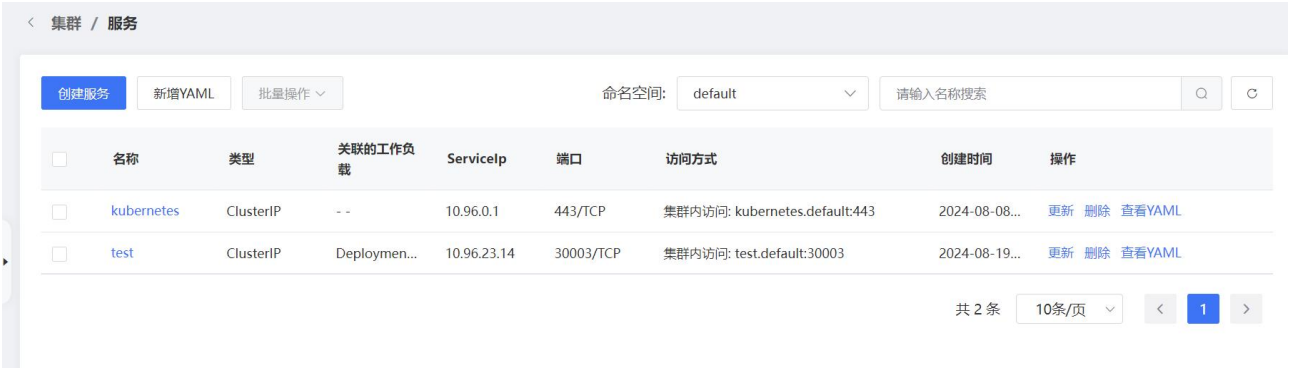
1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群”。
3. 在集群列表页面中，点击目标集群的名称进入集群详情页面。
4. 点击左侧导航栏中的“网络” ，并选择“服务” 。
5. 在服务页面单击左上角的“创建服务”。
6. 进行相关参数的配置：

    a. 填写服务的基本信息：

b. 填写访问设置信息。包括填写端口映射的名称、容器端口、服务端口以及选择协议类型为 TCP 或者 UDP。

3. 进行工作负载绑定，选择在步骤一中创建的 Deployoment 进行绑定。

7. 创建完成后，您可以在服务页面下对已有服务进行更新、删除以及查看 YAML 等操作。

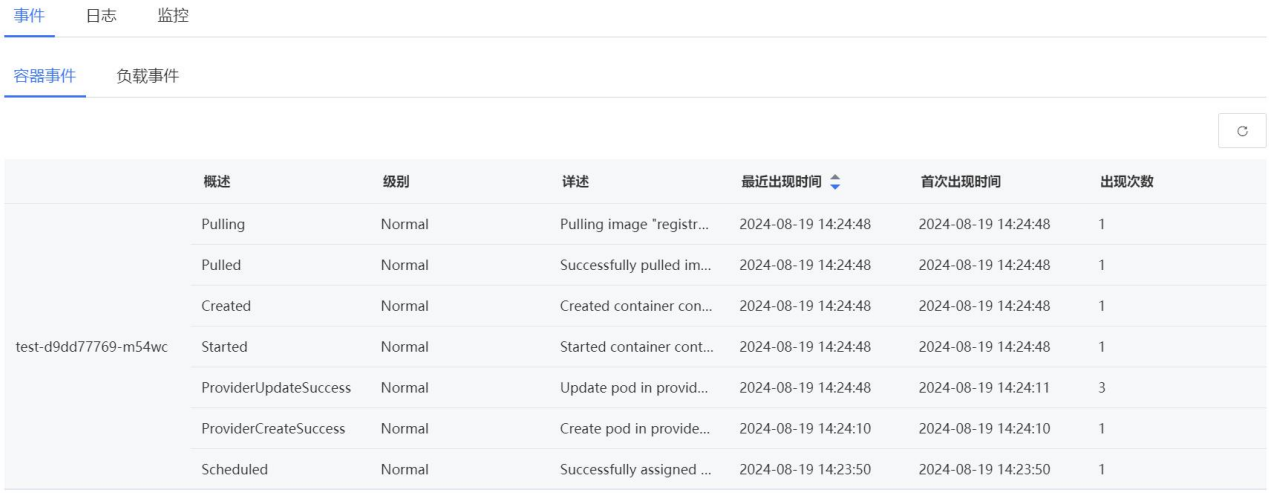


3.3.4 查看容器

操作步骤

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群” 。
3. 在集群管理页面的左侧导航栏中，选择“工作负载”。
4. 点击“容器组” ，在容器组页面下可以查看所有创建的容器。

5. 点击想要查看的实例名称 ，即可进入查看该容器组的详细信息。包括查看容器组的事件、日志以及监控等。



	概述	级别	详述	最近出现时间	首次出现时间	出现次数
test-d9dd77769-m54wc	Pulling	Normal	Pulling image "registr...	2024-08-19 14:24:48	2024-08-19 14:24:48	1
	Pulled	Normal	Successfully pulled im...	2024-08-19 14:24:48	2024-08-19 14:24:48	1
	Created	Normal	Created container con...	2024-08-19 14:24:48	2024-08-19 14:24:48	1
	Started	Normal	Started container cont...	2024-08-19 14:24:48	2024-08-19 14:24:48	1
	ProviderUpdateSuccess	Normal	Update pod in provid...	2024-08-19 14:24:48	2024-08-19 14:24:11	3
	ProviderCreateSuccess	Normal	Create pod in provide...	2024-08-19 14:24:10	2024-08-19 14:24:10	1
	Scheduled	Normal	Successfully assigned ...	2024-08-19 14:23:50	2024-08-19 14:23:50	1

3.4 配置项及密钥

3.4.1 管理配置项

前提条件

确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。

创建配置项

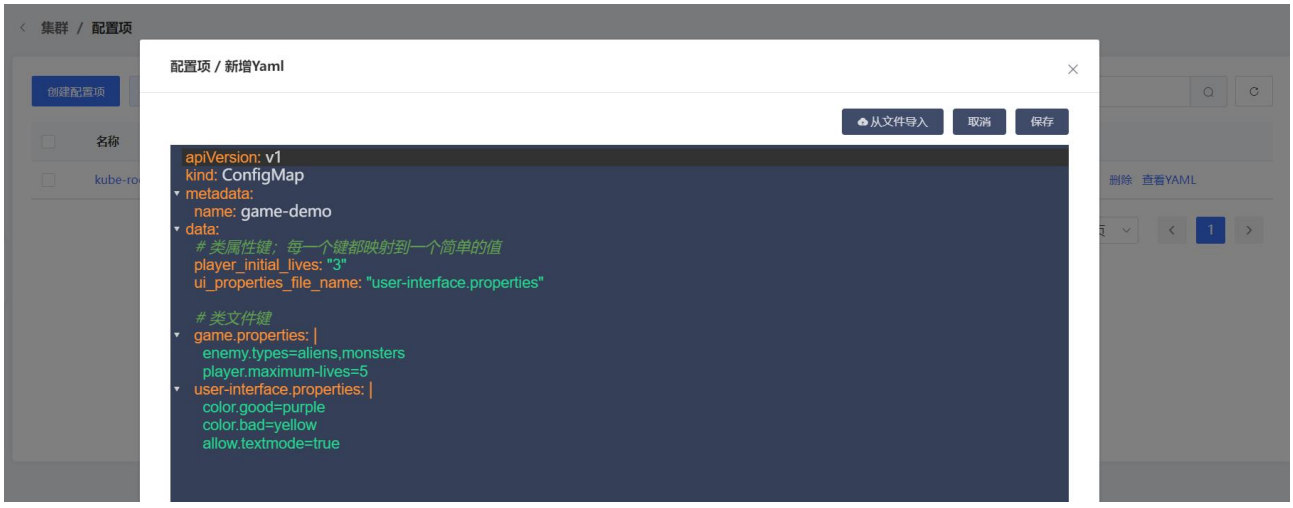
1. 登录云容器引擎控制台，在左侧导航栏中选择“集群”。
2. 在集群列表页面中，单击目标集群名称，并在左侧导航栏中选择“配置管理”。
3. 选择“配置项”，在配置项页面中，您可以通过以下两种方式创建配置项。
   a. 通过配置项菜单创建。
      i. 单击配置项页面左上角的“创建配置项”。

ii. 在配置项新增页面中，填写配置项名称以及配置项内容，包括变量名和对应的变量值，允许添加多个配置项。同时支持从文件导入配置项内容。

b. YAML 创建资源。

i. 单击配置项页面左上角的“新增 YAML”。

ii. 在使用模版部署的页面填写配置项内容，即 ConfigMap 的相关信息；或者选择“从文件导入” ，然后单击保存 。



相关操作

配置项创建完成后，您可以在配置项页面中进行以下操作：

● 在目标配置项选项卡点击单击“查看与更新” ，可以查看、修改或删除该配置项信息。

● 在目标配置项选项卡单击“删除”，可以删除不需要的配置项。

● 在目标配置项选项卡单击“查看 YAML”，看查看该配置项对应的 YAML 文件。

3.4.2 在容器组中使用配置项

背景信息

在 Pod 中使用配置项是一种实现配置管理的常用方式，它可以应用于多种场景，主要包括以下几个方面：

● 应用程序配置：配置项可以存储应用程序所需的所有配置信息，例如数据库连接信息、密钥、证书等。这可以使得应用程序在部署时更加简单和灵活，开发人员可以使用配置项来控制应用程序的行为。

● 环境变量：通过配置项创建的环境变量可以直接注入到容器中，例如在容器中设置数据库的连接信息等环境变量。

● 命令行参数：通过配置项定义命令行参数可以使得容器镜像更加通用，可以使用不同的参数启动不同的容器实例。

● 挂载文件：配置项还可以将配置文件存储在其中，并将其挂载到容器内部。这使得容器可以在运行时动态加载配置文件，以更好地适应不同的部署场景。

使用限制

当您在 Pod 中使用配置项时，为了确保配置项在 Pod 内正确加载和使用，需要将它们部署到相同的命名空间中。

创建配置项

本次示例配置项 configmap-demo 包含 DATABASE_URL 和 LOG_LEVEL 两个键值对。具体的 YAML 示例模板如下所示：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-demo
  namespace: default
data:
  DATABASE_URL: mysql://user:password@hostname/dbname
  API_SECRET: big_secret_key
  LOG_LEVEL: debug
```

使用配置项定义 Pod 环境变量

1. 使用配置项的数据定义 Pod 环境变量。

   a. 登录云容器引擎控制台。

   b. 在控制台的左侧导航栏中点击“集群”。

   c. 在集群列表页面中，点击目标集群的名称进入集群详情页面。

   d. 点击左侧导航栏中的“工作负载 ”，并选择“无状态”。

   e. 在无状态页面中，单击左上角的“新增 YAML”。

f. 填写无状态应用对应的 YAML 配置内容，并使用 valueFrom 引用配置项中的 Value 值，从而定义 Pod 的环境变量。
下面是一个编排示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  namespace: default
  labels:
    app: busybox
    name: busybox
    source: SCE
spec:
  replicas: 3
  selector:
    matchLabels:
      app: busybox
      name: busybox
  template:
    metadata:
      labels:
        app: busybox
        name: busybox
```

```
        source: SCE
    spec:
      containers:
        - name: busybox
          image: busybox:latest
          ports:
            - containerPort: 8080
          env:
            - name: DATABASE_URL
              valueFrom:
                configMapKeyRef:
                  name: configmap-demo
                  key: DATABASE_URL
            - name: LOG_LEVEL
              valueFrom:
                configMapKeyRef:
                  name: configmap-demo
                  key: LOG_LEVEL
```

2. 将配置项的所有 Key/Values 配置为 Pod 的环境变量。

a. 登录云容器引擎控制台。

b. 在控制台的左侧导航栏中点击“集群 ”。

c. 在集群列表页面中，点击目标集群的名称进入集群详情页面。

d. 点击左侧导航栏中的“工作负载” ，并选择“无状态”。

e. 在无状态页面中，单击左上角的“新增 YAML” 。

f. 填写无状态应用相应的 YAML 配置内容，并使用 envFrom 将配置项的所有 Key/Values 键值对配置为 Pod 的环境变量。

下面是一个编排示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  namespace: default
  labels:
    app: busybox
    name: busybox
    source: SCE
spec:
  replicas: 3
  selector:
    matchLabels:
      app: busybox
      name: busybox
  template:
    metadata:
```

```
      labels:
        app: busybox
        name: busybox
        source: SCE
    spec:
      containers:
        - name: busybox
          image: busybox:latest
          envFrom:
            - configMapRef:
                name: configmap-demo
```

3. 通过配置项设置命令行参数。

a. 登录云容器引擎控制台。

b. 在控制台的左侧导航栏中点击“集群 ”。

c. 在集群列表页面中，点击目标集群的名称进入集群详情页面。

d. 点击左侧导航栏中的“工作负载” ，并选择“无状态”。

e. 在无状态页面中，单击左上角的“新增 YAML” 。

f. 填写无状态应用相应的 YAML 配置内容，您可以使用环境变量替换语法 $(VAR_NAME)，将配置项设置为容器中的命令或参数值。

下面是一个编排示例：

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: busybox
  namespace: default
  labels:
    app: busybox
    name: busybox
    source: SCE
spec:
  replicas: 3
  selector:
    matchLabels:
      app: busybox
      name: busybox
  template:
    metadata:
      labels:
        app: busybox
        name: busybox
        source: SCE
    spec:
      containers:
        - name: busybox
          image: busybox:latest
```

```
        command: [ "/bin/sh", "-c", "echo $(DATABASE_URL) $(LOG_LEVEL)" ]
        envFrom:
          - configMapRef:
              name: configmap-demo
```

在数据卷中使用配置项

1. 登录云容器引擎控制台。

2. 在控制台的左侧导航栏中点击“集群 ”。

3. 在集群列表页面中，点击目标集群的名称进入集群详情页面。

4. 点击左侧导航栏中的“工作负载” ，并选择“无状态”。

5. 在无状态页面中，单击左上角的“新增 YAML”。

6. 您可以在数据卷中引用配置项。在 volumes 中指定要使用的配置项名称，并设置挂载路径 mountPath。配置项中的键值对数据将被存储在指定的挂载路径下（例如 /etc/config），每个键值对作为一个文件，文件名是键的名称，文件内容是该键的值。

下面是一个编排示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  namespace: default
  labels:
```

```
    app: busybox
    name: busybox
    source: SCE
spec:
  replicas: 3
  selector:
    matchLabels:
      app: busybox
      name: busybox
  template:
    metadata:
      labels:
        app: busybox
        name: busybox
        source: SCE
    spec:
      volumes:
        - name: config-volume
          configMap:
            name: configmap-demo
```

