云容器引擎 CCSE

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

云容器引擎（简称 CCSE）提供高度可扩展的、高性能的 Kubernetes 集群、一站式容器服务；获得信通院可信云《全栈容器云解决方案》认证，兼容主流国产化服务器和操作系统，取得全栈国产化适配认证证书。其整合了镜像、监控、日志、负载均衡、灰度/蓝绿、多种弹性策略、高效调度、集群插件、模板市场等基础能力，帮助企业快速构建和运行可弹性扩展的应用，实现业务的快速交付与持续创新。

1.2 产品优势

1.2.1 大规模实践

内部上云大规模应用落地，集群部署规模 400+，应用实例超过 8 万。

1.2.2 以应用为中心

结合微服务应用平台帮助用户快速开发、构建、部署和运维分布式应用，提供一站式容器服务。

1.2.3 简单易用

一键创建 Kubernetes 集群，基于控制台轻松地实现 Kubernetes 集群的扩容和缩容。

1.2.4 安全稳定

支持高可用部署，提供集群备份恢复插件和安全容器能力，保障应用安全运行。

1.3 版本比对

功能	云容器引擎	自建 Kubernetes
集群管理	通过控制台一键创建集群，支持创建跨 AZ 高可用的集群。 提供容器优化的 OS 镜像，提供稳定测试和安全加固的 Kubernetes 和 Docker 版本。 支持多集群管理，支持跨 AZ 高可用集群，支持集群联邦管理。	用户手动部署集群并自行开发 用户自行探索和开发。
应用管理	支持灰度发布，支持蓝绿发布。 支持应用监控、应用弹性伸缩。 内置模板市场，支持 Helm 应用一键部署；支持服务目录，简化云服务集成。	用户自行探索和开发。

网络管理	提供针对天翼云优化的高性能 VPC/ENI 网络插件，性能优于普通网络方案。 支持容器访问策略和容器带宽限制。	需要挑选社区网络插件进行适配。 用户自行探索和开发。
存储管理	支持天翼云盘挂载，提供标准的 CSI、FlexVolume 驱动。 支持存储卷自动创建、迁移。	用户自行探索和开发。
运维管理	支持 Kubernetes 新版本一键升级，支持集群组件生命周期管理 支持集群手动和自动弹性伸缩。 提供高性能日志采集 Agent，自动实现日志服务集成。	用户手动运维控制面。
服务保障	中国电信内部上云大规模应用落地，集群部署规模 400+，应用实例超过 8 万，单集群支持 5000 节点。 天翼云专业容器团队作为技术支持，为集群提供及时的稳定性和安全响应。	需要组建专门团队。
安全管理	支持镜像扫描/镜像签名。 支持容器运行时安全检测。	用户自行构建安全能力。

1.4 应用场景

1.4.1 微服务架构

企业应用通过微服务拆分和容器化改造后，推荐使用容器引擎 CCSE+容器镜像服务+微服务应用平台 MSAP+注册配置中心的产品组合，实现一站式云原生应用托管，加速企业业务迭代。

1.4.2 持续集成交付

提供从开发到上线一致的运行环境，借助第三方工具进行流水线开发测试，自动完成从代码变更到代码构建、测试、镜像构建和应用部署的 DevOps 完整流程。进行持续交付，替代传统部署方式，提高交付效率。

1.4.3 弹性伸缩

企业应用按业务流量自动扩缩容，不需要人工干预，预防流量激增时导致的业务不可用风险，并且减少对闲置资源的浪费和费用支出。

1.5 产品规格

云容器引擎 CCSE 提供集群专有版，用户可按需配置应用所需承载的主机规格，具体可见 2.2 计费模式说明。

1.6 基本概念

在使用云容器引擎 CCSE 前，需理解该产品所涉及的概念。本文为您介绍使用容器 CCSE 过程中遇到的常用名词的基本概念和简要描述，以便于您更好地理解 CCSE 产品。

- **集群**

集群指容器运行所需要的云资源组合，关联了若干服务器节点、负载均衡、专有网络等云资源。

CCSE 专有集群， 需要创建 1 个 Master（非高可用），或者 3/5 个 Master（高可用）节点，以及若干 Worker 节点，可对集群基础设施进行更细粒度的控制，需要自行规划、维护、升级服务器集群。

- **节点**

一台服务器（可以是虚拟机实例或者物理服务器）已经安装了 Docker Engine，可以用于部署和管理容器。容器 CCSE 的 Agent 程序会被安装到节点上并注册到一个集群上。

- **专有网络 VPC**

专有网络 VPC 是您自己独有的云上私有网络。您可以完全掌控自己的专有网络，例如选择 IP 地址范围、配置路由表和网关等，您可以在自己定义的专有网络中使用天翼云资源如云服务器、云数据库和负载均衡等。

- **安全组**

安全组是一种虚拟防火墙，具备状态检测和数据包过滤能力，用于在云端划分安全域。安全组是一个逻辑上的分组，由同一地域内具有相同安全保护需求并相互信任的实例组成。

- **应用目录**

应用目录功能集成了 Helm，提供了 Helm 的相关功能，并进行了相关功能扩展，例如提供图形化界面。

- **编排模板**

编排模板是一种保存 Kubernetes YAML 格式编排文件的方式。

- **Kubernetes**

Kubernetes 是一个开源平台，具有可移植性和可扩展性，用于管理容器化的工作负载和服务，简化了声明式配置和自动化。

- **容器（Container）**

打包应用及其运行依赖环境的技术，一个节点可运行多个容器。

- **镜像（Image）**

容器镜像是容器应用打包的标准格式，封装了应用程序及其所有软件依赖的二进制数据。

- **镜像仓库（Image Registry）**

容器镜像仓库是一种存储库，用于存储 Kubernetes 和基于容器应用开发的容器镜像。

- **管理节点（Master Node）**

管理节点是 Kubernetes 集群的管理者，运行着的服务包括 kube-apiserver、kube-scheduler、kube-controller-manager、etcd 组件，和容器网络相关的组件。

- **工作节点（Worker Node）**

工作节点是 Kubernetes 集群中承担工作负载的节点，可以是虚拟机也可以是物理机。工作节点承担实际的 Pod 调度以及与管理节点的通信等。一个工作节点上的服务包括 Docker 运行时环境、