```
    containers:
      - name: busybox
        image: busybox:latest
        volumeMounts:
          - name: config-volume
            mountPath: /etc/config
```

3.4.3 管理保密字典

前提条件

确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。

背景信息

在 Kubernetes 中，保密字典（Secret）是一种用于保存敏感数据的对象。它可以存储像用户名、密码、令牌、密钥、证书等敏感信息，这些信息需要被保护以避免被恶意用户利用。

- 保密字典可以以字符串或者文件的形式保存敏感信息，它们被编码为 base64 格式并存储在 Kubernetes 集群中。
- 保密字典可以被用于各种用例，例如：在容器中或 Pod 中挂载配置文件、合并 Docker 镜像的安全证书等。
- 保密字典的使用方式类似于配置项目，但保密字典显然更加安全，因为它可以加密保存，并且可以限制访问权限。您可以在 Pod 中使用 Volume 或者环境变量引用保密字典，或者将其用作镜像源、终端服务器等。

创建保密字典

1. 登录云容器引擎控制台，在左侧导航栏中选择“集群”。

2. 在集群列表页面中，单击目标集群名称，并在左侧导航栏中选择“配置管理” 。

3. 选择“保密字典”，在保密字典页面中，您可以通过以下两种方式创建配置项。

   a. 通过保密字典菜单创建。

      i. 单击保密字典页面左上角的“创建” 。

      ii. 在创建保密字典页面中，填写保密字典名称。名称最长 100 个字符，由小写字母、数字、"-"及"."组成，且开始和结尾只能是数字和字母。

      iii.填写保密字典内容。包括变量名和对应变量值，允许添加多个配置项。支持“上传文本文件”和“上传二进制文件” 。

   b. 使用 YAML 创建。

      i. 单击保密字典页面左上角的“新增 YAML”。

      ii. 在使用模版部署的页面填写保密字典内容，即 Secret 的相关信息；或者选择从文件导入然后单击保存。

相关操作

保密字典创建完成后，您可以在保密字典页面中进行以下操作：

● 在目标保密字典选项卡单击“查看与更新” ，可以查看、修改或删除保密字典信息。

● 在目标保密字典选项卡单击“删除” ，可以删除不需要的保密字典。

● 在目标保密字典选项卡单击“查看 YAML” ，看查看该保密字典对应的 YAML 文件。

3.4.4 在容器组中使用保密字典

如果需要在 Kubernetes 集群中使用密码、令牌、密钥、证书等敏感信息时，推荐使用保密字典。本文将介绍如何在控制台中创建保密字典（Secret），并使用保密字典配置 Pod 数据卷及环境变量。

前提条件

● 在容器组（Pod）中使用保密字典时，确保两者处于同一个集群和 Namespace 中。

● 已经连接到集群中的 Master 节点，具体操作请参阅通过 kubectl 连接。

创建保密字典

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群” 。
3. 在集群列表页面中，点击目标集群的名称进入集群详情页面。
4. 点击左侧导航栏中的“配置管理” ，并选择保密字典 。

5. 在保密字典页面中，单击左上角的“新增 YAML”。

6. 在弹窗中自定义保密字典的 YAML 配置内容。

具体的 YAML 示例模板如下所示：

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-demo
  namespace: default
type: Opaque
data:
  username: dXNlcm5hbWU= # base64 编码的用户名
  password: cGFzc3dvcmQ= # base64 编码的密码
```

使用保密字典配置 Pod 数据卷

1. 创建 example.yaml 配置文件，具体示例内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  namespace: default
  labels:
    app: busybox
```

```
    name: busybox
    source: SCE
spec:
  replicas: 3
  selector:
    matchLabels:
      app: busybox
      name: busybox
  template:
    metadata:
      labels:
        app: busybox
        name: busybox
        source: SCE
    spec:
      volumes:
        - name: secret-volume
          secret:
            name: secret-demo
      containers:
        - name: busybox
          image: busybox:latest
          volumeMounts:
```

```
      - name: secret-volume
        mountPath: /mnt/secret
```

2. 执行以下命令，配置保密字典。

```
kubectl apply -f example.yaml
```

在上述示例中，我们定义了一个 Pod，并在其中添加了一个名为 my-secret-volume 的 Volume，其类型为 Secret，并将其挂载到名为 my-container 的容器中。在容器中，我们将 Volume 挂载到了/mnt/secret 路径下。此时，我们就可以在容器中使用 /mnt/secret/username 和 /mnt/secret/password 两个文件，来获取 Secret 中的用户名和密码信息。

使用保密字典设置 Pod 的环境变量

1. 通过命令行进行配置。

创建 example.yaml 配置文件，具体示例内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: busybox
  namespace: default
  labels:
    app: busybox
    name: busybox
    source: SCE
spec:
```

```
replicas: 3
selector:
  matchLabels:
    app: busybox
    name: busybox
template:
  metadata:
    labels:
      app: busybox
      name: busybox
      source: SCE
  spec:
    containers:
      - name: busybox
        image: busybox:latest
        ports:
          - containerPort: 8080
        env:
          - name: username
            valueFrom:
              secretKeyRef:
                name: secret-demo
                key: username
```

```
      - name: password
        valueFrom:
          secretKeyRef:
            name: secret-demo
            key: password
```

执行以下命令，配置保密字典。

```
kubectl apply -f example.yaml
```

在上述示例中，我们定义了一个 Pod，并在其中添加了一个名为 my-container 容器，接着为其定义了两个环境变量 MY_USERNAME 和 MY_PASSWORD，并使用 valueFrom 和 secretKeyRef 来引用 Secret 中的值。secretKeyRef 的 name 属性用于指定 Secret 的名称，key 属性用于指定需要使用的 Secret 中对应的键名。通过上述方式，在容器中就可以使用 MY_USERNAME 和 MY_PASSWORD 环境变量，来获取 Secret 中的用户名和密码信息。

2. 通过控制台进行配置。

a. 登录云容器引擎控制台。

b. 在控制台的左侧导航栏中点击“集群” 。

c. 在集群列表页面中，点击目标集群的名称进入集群详情页面。

d. 点击左侧导航栏中的“工作负载” ，并选择“无状态” 。

e. 在无状态页面中，单击左上角的“创建 deployment” 。详细操作步骤请参阅 Serverless 容器引擎 SCE 使用快速入门 。

f. 在“数据卷（选填）”中添加目标保密字典的数据卷。

g. 在实例内容器中的环境变量区域单击“新增变量”，类型选择“secretKeyRef”，变量/变量引用选择在创建保密字典中新建的 Secret，再分别选择 Secret 的 Key 以及填写它们对应的变量名。
本次示例配置如下所示：

3.4.5 Service 管理

创建服务

Kubernetes 中每一个工作负载会有一个或多个实例（Pod），每个实例（Pod）的 IP 地址由网络插件动态随机分配（Pod 重启后 IP 地址会改变）。为屏蔽这些后端实例的动态变化和对多实例的负载均衡，引入了服务（Service）这个资源对象。本文将介绍如何创建服务并对外发布应用。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 在本地使用命令之前，需要先通过 kubectl 连接 Kubernetes 集群。

通过命令创建应用

步骤一：创建 Deployment

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群”。
3. 在集群列表页面中，单击目标集群的名称进入集群详情界面。
4. 在集群管理页面的左侧导航栏中，选择“工作负载” ，然后单击“无状态”。
5. 在无状态页面中单击左上角的“新增 YAML” ，本次示例模板是一个 Nginx 的 Deployment，具体内容如下所示：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
  labels:
    app: nginx
    name: nginx
    source: SCE
spec:
  replicas: 1
  selector:
    matchLabels:
```

```
        app: nginx
        name: nginx
  template:
    metadata:
      labels:
        app: nginx
        name: nginx
        source: SCE
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
```

6. 创建完成后可查看该应用。

a. 在集群管理页面的左侧导航栏中，选择“工作负载” ，然后单击“无状态” 。

b. 在无状态页面中可以查看所有已经创建的 Deployment。

c. 在目标 Deployment 项选项卡单击创建好的应用名称 ，查看其详情。

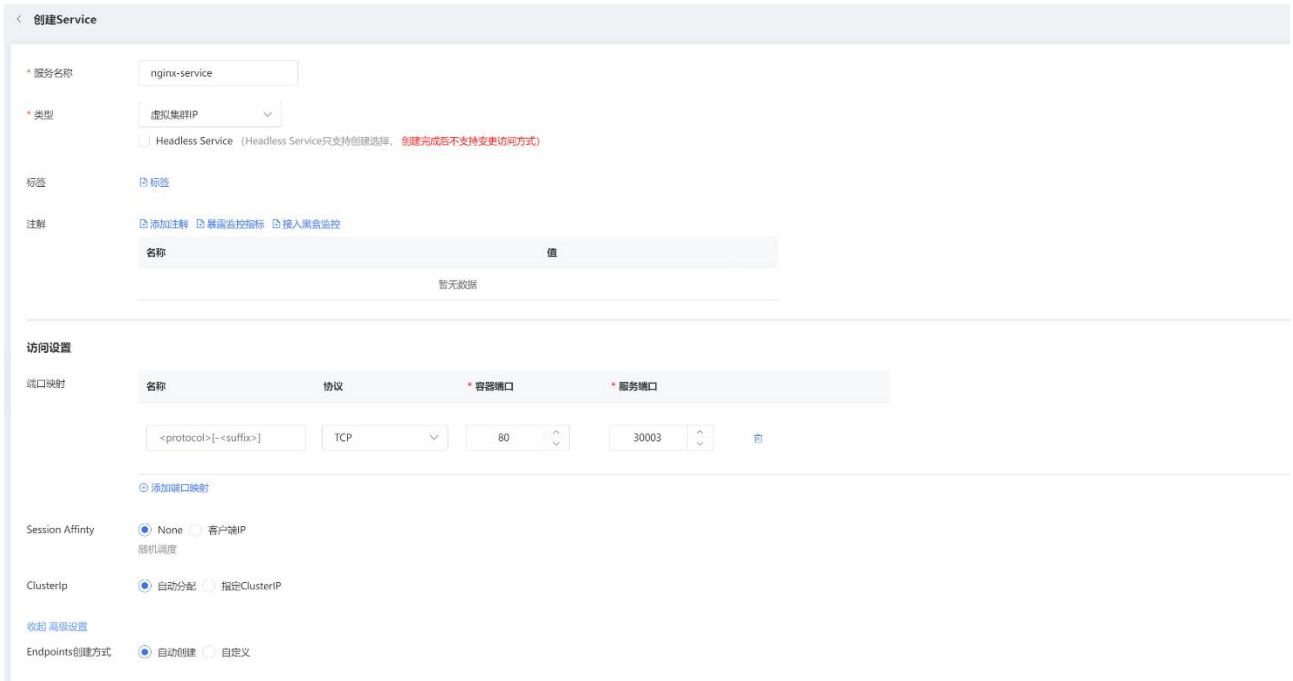
步骤二：创建服务

1. 在集群列表页面中，单击目标集群的名称进入集群详情界面。

2. 在集群管理页面的左侧导航栏中，选择“网络”，然后单击“服务”。

3. 单击左上角的“创建服务” ，填写基本信息。

4. 配置访问信息，选择协议类型 TCP 或者 UDP，容器端口是业务容器对外暴露的端口，服务端口是服务对外提供访问的端口，同时支持 Session 粘连，并且可以指定集群内部的服务 IP 地址。



5. 在工作负载绑定区域内，选择上一步中创建的无状态应用。

6. 单击“提交”，完成服务的创建。在服务页面中可以查看所有创建的服务，可以对其进行更新 、删除以及查看 YAML 等操作。

3.5 网络

3.5.1 Nginx Ingress 管理

3.5.1.1 Nginx Ingress 概述

本文将介绍 Ingress 基本概念、Ingress Controller 工作原理以及 Nginx Ingress Controller 的使用说明。

Ingress 基本概念

Ingress 是 Kubernetes 集群中一种 API 对象，属于网络路由和负载均衡中的一个概念。它的主要作用是将外部的流量路由到集群内部的服务，您可以通过 Ingress 资源来配置不同的转发规则，从而根据不同的规则设置访问集群内不同的 Service 所对应的后端 Pod。Ingress 可以看作是在 Service 的基础上提供了更高级别的负载均衡和路由规则控制。与 Service 不同的是，Service 只提供了基础的负载均衡和服务发现功能

Ingress Controller 工作原理

Ingress API 对象的创建需要使用 Ingress Controller，它是一个独立于 Kubernetes 集群之外的组件。Ingress Controller 会不断地监视 Ingress 对象的变化，当检测到 Ingress 对象的变化时，它会自动更新代理服务器的配置信息。通常情况下，Ingress Controller 会将请求代理到运行在同一集群内的 Service 中，但它也可以被配置为代理到集群外部的其他服务。Ingress Controller 常见的工作流程如下：

1. Kubernetes 集群中的 Ingress Controller 通过 API Server 监控 Ingress 的创建、删除和更新。
2. 当有新的 Ingress 创建时，Ingress Controller 会解析 Ingress 的规则，并将其实现为代理服务器的转发规则。

3. 当有新的 Service 创建或修改时，Ingress Controller 会读取对应的 Service 的信息，例如其 IP 和端口等，并将其添加到代理服务器的配置中，实现对后端服务的指向。
4. 当新的 Pod 创建、修改或删除时，Ingress Controller 会读取每个 Pod 上的 Service 的信息，并将其添加到代理服务器的配置中，实现对后端服务的指向。
5. 当有 Ingress 被删除时，Ingress Controller 会删除代理服务器上对应的路由规则。

Nginx Ingress Controller 使用说明

目前，Kubernetes 官方维护的是 Nginx Ingress Controller。SCE 则在社区版的 Nginx Ingress Controller 基础上进行了优化。在创建 SCE 集群时，您选择安装的 Nginx Ingress 插件即为 SCE 定制版的 Nginx Ingress Controller 插件。

3.5.1.2 安装 Nginx Ingress Controller

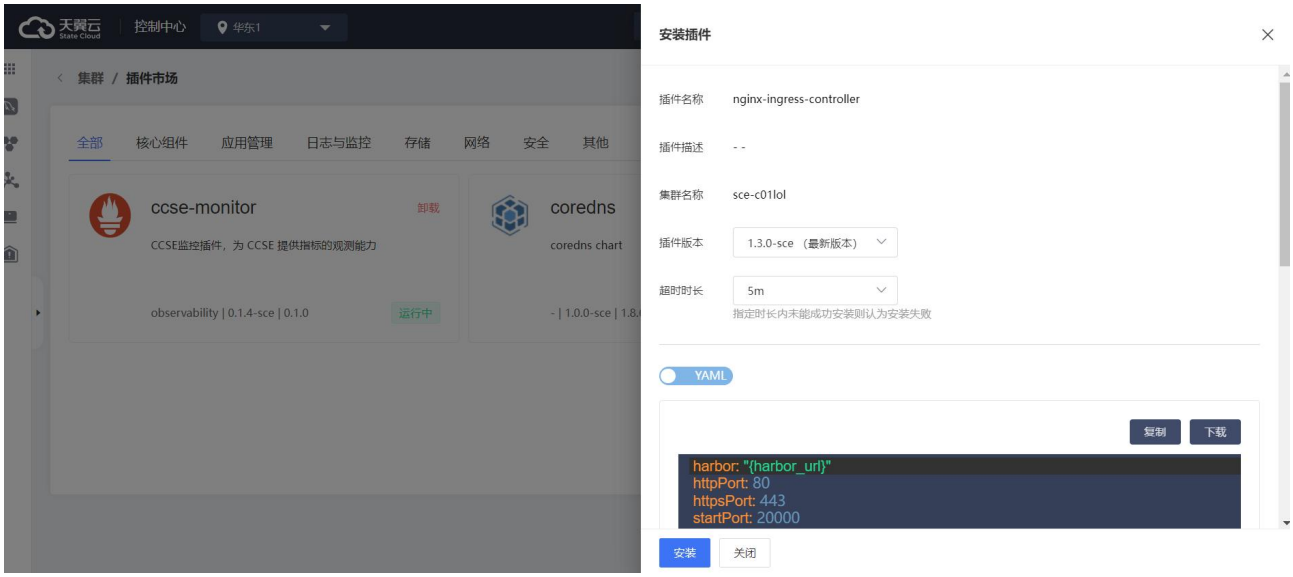
您可通过在插件管理页面安装 Nginx Ingress Controller，详细操作步骤如下：

操作步骤

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群”。
3. 在集群列表页面中，单击目标集群的名称进入集群详情界面。
4. 在集群管理页面的左侧导航栏中，选择“插件” ，然后单击“插件市场”。

5. 在插件市场页面找到“nginx-ingress-controller ”，点击“安装”。

6. 在弹出页中确定插件版 、超时时长以及 YAML 信息，单击“安装”。



7. 如果在 nginx-ingress-controller 插件区域的右上角显示已安装，则表示该插件已安装成功。

3.5.1.3 创建 Nginx Ingress

Ingress 是 Kubernetes 集群中一种 API 对象，属于网络路由和负载均衡中的一个概念。它的主要作用是将外部的流量路由到集群内部的服务，您可以通过 Ingress 资源来配置不同的转发规则，从而根据不同的规则设置访问集群内不同的 Service 所对应的后端 Pod。本文将介绍如何通过控制台和 Kubectl 方式创建、查看、更新和删除 Ingress。

前提条件

确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。

方式一：控制台操作指导

创建 Ingress

1. 登录云容器引擎控制台。

2. 在控制台的左侧导航栏中点击“集群”。
3. 在集群列表页面中，单击目标集群的名称进入集群详情界面。
4. 在集群管理页面的左侧导航栏中，选择“网络” ，然后单击“路由”。
5. 单击路由页面左上角的“创建路由” ，跳转到创建 Ingress 页面。创建路由时，整体上可以分为生产 Ingress 和灰度 Ingress 。
6. 创建灰度 Ingress ：
    a. 在流量切换方式中选择灰度 。
    b. 选择命名空间，创建灰度 Ingress 时必须关联一个生产 Ingress，用于实现生产 Ingress 和灰度 Ingress 之间的流量切换。
    c. 选择流量切换方式，可以选灰度或者蓝绿 。
    d. 选择灰度后，支持两种转发方式：Header 和 Cookie。
        i. Header ：根据 http 请求的头部中是否包含指定 key-value 来选择将请求转发到生产 Ingress 还是灰度 Ingress。
        ii. Cookie ：根据 http 请求的 Cookie 中是否包含指定 key-value 来选择将请求转发到生产 Ingress 还是灰度 Ingress，Cookie 转发时无法更改 value 和匹配方式。
    e. 选择蓝绿后，蓝绿转发可以选择转发到生产 Ingress 和灰度 Ingress 的流量权重：
        i. 全部切到生产：将请求全部转发到生产 Ingress。
        ii. 全部切到灰度：将请求全部转发到灰度 Ingress 。
        iii. 自定义百分比：如 30%表示将 30%的请求转发到灰度 Ingress，70%的请求转发到生产 Ingress。
7. 创建生产 Ingress ：
    a. 在灰度 Ingress 选项中选择否 。
    b. 填写域名路径规则，包括协议、域名、路径、服务名称以及服务端口等信息，支持添加多个规格。

管理 Ingress

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群”。
3. 在集群列表页面中，单击目标集群的名称进入集群详情界面。
4. 在集群管理页面的左侧导航栏中，选择“网络” ，然后单击“路由”。
5. 在路由详情页面中，可以在目标路由的操作列中进行更新、删除以及查看 YAML 等操作。
6. 在路由详情页面中可以查看所有已经创建路由，点击目标路由名称可进一步看到该路由详情。

方式二：Kubectl 操作指导

创建 Ingress

1. 创建 test-ingress.yaml 文件，示例内容如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-app
```

```
            image: nginx:latest
            ports:
              - name: http
                containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: my-app-service
spec:
  selector:
    app: my-app
  ports:
    - name: http
      port: 8080
      targetPort: 8080
  type: ClusterIP
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-app-ingress
spec:
  rules:
    - host: my-app.example.com
      http:
        paths:
```

```
      - path: /my-app/
        pathType: Prefix
        backend:
          service:
            name: my-app-service
            port:
              number: 8080
```

2. 执行以下命令对相关对象进行创建。

```
kubectl apply -f test-ingress.yaml
```

查看 Ingress

执行以下命令查看 Ingress。

```
kubectl get ingress
```

更新 Ingress

执行以下命令更新 Ingress。

```
kubectl edit ingress my-app-ingress
```

删除 Ingress

执行以下命令删除 Ingress。

```
kubectl delete ingress my-app-ingress
```

3.5.2 服务发现 DNS

DNS 使得 SCE 集群支持应用的服务发现，为集群内的工作负载提供域名解析服务方案。本文将介绍 Kubernetes 集群中 DNS 域名解析的原理以及 SCE 集群中的域名服务方案 CoreDNS。

注意事项

对于已经部署的 SCE 集群，如果后续安装了 coredns 插件，这将只对新部署的工作负载生效。那些在启动 DNS 服务发现之前创建的工作负载不会自动更新它们的 DNS 配置。

Kubernetes 集群中 DNS 域名解析原理

Kubernetes 集群中的 DNS 主要作用包括：

1. 为 Kubernetes 集群内的 Pod 和 Service 提供解析域名的能力。
2. 将 Service 名称解析为 ClusterIP。
3. 将外部名称解析为 Service 的 EXTERNAL-IP。

在创建 SCE 集群中的工作负载时，默认生成的 /etc/resolv.conf 文件包含特定配置。这个文件定义了 DNS 服务器的地址、搜索域以及其他一些配置参数。

```
# search 定义了一个或多个域名搜索列表，当进行 DNS 查询时，这些域名会按顺序添加到仅具有部分域名的查询中
search kube-system.svc.cluster.local svc.cluster.local cluster.local
# nameserver 用于解析域名的 DNS 服务器的 IP 地址
nameserver 10.96.0.10
# options 设置一系列的解析选项
options ndots:5
```

CoreDNS 方案

CoreDNS 是 Kubernetes 集群中的 DNS 服务器，是解析 Kubernetes 集群内部和外部域名的核心组件，能够实现域名解析和服务发现功能。CoreDNS 具备强大的插件集，支持自定义 DNS 解析、自定义 hosts 映射、CNAME、Rewrite 等功能。

CoreDNS 为 Kubernetes 集群的 DNS 提供一个可插拔的插件框架，并同时支持 DNS-over-TLS 和 DNS-over-HTTPS，具有插件编写简单、配置灵活、易于扩展等特点。

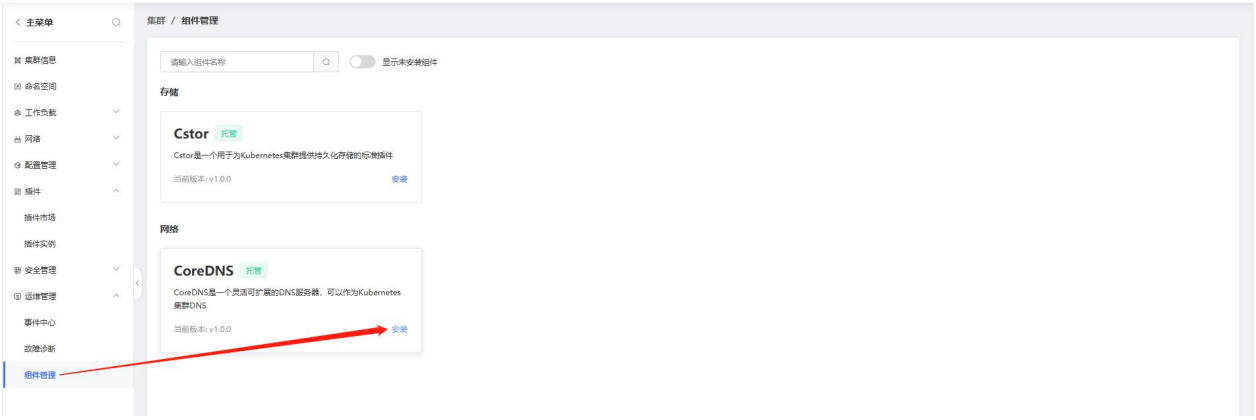
除此之外，CoreDNS 还内置了多种 DNS 解析插件，能够支持对 Kubernetes 内部服务和外部域名进行解析，例如支持 Kubernetes 的服务发现机制，监听 Kubernetes 的 API 服务器上的 Service 和 Endpoint 变更事件，自动更新 DNS 解析信息。同时，在 Kubernetes 集群中，

CoreDNS 还能够支持多种插件，例如 DNS 重定向、地址过滤、负载均衡等，能够为用户提供更加全面、灵活、高效的 DNS 服务。

操作步骤

对于已经创建的 SCE 集群，您可以安装 CoreDNS 的相关插件来为集群提供 DNS 服务发现能力：

1. 登录云容器引擎控制台。
2. 在控制台的左侧导航栏中点击“集群 ”。
3. 在集群列表页面中，单击目标集群的名称进入集群详情界面。
4. 在集群管理页面的左侧导航栏中，选择“运维管理” ，然后单击“组件管理”。
5. 在组件管理页面找到 coredns 插件，点击“安装”。



6. 如果安装成功，则会显示“已安装”。

4 最佳实践

4.1 在 SCE 集群中部署 Jenkins 并完成应用构建和部署

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群 。
- 确保目标集群的安全组已经开放相关端口号。
- 确保 kubectl 工具已经连接目标集群。

操作步骤

1. 首先需要配置 Jenkins Helm 仓库。

```
helm repo add jenkins https://charts.jenkins.io
helm repo update
```

2. 安装 Jenkins。

   a. 创建 cicd 命名空间。

   ```
   kubectl create ns cicd
   ```

   b. 创建存储卷，用于保存 jenkins 中的数据。
   当您使用 CSI 插件来创建存储卷时，创建具体操作，请参见 CSI 插件安装。

   c. 在 cicd 命名空间下部署 jenkins 应用。

   ```
   helm -n cicd install jenkins jenkins/jenkins \
   ```

```
--set persistence.existingClaim=pvc-csi \
--set controller.adminPassword="adminpwd" \
--set controller.serviceType="LoadBalancer"
```

persistence.existingClaim=pvc-nas：必选项，在 cicd 命名空间下创建的存储卷的 PVC 名称为 pvc-csi。

controller.adminPassword="admin"：可选项，默认将生成随机密码。

controller.serviceType="LoadBalancer"：可选项，默认为 ClusterIP 类型。

d. 上面步骤会创建 1 个 jenkins 的 pod，查看 jenkins 的 pod 是否正常。

```
kubectl -n cicd get po
```

e. 使用浏览器访问 jenkins 服务，输入账号密码进行登录。

3. 创建流水线任务。

a. 登录 Jenkins，在左侧菜单栏单击 New Item。

b. 在 Enter an item name 区域，输入名称 my-pipeline，选择 Pipeline 类型，然后单击 OK。

c. 在页面顶部单击 Pipeline 页签，选择 Hello World 模板，然后单击 Save。

d. 点击 Build Now 执行构建。

e. 可以点击 Build History，然后单击 1#进入该流水线详情页面，然后单击 Console Output 即可查看流水线构建结果。

4.2 搭建 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 架设网站，也可以用作内容管理系统（CMS）。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保目标集群的安全组已经开放相关端口号。
- 确保 kubectl 工具已经连接目标集群。

操作步骤

1. 部署 WordPress 应用。

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

使用上述 YAML 部署 WordPress 应用会自动创建一个 loadbalance 类型的 service，可以通过公网访问 wordpress 应用。

2. 查看 pod 是否已经运行，当 Pod 的状态为 Running 时，表示部署成功。

```
kubectl get pods |grep wordpress
```

3. 访问 WordPress 应用。在浏览器中输入 WordPress 应用的集群外网地址和端口。

4. 选择语言后点击继续，然后填写网站基本信息，点击安装 WordPress。

需要填写的基本信息说明如下：

- 站点标题：WordPress 网站的名称。

- 用户名：登录 WordPress 时所需的用户名，请注意安全性。
- 密码：登录 WordPress 时所需的密码，建议您设置安全性高的密码。
- 您的电子邮件：用于接收通知的电子邮件。

5. 点击登录。

输入在安装 WordPress 时设置的用户名和密码，然后单击登录。

6. 登录成功后，即可打开 WordPress。

4.3 给应用服务挂载弹性公网 IP

背景信息

SCE 集群支持应用服务挂载弹性公网 IP 功能，无需创建 VPC NAT 网关即可让应用访问公网，此功能使得 Serverless 容器应用的部署和服务访问变得更加简单和便利。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保目标集群的安全组已经开放相关端口号。
- 确保 kubectl 工具已经连接目标集群。

操作步骤

1. 登录 VPC 管理控制台，购买弹性公网 IP。

2. 进入弹性负载均衡控制台，创建弹性负载均衡 ELB，并且使用 ELB 绑定弹性公网 IP。

3. 登录容器服务管理控制台，在左侧菜单栏选择“集群”。

4. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“工作负载”下的“无状态”，选择创建 Deployment。

您也可以使用如下 YAML 示例模板创建 Pod：

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - image: nginx:alpine
    imagePullPolicy: Always
    name: nginx
    ports:
    - containerPort: 80
      name: http
      protocol: TCP
  restartPolicy: OnFailure
```

5. 创建 loadbalance 类型的 service，绑定公网 ELB。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress-elb
  namespace: kube-system
  annotations:
```

```
    service.beta.kubernetes.io/ctyun-loadbalancer-id: lb-5jkwoaxf66   # elb id
    service.beta.kubernetes.io/ctyun-loadbalancer-address-type: internet # #私网或公网类型，私网 intranet，公网 internet
spec:
  ports:
  - name: nginx-ingress
    port: 30002
    protocol: TCP
    targetPort: 80
  selector:
    k8s-app: nginx-ingress-controller
  type: LoadBalancer
```

6. 在集群左侧导航栏中，选择“工作负载”下的“容器组”，查看容器组的状态。

7. 使用弹性公网 IP 访问 pod。在浏览器中输入 http://ip:端口地址，您可访问 nginx 欢迎页。其中 ip 为申请的弹性公网 IP 的 IP 地址。

4.4 搭建 Spark 应用

背景信息

Spark 是新一代分布式内存计算框架，Apache 开源的顶级项目。相比于 Hadoop Map-Reduce 计算框架，Spark 将中间计算结果保留在内存中，速度提升 10~100 倍；同时它还提供更丰富的算子，采用弹性分布式数据集(RDD)实现迭代计算，更好地适用于数据挖掘、机器学习算法，极大提升开发效率。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。

操作步骤

步骤一：准备镜像和创建命名空间 namespace

1. 从 dockerHub 镜像仓库获取 Spark 相关镜像。

```
docker pull index.docker.io/caicloud/spark:1.5.2
docker pull index.docker.io/caicloud/zeppelin:0.5.6
```

2. 创建命名空间。

```
#namespace-spark-cluster.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: spark-cluster
  labels:
    name: spark-cluster
```

```
$ kubectl create -f namespace-spark-cluster.yaml
```

3. 查看 Namespace。

```
$ kubectl get ns
NAME            LABELS              STATUS
default         <none>              Active
```

```
spark-cluster name=spark-cluster Active
```

步骤二：启动 master 服务

1．创建无状态工作负载，spark-master-deployment.yaml 可参考如下：

```
#spark-master-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spark-master-controller
  namespace: spark-cluster
spec:
  replicas: 1
  selector:
    matchLabels:
      component: spark-master
  template:
    metadata:
      labels:
        component: spark-master
    spec:
      containers:
      - name: spark-master
        image: index.****/spark:1.5.2       ##替换成您自己的 spark 镜像
```

```
        imagePullPolicy: Always
        command: ["/start-master"]
        ports:
        - containerPort: 7077
        - containerPort: 8080
        resources:
          requests:
            cpu: 100m
```

```
$ kubectl create -f spark-master-deployment.yaml
```

2. 创建 Master-Service，spark-master-service.yaml 可参考如下：

```
# spark-master-service.yaml
kind: Service
apiVersion: v1
metadata:
  name: spark-master
  namespace: spark-cluster
spec:
  ports:
    - port: 7077
      targetPort: 7077
```

```
  selector:
    component: spark-master
```

```
$ kubectl create -f spark-master-service.yaml
service "spark-master"created
```

3. 创建 WebUI-Service，spark-webui.yaml 可参考如下：

```
# spark-webui.yaml
kind: Service
apiVersion: v1
metadata:
  name: spark-webui
  namespace: spark-cluster
spec:
  ports:
    - port: 8080
      targetPort: 8080
  selector:
    component: spark-master
```

```
$ kubectl create -f service/spark-webui.yaml
service "spark-webui" created
```

4. 检查 Master 是否能运行和访问：

```
$ kubectl get deploy -nspark-cluster
```

```
NAME                        DESIRED    CURRENT    AGE
spark-master-controller     1          1          23h


$ kubectl get svc
NAME            CLUSTER-IP       EXTERNAL-IP    PORT(S)     AGE
spark-master    10.254.106.29    <none>         7077/TCP    1d
spark-webui     10.254.66.138    <none>         8080/TCP    18h


$ kubectl get pod -nspark-cluster
NAME                              READY      STATUS      RESTARTS    AGE
spark-master-controller-b3gbf     1/1        Running     0           23h
```

5. 确认 master 正常运行后，再使用 Kubernetes proxy 连接 Spark WebUI：

```
$ kubectl proxy --port=8001
```

然后通过浏览器访问 http://localhost:8001/api/v1/proxy/namespaces/spark-cluster/services/spark-webui/](http://localhost:8001/api/v1/proxy/namespaces/spark-cluster/services/spark-webui/)查看 spark 的任务运行状态。其中 localhost 替换成执行 kubectl proxy 命令的主机 IP，如若在本地主机上执行 kubectl proxy 命令，直接在本地浏览器访问 localhost 即可。

步骤三：启动 Spark workers

Spark workers 启动时需要 Master service 处于运行状态，您可以通过修改 replicas 来设定 worker 数目（比如设定 replicas: 4，即可建立 4 个

Spark Worker）。您可以为每一个 worker 节点设置了 CPU 和内存的配额，保证 Spark 的 worker 应用不会过度抢占集群中其他应用的资源。spark-worker-deployment.yaml 可参考如下：

```
#spark-worker-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: spark-worker-controller
  namespace: spark-cluster
spec:
  replicas: 4
  selector:
    matchLabels:
      component: spark-worker
  template:
    metadata:
      labels:
        component: spark-worker
    spec:
      containers:
      - name: spark-worker
        image: index.caicloud.io/spark:1.5.2
        imagePullPolicy: Always
```

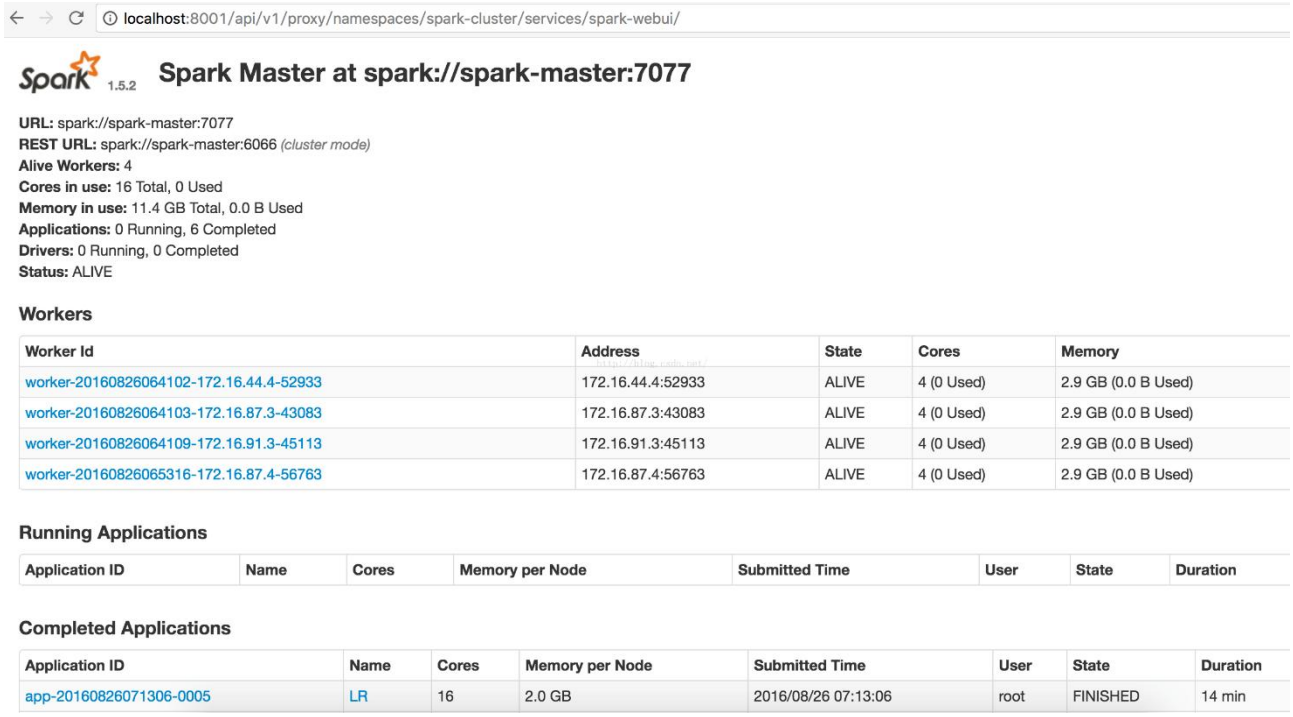
```
        command: ["/start-worker"]
        ports:
        - containerPort: 8081
        resources:
          requests:
            cpu: 100m
```

```
$ kubectl create -f spark-worker-deployment.yaml
deployment "spark-worker-controller" created
```

查看 workers 是否正常运行，通过 kubectl 查询状态（可看到 spark-worker 都已经正常运行）：

```
$ kubectl get pods -nspark-cluster
NAME                            READY     STATUS              RESTARTS   AGE
spark-master-controller-b3gbf   1/1       Running             0          1d
spark-worker-controller-ill4z   1/1       Running             1          2h
spark-worker-controller-j29sc   1/1       Running             0          2h
spark-worker-controller-siue2   1/1       Running             0          2h
spark-worker-controller-zd5kb   1/1       Running             0          2h
```

通过 WebUI 查看： worker 就绪后应该出现在 UI 中。



步骤四：提交 Spark 任务

1. 通过 Spark-client，可以利用 spark-submit 来提交复杂的 Python 脚本、Java/Scala 的 jar 包代码。

```
$ kubectl get pods -nspark-cluster | grep worker
NAME                              READY     STATUS    RESTARTS   AGE
spark-worker-controller-1h0l7     1/1       Running   0          4h
spark-worker-controller-d43wa     1/1       Running   0          4h
spark-worker-controller-ka78h     1/1       Running   0          4h
spark-worker-controller-sucl7     1/1       Running   0          4h
$ kubectl exec spark-worker-controller-1h0l7 -it bash
$ cd /opt/spark


# 提交 python spark 任务
```

```
./bin/spark-submit \
    --executor-memory 4G \
    --master spark://spark-master:7077 \
     examples/src/main/python/wordcount.py \
    "hdfs://hadoop-namenode:9000/caicloud/spark/data"


# 提交 scala spark 任务
./bin/spark-submit
    --executor-memory 4G
    --master spark://spark-master:7077
    --class io.caicloud.LinearRegression
     /nfs/caicloud/spark-mllib-1.0-SNAPSHOT.jar
    "hdfs://hadoop-namenode:9000/caicloud/spark/data"
```

2. 通过 Zeppelin，可以直接在命令行或 UI 编写简单的 spark 代码。

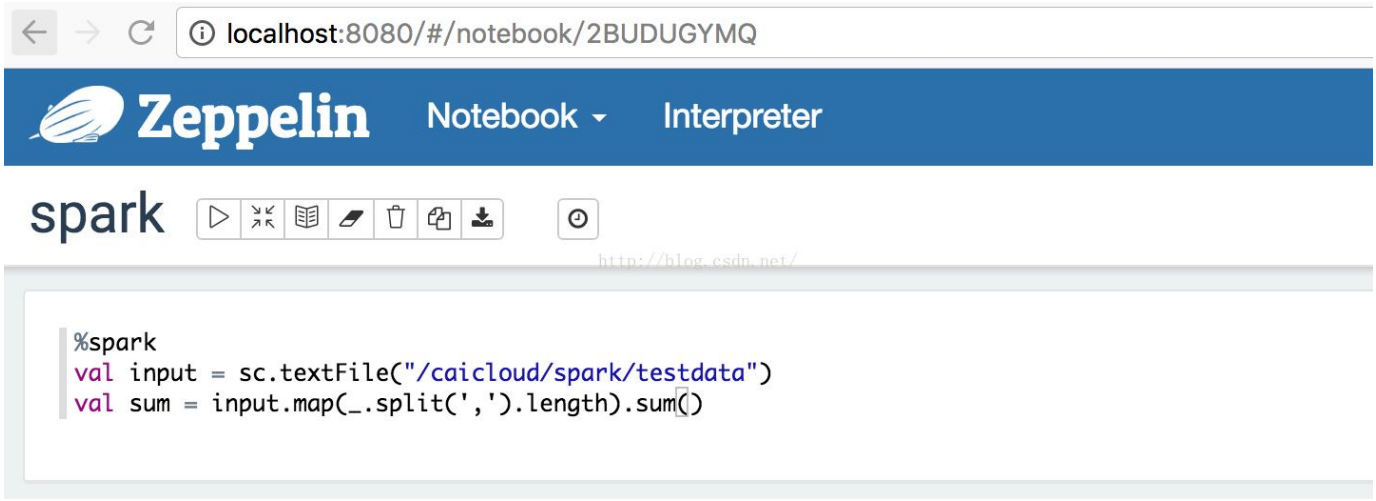
先创建 zeppelin 的工作负载：

```
$ kubectl create -f zeppelin-controller.yaml
deployment "zeppelin-controller"created
$ kubectl get pods -nspark-cluster -l component=zeppelin
NAME                          READY     STATUS     RESTARTS    AGE
zeppelin-controller-5g25x     1/1       Running    0           5h
```

使用已创建的 Zeppelin pod，设置 WebUI 的映射端口：

```
$ kubectl port-forward zeppelin-controller-5g25x 8080:8080
```

访问 http://localhost:8080/，并提交测试代码。



4.5 基于 DNS 的服务发现

背景信息

DNS 为 Kubernetes 集群内的工作负载提供域名解析服务。CoreDNS 是 Kubernetes 集群中负责 DNS 解析的组件，能够支持解析集群内部自定义服务域名和集群外部域名。CoreDNS 具备丰富的插件集，在集群层面支持自建 DNS、自定义 hosts、CNAME、rewrite 等需求。与 Kubernetes 一样，CoreDNS 项目由 CNCF 托管。关于 CoreDNS 的更多信息，，可浏览 CoreDNS 的官网。SCE 集群使用 CoreDNS 负责集群的服务发现，您可以根据不同使用场景配置 CoreDNS 及使用 CoreDNS 提升集群 DNS QPS 性能。。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。

操作步骤

步骤一：安装 CoreDNS 插件

1. 登录容器服务控制台，在左侧菜单栏选择“集群”。
2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“插件”下的“插件市场”，安装 CoreDNS 插件。
3. 提交“安装插件”，稍后会在集群中创建 2 个 coredns pod，集群会根据 Pod 内的配置，将域名请求发往集群 DNS 服务器获取结果。
4. 查看 coredns 是否正常运行。

步骤二：创建 Nginx 工作负载，以及 service

1. 创建无状态工作负载，参考如下：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: "nginx-deploy"
  namespace: "default"
spec:
  replicas: 2
  selector:
    matchLabels:
      name: "nginx-deploy"
  template:
    metadata:
```

```
        labels:
          name: "nginx-deploy"
           source:"CCSE"
      spec:
        containers:
          - image: "registry-vpc-crs-huadong1.ctyun.cn/open-source/nginx:1.25-alpine-amd64"
          imagePullPolicy: "IfNotPresent"
          name: "nginx"
```

2. 查看 nginx pod 是否正常运行，从 eci 控制台进入 pod 容器内部。

```
kubectl get po -ndefault |grep nginx
```

查看 Pod 内的 DNS 域名解析配置文件为/etc/resolv.conf，文件内容如下：

```
nameserver 10.96.0.10
search kube-system.svc.cluster.local svc.cluster.local cluster.local
options ndots:5
```

3. 创建 service，参考如下：

```
apiVersion: v1
kind: Service
metadata:
```

```
  name: nginx-deploy
  namespace: default
spec:
  ports:
  - name: tcp80
    port: 30002
    protocol: TCP
    targetPort: 80
  selector:
    name: nginx-deploy
  type: ClusterIP
```

步骤三：访问 Nginx 服务

● 当业务 Pod（Pod Client）试图访问 Nginx 服务（Service Nginx）时，先会请求本地 DNS 配置文件（/etc/resolv.conf）中指向的 DNS 服务器（nameserver 10.96.0.10，即 Service kube-dns）获取服务 IP 地址，得到解析结果为 10.96.84.23 的 IP 地址。

● 业务 Pod（Pod Client）再直接发起往该 IP 地址的请求，请求最终经过 Nginx 服务（Service Nginx）转发到达后端的 Nginx 容器（Pod Nginx-1 和 Pod Nginx-2）上。

4.6 基于 Service 实现集群内访问负载均衡

背景信息

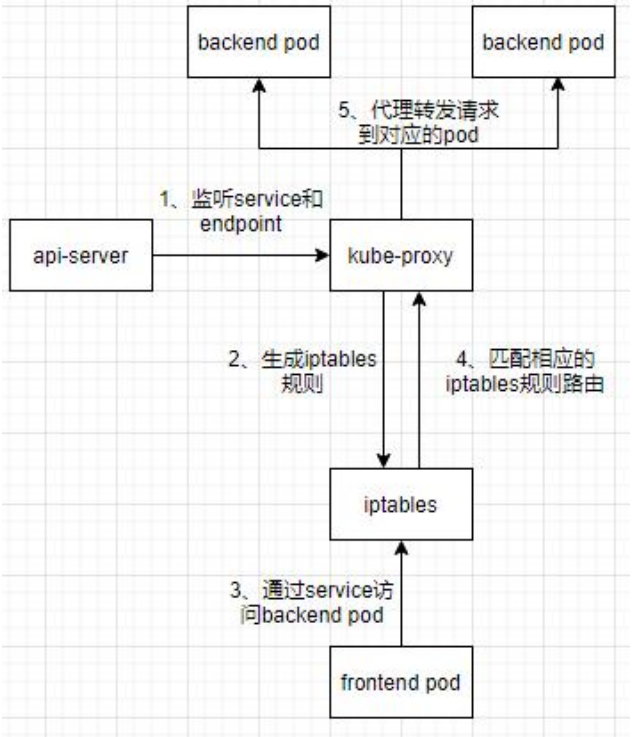
Kubernetes Service 是一组具有相同标签的 Pod 集合的抽象，可以简单地理解为集群内的负载均衡器，集群内外的各个服务可以通过 Service 进行互相通信。不同类型的 Service 适合不同的场景，本文将重点讨论每种类型的 Service 的适用场景以及 kube-proxy 如何实现 Service 的负载均衡。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。
- 假设您在一个 K8S 的集群里面发布了两个应用，前端应用 frontend 和后端应用 backend，前端应用要访问后端应用。
- 确保您的 SCE 集群已安装 CoreDNS 插件。

实现原理

Service 和 iptables 模式的 kube-proxy 在 kubernetes 集群中的工作原理如下：



操作步骤

步骤一：创建后端应用和前端应用

这里使用 nginx 代表后端应用，部署 2 个副本；busybox 代表前端应用，可以从 busybox 中访问 nginx 的 80 端口。

1. 登录容器服务控制台，在左侧菜单栏选择“集群”。
2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“工作负载”下的“无状态”，点击“创建 deployment”。

也可以使用 yaml 创建工作负载，创建 nginx 工作负载参考如下：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: "nginx-deploy"
  namespace: "default"
spec:
  replicas: 2
  selector:
    matchLabels:
      name: "nginx-deploy"
  template:
    metadata:
      labels:
        name: "nginx-deploy"
```

```
            source: "CCSE"
        spec:
          containers:
          - image: "registry-vpc-crs-huadong1.ctyun.cn/open-source/nginx:1.25-alpine-amd64"
            imagePullPolicy: "IfNotPresent"
            name: "nginx"
```

创建 busybox 工作负载参考如下：

```
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  name: "lin-test-3-deploy"
  namespace: "default"
spec:
  replicas: 1
  selector:
    matchLabels:
      name: "busybox"
  template:
    metadata:
      labels:
        name: "busybox"
    spec:
      containers:
      - image: "registry-huadong1.crs-internal.ctyun.cn/open-source/busybox:1.36"
        imagePullPolicy: "IfNotPresent"
```

```
      name: "busybox"
      command:
        - "tail"
        - "-f"
        - "/dev/null"
```

步骤二：创建后端应用和前端应用

创建 service，参考如下：

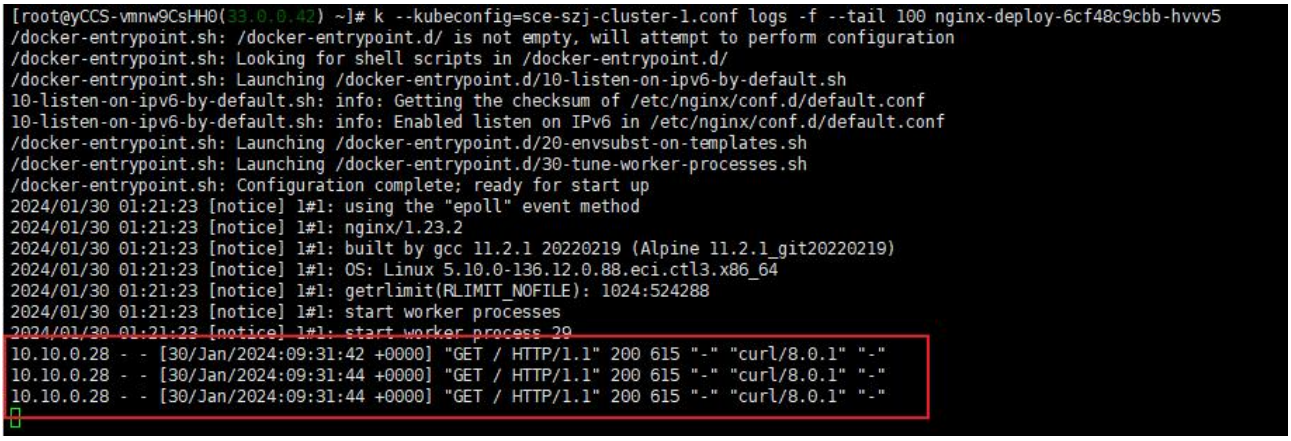
```
apiVersion: v1
kind: Service
metadata:
  name: nginx-deploy
  namespace: default
spec:
  ports:
  - name: tcp80
    port: 30002
    protocol: TCP
    targetPort: 80
  selector:
    name: nginx-deploy
  type: ClusterIP
```

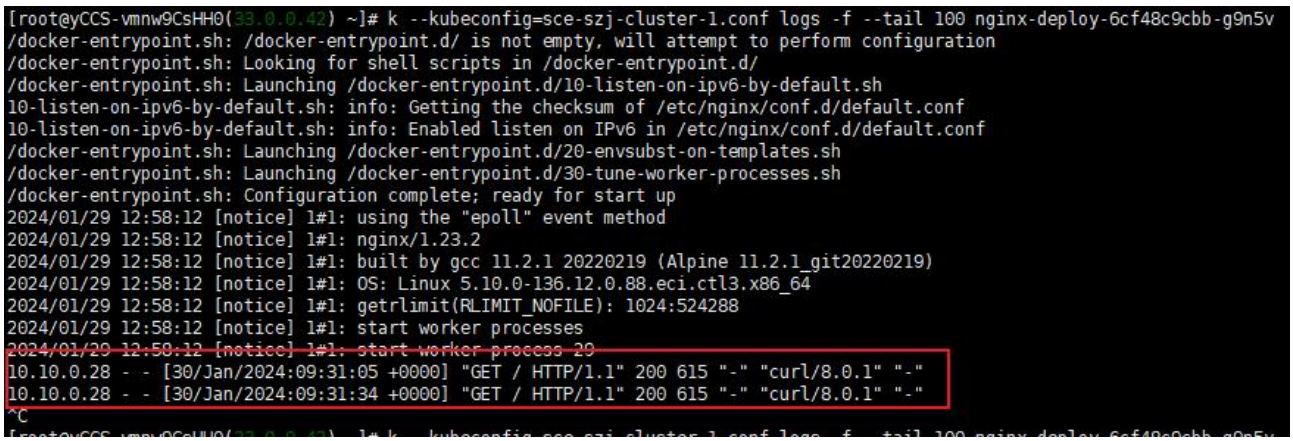
步骤三：集群内使用应用名进行访问后端实现负载均衡

进入前端 busybox 容器内部，使用 service 域名访问后端应用，一共请求 5 次。

```
wget -qO - nginx-deploy:30002
```

可以看到后端请求分别请求到了 2 个后端服务中，实现了负载均衡。





4.7 基于 Ingress 实现集群外访问负载均衡

背景信息

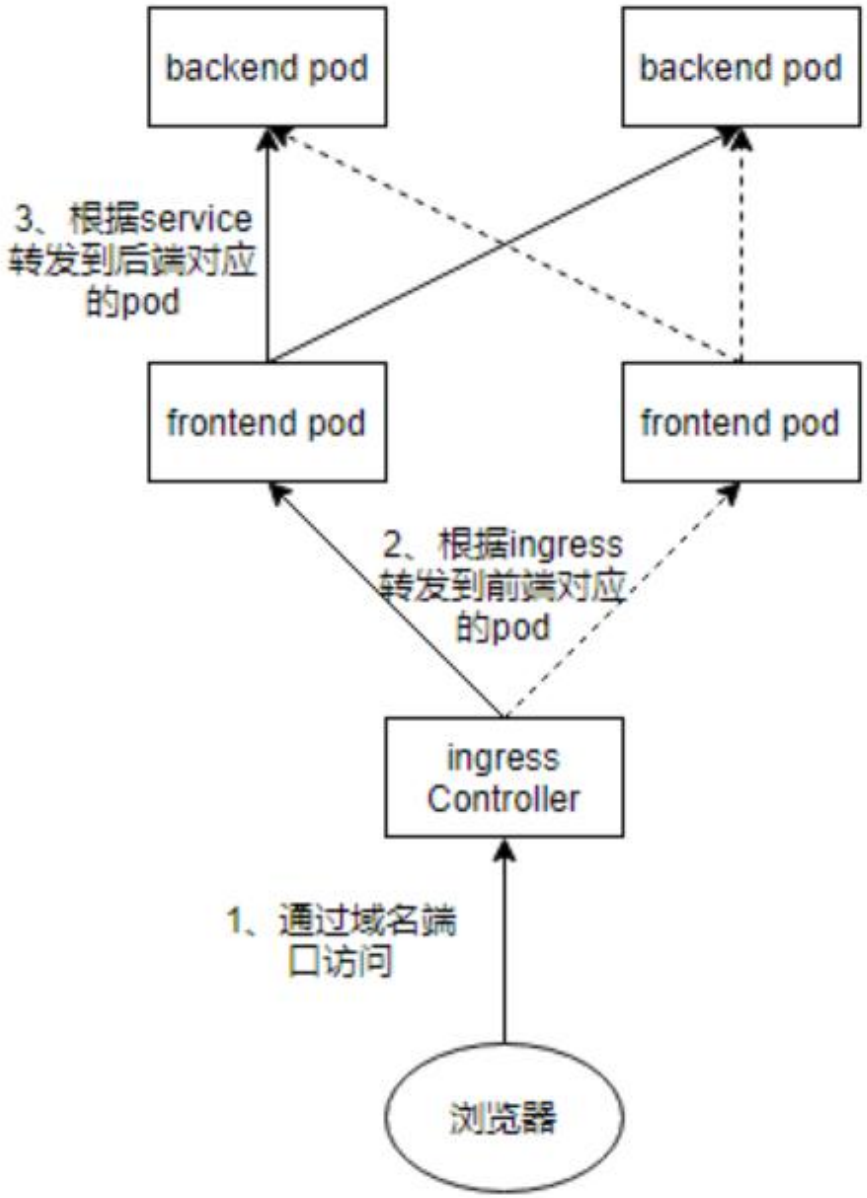
SCE 集群不支持 NodePort，k8s 中的 Ingress 实现了对外部负载均衡器的自动配置，是对 NodePort 的替代方案。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。
- 假设您在一个 K8S 的集群里面发布了两个应用，前端应用 frontend 和后端应用 backend，要在浏览器访问前端应用。
- 确保您的 SCE 集群已安装 CoreDNS 插件。

实现原理

Ingress 和 Service 在 kubernetes 集群中的工作原理如下：



操作步骤

步骤一：安装 nginx-ingress-controller 插件

1. 登录容器服务控制台，在左侧菜单栏选择“集群”。
2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“插件”下的“插件市场”，点击安装 nginx-ingress-controller 插件。
3. 提交安装插件，稍后会在集群中创建 2 个 nginx-ingress-controller pod。

4. 查看 nginx-ingress-controller 是否正常运行。

步骤二：创建前端应用

这里使用 nginx 代表前端应用，部署 2 个副本。

1. 登录容器服务控制台，在左侧菜单栏选择“集群”。
2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“工作负载”下的“无状态”，点击“创建 deployment”。

也可以使用 yaml 创建工作负载，创建 nginx 工作负载参考如下：

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

```
      containers:
      - image: registry-huadong1.crs-internal.ctyun.cn/open-source/nginx:1.25-alpine
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
```

步骤三：创建 service 服务

1. 为前端 nginx 工作负载创建 service，参考如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  ports:
  - port: 30003
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

```
  type: ClusterIP
```

2. 为 nginx-ingress-controller pod 创建 service，由于 SCE 集群不支持 NodePort 类型的 service，这里创建 LoadBalance 类型，绑定 ELB，参考如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress-elb
  namespace: kube-system
  annotations:
    service.beta.kubernetes.io/ctyun-loadbalancer-id: lb-5jkwoaxf66   # elb id
    service.beta.kubernetes.io/ctyun-loadbalancer-address-type: internet # #私网或公网类型，私网 intranet，公网 internet
spec:
  ports:
  - name: nginx-ingress
    port: 30002
    protocol: TCP
    targetPort: 80
  selector:
    k8s-app: nginx-ingress-controller
  type: LoadBalancer
```

步骤四：为前端应用创建 Igress 路由

1. 创建 ingress，参考如下：

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  namespace: default
spec:
  rules:
  - host: myingress.com
    http:
      paths:
      - backend:
          service:
            name: nginx-service
            port:
              number: 30003
        path: /nginx
        pathType: Prefix
```

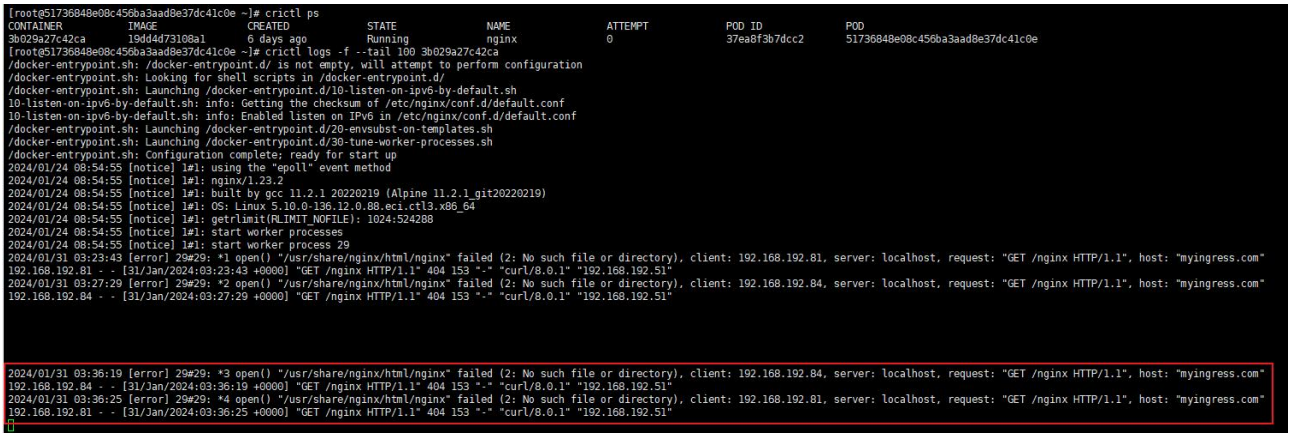
步骤五：使用域名访问前端应用实现负载均衡

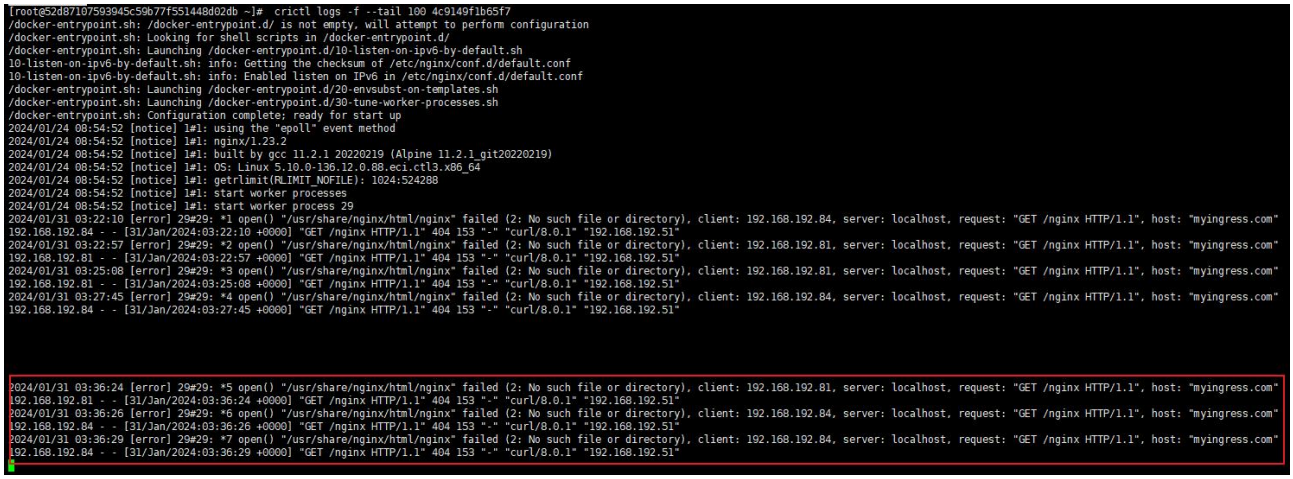
进入任意 pod 容器内部，使用 myingress.com 路由访问前端应用，一共请求 5 次。

```
$curl -H "Host:myingress.com" nginx-ingress-elb.kube-system.svc:30002/nginx
```

```
5c805af62e1348cbb3a8bc3262d203cf:~# curl -H "Host:myingress.com" ngixn-ingress-svc.kube-system:30002/nginx
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.23.2</center>
</body>
</html>
5c805af62e1348cbb3a8bc3262d203cf:~#
```

可以看到通过 myingress.com/nginx 域名是能正常请求到前端 nginx 应用的，并且分别请求到了 2 个前端 nginx 服务中，实现了负载均衡。





4.8 部署高可靠 Ingress Controller

背景信息

Nginx Ingress Controller 是一个用于 Kubernetes 环境的开源 Ingress 控制器，它基于 Nginx 服务器实现了负载均衡、SSL 终止和路由

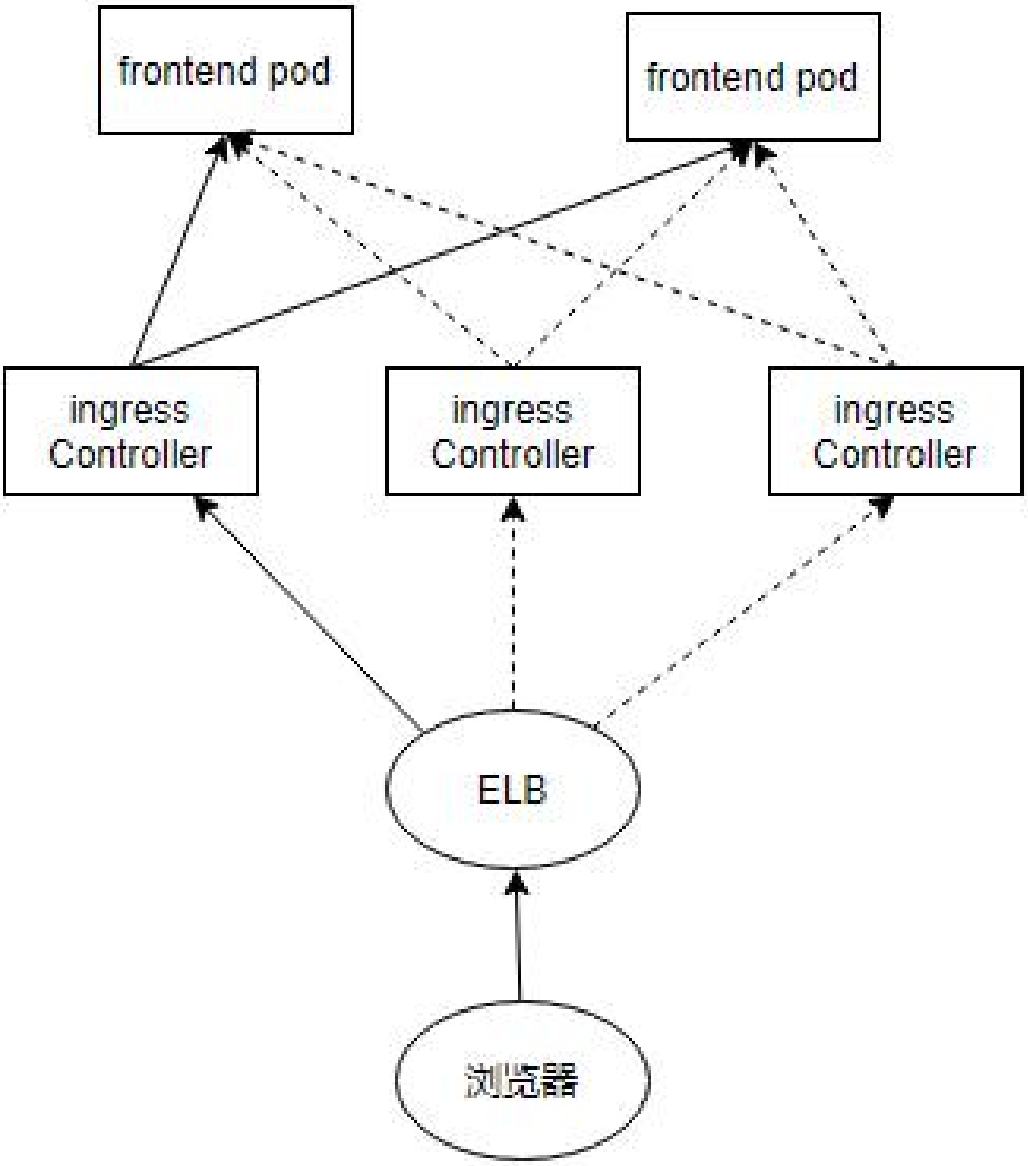
功能。通过使用 Nginx Ingress Controller，你可以轻松地在 Kubernetes 集群中管理入站流量，并将 HTTP 和 HTTPS 请求路由到不同的服务。它还支持基于规则的路由、TLS 终止和灵活的配置选项，使得在 Kubernetes 中管理和控制流量变得更加简单和高效。作为集群流量接入层，Ingress 的高可用性显得尤为重要，为了达到生产级的阈值，我们必须要要配置 ingress 的高可用。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。

实现原理

高可用首先要解决的就是单点故障问题，在 sce 集群中 Nginx Ingress Controller 通常采用多副本部署的方式，同时由于 Ingress 作为集群流量接入口，可以在 ingress 前面使用 ELB 来统一代理 ingress-controller 的服务，以负载均衡到不同的 ingress-controller pod。高可用架构图如下：



如上述部署架构图所示，由多个 Ingress-controller 实例组成统一接入层来承载集群入口流量，同时可依据后端业务流量水平扩缩容 Ingress-controller pod。

您可以在容器服务控制台页面上，通过为应用创建不同的 Ingress 对象，来为不同的应用指定不同的域名。IngressController 目前不支持配置 HTTPS 证书，后续会支持，所以该方案目前不支持 HTTPS，只支持 HTTP。

操作步骤

步骤一：安装 nginx-ingress-controller 插件

1. 登录容器服务控制台，在左侧菜单栏选择集群。
2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“插件”下的“插件市场”，点击安装 nginx-ingress-controller 插件。
3. 提交安装插件，稍后会在集群中创建 2 个 nginx-ingress-controller pod。
4. 查看 nginx-ingress-controller 是否正常运行。
5. 您可以根据业务流量水平修改 nginx-ingress-controller deployment 的副本数量。

步骤二：创建前端应用

这里使用 nginx 代表前端应用，部署 2 个副本。

1. 登录容器服务控制台，在左侧菜单栏选择“集群”。
2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“工作负载”下的“无状态”，点击“创建 deployment”。

也可以使用 yaml 创建工作负载，创建 nginx 工作负载参考如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
```

```
    template:
      metadata:
        labels:
          app: nginx
      spec:
        containers:
        - image: registry-huadong1.crs-internal.ctyun.cn/open-source/nginx:1.25-alpine
          imagePullPolicy: Always
          name: nginx
          ports:
          - containerPort: 80
            protocol: TCP
```

步骤三：创建 service 服务

1. 为前端 nginx 工作负载创建 service，参考如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  ports:
  - port: 30003
    protocol: TCP
    targetPort: 80
```

```
    selector:
      app: nginx
    type: ClusterIP
```

2. 为 nginx-ingress-controller pod 创建 service，创建 LoadBalance 类型的 service，并绑定 ELB，参考如下：

```
  apiVersion: v1
  kind: Service
  metadata:
    name: ngixn-ingress-svc
    namespace: kube-system
  spec:
    ports:
    - name: nginx-ingress
      port: 30002
      protocol: TCP
      targetPort: 80
    selector:
      k8s-app: nginx-ingress-controller
    type: LoadBalance
```

步骤四：为前端应用创建 Ingress 路由

创建 ingress，参考如下：

```
  apiVersion: networking.k8s.io/v1
```

```
kind: Ingress
metadata:
  name: nginx-ingress
  namespace: default
 annotations:
    kubernetes.io/ingress.class: "nginx-ingress-controller"
spec:
  rules:
  - host: myingress.com
    http:
      paths:
      - backend:
          service:
            name: nginx-service
            port:
              number: 30003
        path: /nginx
          pathType: Prefix
```

步骤五：使用域名进行访问前端应用

在浏览器访问 myingress.com 域名需要在 hosts 中配置该域名对应的 ELB 的 IP 地址。这样在浏览器通过域名访问，DNS 服务器把域名解析为 ELB 的 IP，ELB 把请求转给其中某个 IngressController，IngressController 通过域名转发到不同的集群内应用。

4.9 集群内请求会话保持

背景信息

会话保持，有时也称为“粘性会话”（Sticky Sessions）。启用会话保持后，负载均衡会将来自同一客户端的访问请求持续分发到同一台后端云服务器上进行处理。简单来说，如果用户需要登录，就可以理解为会话；如果不需要登录，就可以理解为连接。

实际上，会话保持机制与负载均衡的基本功能是完全矛盾的。负载均衡的目标是将来自客户端的连接和请求均衡地转发至后端的多台服务器，以避免单台服务器负载过高；而会话保持机制则要求将某些请求转发至同一台服务器进行处理。因此，在实际的部署环境中，需要根据应用环境的特点选择适当的会话保持机制。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。
- 确保您的 SCE 集群已安装 CoreDNS 插件。

操作步骤

不启用会话保持

步骤一：创建工作负载

创建一个 nginx 工作负载，并确保工作负载的实例个数大于 1，部署 2 个副本，工作负载不需要其他的额外特殊配置。

1. 登录容器服务控制台，在左侧菜单栏选择“集群”。

2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“工作负载”下的“无状态”，点击“创建 deployment”。

也可以使用 yaml 创建工作负载，创建 nginx 工作负载参考如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: registry-huadong1.crs-internal.ctyun.cn/open-source/nginx:1.25-alpine
        imagePullPolicy: Always
        name: nginx
```

```
        ports:
        - containerPort: 80
          protocol: TCP
```

步骤二：创建 service 服务

为 nginx 工作负载创建 service，注意 Session Affinity 不需要设置，保持默认值即可。参考如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  ports:
  - port: 30003
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: ClusterIP
```

步骤三：使用 service 域名访问前端应用实现负载均衡

进入任意 pod 容器内部，发起服务调用，使用 service 域名访问后端应用，在 pod 容器内部执行这个命令 100 次。

```
$ for i in $(seq 1 100); do curl nginx-service.default:30003; done;
```

观察工作负载日志，查看 Pod 实例的日志输出。

可以看到服务请求会随机的转发到任一个 Pod 实例，实现了负载均衡。

启用会话保持

步骤一：创建工作负载

创建一个 nginx 工作负载，部署 2 个副本。同“不启用会话保持”。

步骤二：创建 service 服务

创建一个 ClusterIP 类型的服务（Service）并关联到上述 nginx 工作负载，注意需要展开高级设置，并设置 Session Affinity 为客户端 IP。参考如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  ports:
  - port: 30003
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

```
  type: ClusterIP
  sessionAffinity: ClientIP
  sessionAffinityConfig:
    clientIP:
      timeoutSeconds: 10800
```

当设置了会话保持之后，k8s 会根据访问的 ip 来把请求转发给以前访问过的 pod，其中 timeoutSeconds 指的是会话保持的时间，这个时间默认是 10800 秒。

步骤三：使用 service 域名访问前端应用实现会话保持

进入任意 pod 容器内部，发起服务调用，使用 service 域名访问后端应用，在 pod 容器内部执行这个命令 100 次。

```
$for i in {1..100};do curl nginx-service:30003;done;
```

观察工作负载日志，查看 Pod 实例的日志输出。

可以看到服务请求会全部转发到某一个 Pod 实例，进行会话保持。

4.10 集群外 Ingress 访问请求会话保持

背景信息

7 层的模式下可以开启基于 http cookie 和 app cookie 的会话保持 ，在 ingress 上开启基于 cookie 的会话保持需要满足以下条件：

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。

操作步骤

步骤一：创建工作负载

创建一个 nginx 工作负载，并确保工作负载的实例个数大于 1，部署 2 个副本，工作负载不需要其他的额外特殊配置。

1. 登录容器服务控制台，在左侧菜单栏选择“集群”。
2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“工作负载”下的“无状态”，点击“创建 deployment”。

也可以使用 yaml 创建工作负载，创建 nginx 工作负载参考如下：

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

```
      containers:
      - image: registry-huadong1.crs-internal.ctyun.cn/open-source/nginx:1.25-alpine
        imagePullPolicy: Always
        name: nginx
        ports:
        - containerPort: 80
          protocol: TCP
```

步骤二：创建 service 服务

为 nginx 工作负载创建 service，注意 Session Affinity 不需要设置，保持默认值即可。参考如下：

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
  namespace: default
spec:
  ports:
  - port: 30003
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
```

```
  type: ClusterIP
```

步骤三：确保当前命名空间已经绑定到一个负载均衡器

步骤四：创建一个生产路由（Ingress）并关联到上述服务（Service）

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  namespace: default
  annotations:
    kubernetes.io/ingress.class: "nginx-ingress-controller"
spec:
  rules:
  - host: myingress.com
    http:
      paths:
      - backend:
          service:
            name: nginx-service
            port:
```

```
            number: 30003
        path: /nginx
        pathType: Prefix
```

步骤五：发起服务调用

添加本地 hosts 映射：ip 为 Nginx-Ingress-Controller 的访问地址；域名为创建 Ingress 时填入的域名，如：10.142.232.160 myingress.com。

在浏览器中多次发起对服务的请求，这里没法通过 curl 来测试验证，因为 curl 请求时没法保持 Cookie。

观察工作负载日志。

结论：浏览器中的请求会全部转发到某一个 Pod 实例，进行会话保持。

4.11 将 Ingress 服务暴露到公网

背景信息

Ingress 基于 Nginx 服务器实现了负载均衡、SSL 终止和路由功能。如果您的服务是通过 Ingress 进行访问的，并且需要通过公网访问，Ingress 作为集群流量接入层，可为集群 Ingress 服务配置天翼云 ELB。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 确保 kubectl 工具已经连接目标集群。
- 确保已经开通天翼云 ELB 服务，并且已创建一个外网可用的 ELB（在 ELB 控制台创建）。

操作步骤

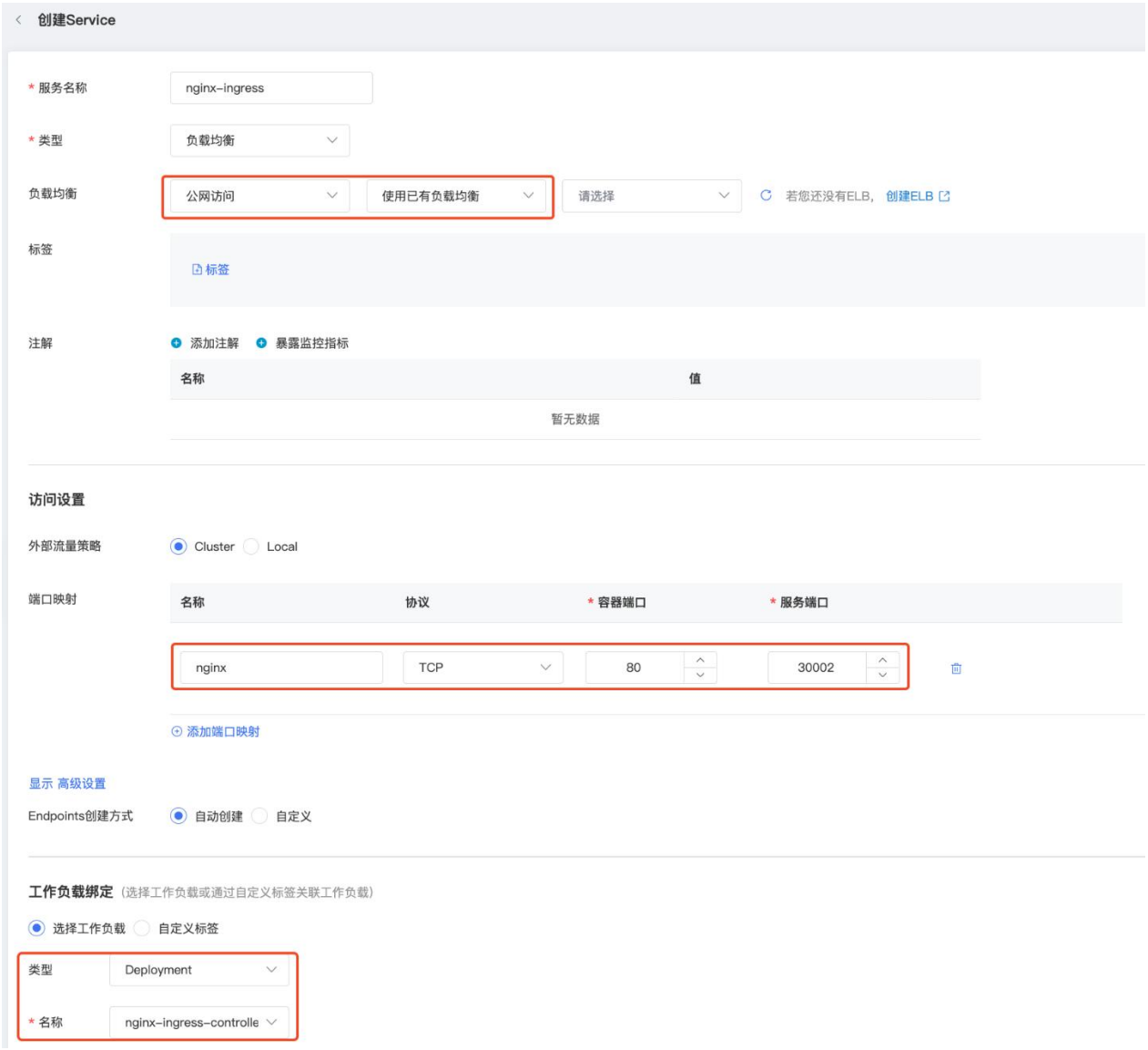
1. 登录容器服务控制台，在左侧菜单栏选择“集群”。

2. 在集群列表页面，选择目标集群名称，然后在左侧菜单栏选择“网络”下的“服务”，命名空间选择“kube-system”，点击”创建服务“按钮，对以下信息进行配置：

   a. 填写 service 相关信息，其中负载均衡一栏，选择公网访问，从列表中选择一个要绑定的 ELB（如果列表为空，请确认您是否有可用的 ELB，没有的话，请先到 ELB 控制台创建）。

   b. 在端口映射一栏，填写好容器端口和要映射的服务端口（该端口也是负载均衡的监听端口）。

   c. 在工作负载绑定一栏，类型选择 Deployment，名称选择 nginx-ingress-controller-nginx-ingress-controller，然后点击提交。



3. 待 ELB 绑定后，即可通过服务列表中“集群外访问”中的外网地址访问服务了。

4.12 基于 Ingress 实现服务发布

背景信息

当对服务进行版本更新升级时，需要使用到滚动升级、分批暂停发布、蓝绿发布以及灰度发布等发布方式。本文将介绍在 SCE 集群中如何通过 Nginx Ingress Controller 来实现应用服务的灰度发布。

当对服务进行版本更新升级时，需要使用到滚动升级、蓝绿发布以及灰度发布等发布方式。

- 滚动更新：依次进行新旧替换，直到旧的全部被替换为止。
- 蓝绿发布：两套独立的系统，对外提供服务的称为绿系统，待上线的服务称为蓝系统，当蓝系统里面的应用测试完成后，用户流量接入蓝系统，蓝系统将称为绿系统，以前的绿系统就可以销毁。
- 灰度发布：在一套集群中存在稳定和灰度两个版本，灰度版本可以限制只针对部分人员可用，待灰度版本测试完成后，可以将灰度版本升级为稳定版本，旧的稳定版本就可以下线了，也称之为金丝雀发布。

前提条件

- 确保您已经创建 SCE 集群，具体操作请参阅创建 SCE 集群。
- 在集群中安装 nginx-ingress-controller 插件，作为 Ingress Controller，并通过 Nginx 对外暴露统一的流量入口。详细操作可参考安装插件。

实现原理

nginx-ingress 是 Kubernetes 官方推荐的 ingress controller，它是基于 nginx 实现的，增加了一组用于实现额外功能的 Lua 插件。

为了实现灰度发布，ingress-nginx 通过定义 annotation 来实现不同场景的灰度发布，其支持的规则如下：

- nginx.ingress.kubernetes.io/canary-by-header：基于 Request Header 的流量切分，适用于灰度发布以及 A/B 测试。当 Request Header 设置为 always 时，则将请求切分到 Canary Ingress 定义的 Service 上；当 Request Header 设置为 never 时，请求不会被发送到 Canary 入口，会将请求转发到常规 Ingress 定义的 Service 上；对于任何其他 Header 值，将忽略 Header，并通过优先级将请求与其他金丝雀规则进行优先级的比较。
- nginx.ingress.kubernetes.io/canary-by-header-value：要匹配的 Request Header 的值，用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务。当 Request Header 设置为此值时，它将被路由到 Canary 入口，将请求切分到 Canary Ingress 定义的 Service 上。该规则允许用户自定义 Request Header 的值，必须与上一个 annotation (即：canary-by-header）一起使用。
- nginx.ingress.kubernetes.io/canary-by-cookie：基于 Cookie 的流量切分，适用于灰度发布与 A/B 测试。用于通知 Ingress 将请求路由到 Canary Ingress 中指定的服务的 cookie。Cookie 值仅支持“always”和“never”。当 cookie 值设置为 always 时，它将被路由到 Canary 入口；当 cookie 值设置为 never 时，请求不会被发送到 Canary 入口；对于任何其他值，将忽略 cookie 并将请求与其他金丝雀规则进行优先级的比较。
- nginx.ingress.kubernetes.io/canary-weight：基于服务权重的流量切分，适用于蓝绿部署，权重取值范围为[0-100]，表示 Canary Ingress 所切分到的流量百分比。权重为 0 意味着该金

丝雀规则不会向 Canary 入口的服务发送任何请求。权重为 100 意味着所有请求都将被发送到 Canary 入口。

以上策略的优先级顺序为： canary-by-header > canary-by-cookie > canary-weight 。

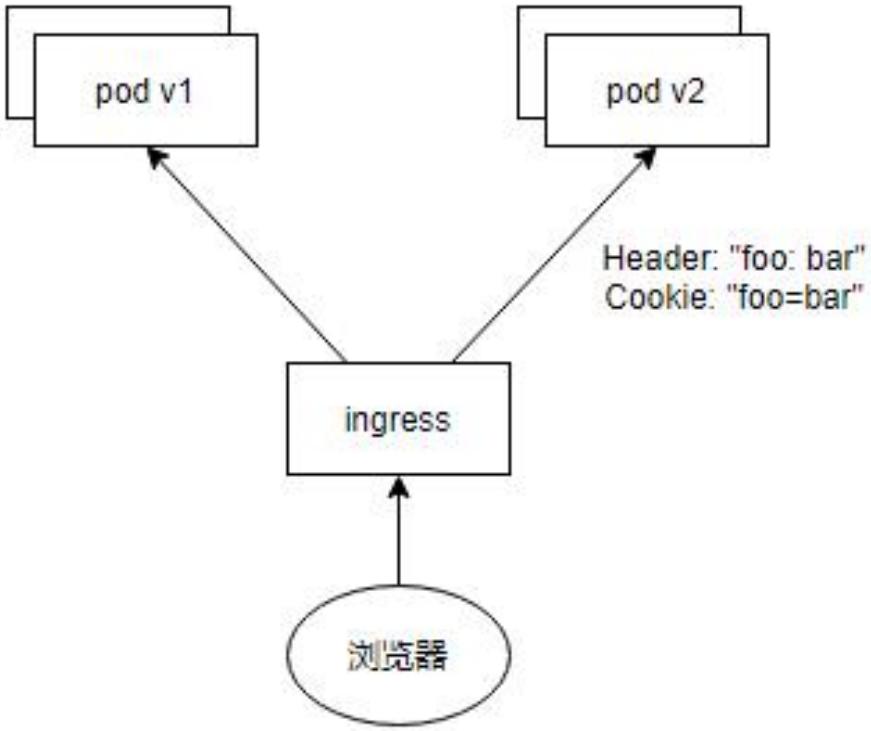
基于以上 annotation 的发布思路如下：

1. 在集群中部署新旧两套应用版本，一套是 stable 版本，一套是 canary 版本，两个版本都有自己的 service。
2. 定义两个 Ingress 配置，一个正常提供服务，一个增加 canary 的 annotation。
3. 待 canary 版本无误后，将其切换成 stable 版本，并且将旧的版本下线，流量全部接入新的 stable 版本。

应用场景

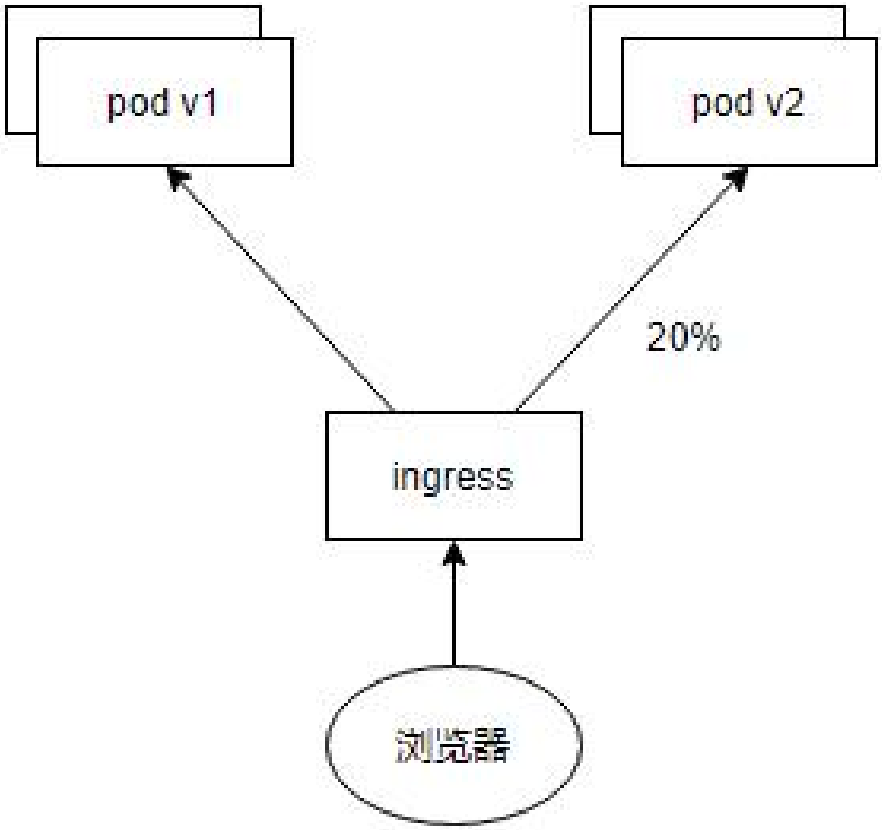
场景一：基于用户请求将匹配的业务流量切分到新版本

假设在当前线上环境中，您已经有一套服务 Service v1 对外提供 7 层服务，此时开发了一些新的功能，现需发布新版本 Service v2 服务。但又不想直接替换 Service v1 服务，而是希望将请求头包含 “foo=bar” 或者 Cookie 包含 “foo=bar” 的客户端请求转发到 Service v2 服务中，验证一下新版本功能是否正常，待稳定运行后，再逐步全量切到 Service v2 服务，平滑下线 Service v1 服务。示意图如下：



场景二：基于服务权重将业务流量切分到新版本

假设当前线上环境，您已经有一套服务 Service v1 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 v2。但又不想将所有客户端流量切换到新版本 Service v2 中，而是希望将 20%的流量灰度到 Service v2，待稳定运行后，逐步全量切到 Service v2，平滑下线 Service v1。示意图如下：



操作步骤

场景一：基于用户请求将匹配的业务流量切分到新版本

步骤一：部署旧版本 Service v1 和常规 Ingress

这里使用 Ingress 作为 service v1 应用服务，并且为方便观测流量切分的效果，将 nginx 欢迎页设置为“v1”。

1. 创建配置 configmap，key 为 index.html，value 为 v1。
2. 创建 nginx 工作负载，配置数据卷为刚才创建的 configmap；配置镜像和挂载卷，挂载容器路径为：/usr/share/nginx/html；配置访问设置，选择虚拟集群 IP 类型，容器端口 80，服务端口 30080。

3. 创建旧版本 service v1 的常规 ingress。灰度 ingress 一栏选择否，在域名路径规则一栏填写域名，指定服务名称以及端口等。

4. 检查通过 Ingress 域名能正常访问旧版本 service v1 服务。

步骤二：部署新版本 Service v2

这里同样使用 Ingress 作为 service v2 应用服务，并且为方便观测流量切分的效果，将 nginx 欢迎页设置为“v2”。

1. 创建配置 configmap，key 为 index.html，value 为 v2。

2. 创建 Ingress 工作负载，配置数据卷为刚才创建的 configmap；配置镜像和挂载卷，挂载容器路径为：/usr/share/nginx/html；配置访问设置，选择虚拟集群 IP 类型，容器端口 80，服务端口 30081。

步骤三：创建灰度 ingress，在灰度发布新版本

1. 基于 Header 创建新版本 service v2 的 Ingress。

2. 在灰度 Ingress 一栏选择是；在生产 ingress 一栏选择旧版本 service v1 的常规 Ingress；在流量切换方式一栏选择灰度，基于 Header 的区分方式，填写 Header key 为 foo，Header value 为 bar，精确匹配；在域名路径规则一栏填写域名，指定服务名称和端口等。

执行命令进行访问测试，<EXTERNAL_IP>为 Nginx Ingress 对外暴露的 IP：

```
# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com'

v1

# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com' -H 'foo: bar'

v2

# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com'
```

```
v1
# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com' -H 'foo: bar'
v2
```

可以看出，仅当 Header 中包含 foo 且值为 bar 的流量才会切分到新版本 service v2 服务。

3. 基于 Cookie 创建新版本 service v2 的 Ingress。

在灰度 Ingress 一栏选择是；在生产 Ingress 一栏选择旧版本 service v1 的常规 Ingress；在流量切换方式一栏选择灰度，基于 Cookie 的区分方式，填写 Cookie key 为 foo，Cookie value 为 always，精确匹配；在域名路径规则一栏填写域名，指定服务名称和端口等。

执行命令进行访问测试，<EXTERNAL_IP>为 Nginx Ingress 对外暴露的 IP：

```
# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com'
v1
# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com' --cookie 'foo=bar'
v2
# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com'
v1
# curl http://<EXTERNAL_IP> -H 'Host: test-gray.com' --cookie 'foo=bar'
v2
```

可以看出，仅当 Cookie 中包含 foo 且值为 bar 的流量才会切分到新版本 service v2 服务。

步骤四：下线旧版本 Service v1 服务

1. 将 service v1 的常规 Ingress 的服务名称改为 service v2 服务。

2. 删除 service v2 的 Ingress。

3. 删除旧版本 service v1 的无状态工作负载和配置项。

平滑下线旧版本后，通过原来的常规 Ingress 请求的流量都会切分到新版本 Service v2 服务了。

场景二：基于服务权重将业务流量切分到新版本

步骤一：部署旧版本 Service v1 和常规 Ingress

同“场景一：基于用户请求将匹配的业务流量切分到新版本”。

步骤二：部署新版本 Service v2

同“场景一：基于用户请求将匹配的业务流量切分到新版本”。

步骤三：创建灰度 Ingress，在灰度发布新版本

1. 基于服务权重新版本 service v2 的 Ingress。

2. 在灰度 Ingress 一栏选择是；在生产 ingress 一栏选择旧版本 service v1 的常规 Ingress；在流量切换方式一栏选择蓝绿，配置全部切到灰度的权重百分比；在域名路径规则一栏填写域名，指定服务名称和端口等。

执行命令进行访问测试，<EXTERNAL_IP>为 Nginx Ingress 对外暴露的 IP：

```
$ for i in {1..10}; do curl http://<EXTERNAL_IP> -H 'Host: test-gray.com'; done;
v2
v2
v2
v2
```

```
v1
v1
v1
v2
v1
v2
```

可以看出，有近 50%的流量切分到新版本 service v2 服务，当请求的数量越多时比例会越接近 50%。

步骤四：下线旧版本 Service v1 服务

1. 将 service v1 的常规 Ingress 的服务名称改为 service v2 服务。

2. 删除 service v2 的 Ingress。

3. 删除旧版本 service v1 的无状态工作负载和配置项。

平滑下线旧版本后，通过原来的常规 Ingress 请求的流量都会切分到新版本 Service v2 服务了。

5 常见问题

5.1 SCE 集群创建失败的解决方法

SCE 集群创建失败的原因多为两类：订购集群订单时提交失败，集群订单已支付但开通失败。

订购集群时无法提交订单

问题的提示：校验失败 apiserver ELB 的 6443 端口已被监听。

可能原因：选择的弹性负载均衡 ELB 的 6443 端口已经被绑定。

解决方法：建议您重新选择另一个 ELB 或者创建 ELB。

订购集群订单时提交失败

- 问题的提示：系统异常，请稍后重试。

可能原因：网路异常或天翼云官网异常。

解决方法：建议您刷新几次，如恢复正常则可，否则提交客服工单处理。

- 问题的提示：询价异常，请刷新重试。

可能原因：网路异常或天翼云订单模块异常。

解决方法：建议您刷新几次，如恢复正常则可，否则提交客服工单处理。

- 问题的提示：错误码是 404 或 900。

可能原因：资源获取异常。

解决方法：提交客服工单处理。

集群订单已支付但开通失败

问题的提示：CCSE 控制台显示“开通中”状态，并且集群开通超过 15 分钟仍未能正常开通。

可能原因：容器产品依赖组件较多，部署准备工作时间久，依赖底层资源创建失败。

解决方法：建议稍等几分钟，如持续未能正常开通，保留页面截图和记录订单号后联系天翼云客服工单处理。

5.2 SCE 集群退订或删除常见问题

SCE 集群无法退订

问题的提示：CCSE 控制台显示“开通中”状态，并且集群开通超过 15 分钟仍未能正常开通。

可能原因：容器产品依赖组件较多，依赖底层资源创建失败。只有运行中的集群才能退订。

解决方法：建议稍等几分钟，如持续未能正常开通，保留页面截图和记录订单号后联系天翼云客服工单处理。

SCE 集群是否可以直接删除?

SCE 集群无法直接删除，只能通过退订销毁 SCE 集群实例。

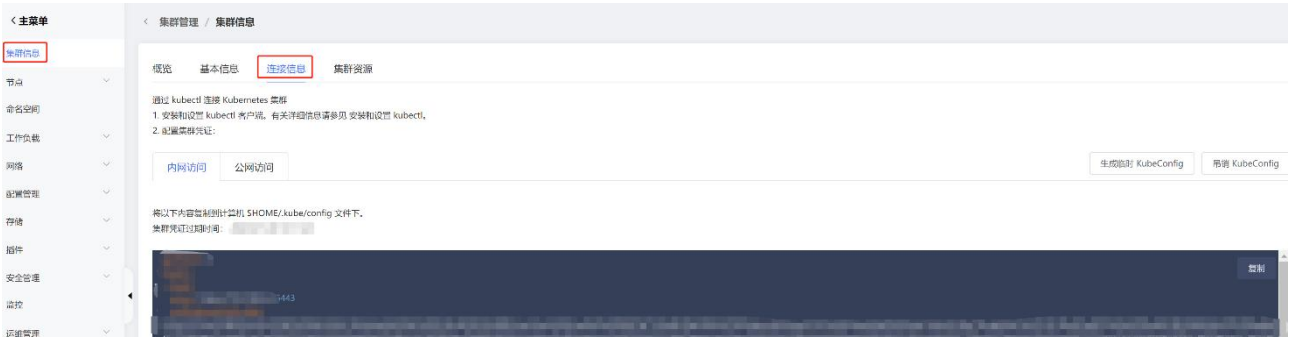
集群退订之后相关数据能否再次找回?

集群退订之后，部署在集群上的工作负载也会同步删除，无法恢复，请慎重退订集群。

5.3 API&kubectl

用户访问集群 API Server 的方式有哪些?

集群 API 方式：集群 API 需要使用证书认证访问，在 CCSE 控制台集群信息 > 连接信息获取 kubeconfig 文件，通过 kubectl 直接连接集群 API Server。



5.4 域名 DNS 异常排查

域名解析失败，如何定位处理?

问题现象：域名解析失败。

可能原因：域名解析失败可能有如下 4 种情况：sce 集群内是否已经安装了 coreDNS 插件、coreDNS 服务是否正常、集群使用的安全组是否已经放开 udp 规则、pod 容器到 coreDNS 网络是否连通。

解决方法：

1. 判断当前的异常原因。
2. 检查业务 Pod 的 DNS 配置，是否已经接入 CoreDNS。
3. 检查 CoreDNS Pod 运行状态进行诊断。
4. 检查 CoreDNS 运行日志进行诊断。
5. 检查 pod 是否能访问 CoreDNS。
6. 检查安全组是否已经放开 UDP 协议的 53 端口。

域名解析失败，如何定位处理?

问题现象：域名解析失败。

可能原因：域名解析失败可能有如下 4 种情况：sce 集群内是否已经安装了 coreDNS 插件、coreDNS 服务是否正常、集群使用的安全组是否已经放开 udp 规则、pod 容器到 coreDNS 网络是否连通。

解决方法：

1. 判断当前的异常原因。
2. 检查业务 Pod 的 DNS 配置，是否已经接入 CoreDNS。
3. 检查 CoreDNS Pod 运行状态进行诊断。
4. 检查 CoreDNS 运行日志进行诊断。
5. 检查 pod 是否能访问 CoreDNS。
6. 检查安全组是否已经放开 UDP 协议的 53 端口。

CoreDNS 插件已安装但是在 pod 内部无法解析 kubernetes 等 service 域名

问题现象：在容器内部是可以 ping 通 coredns pod 的 ip，查看容器内部的/etc/resolv.conf 也是正常的，但是 nslookup kubernetes 就是不能解析出 ip。

可能原因：在 nslookup kubernetes 的时候，容器先通过 /etc/resolv.conf 中的 nameserver 写的 coredns 的 serviceIP 找到 dns 服务器，再通过 dns 服务器解析内部 service 域名。首先得确保 coredns 的 service 正常工作，使用 curl 10.96.0.10:9153 测试 coredns 的 service 明显不通。

解决方法：需要排查 kube-proxy 是否正常。

安装 CoreDNS 插件后并没有修改容器内部的/etc/resolv.conf

问题现象：查看随便一个 pod，进入容器内部查看 cat /etc/resolv.conf，发现并没有被 coredns 修改。

可能原因：可能是 CoreDNS 工作不正常。检查 coredns pod 是否有事件报错健康检查失败。

解决方法：检查 coredns 日志是否正常，重启 coreDNS。

安全组配置错误

问题现象：随便进入一个 pod 容器内部，使用 service ip:端口可以正常访问，但是使用 service 域名:端口就不通。

可能原因：查看 dns 日志一直没有变化，说明没有请求到达 dns。可能是修改了容器使用的安全组，拦截了 UDP 协议下 53 端口的通信。

解决方法：修改集群安全组，放开 UDP 协议的 53 端口规则。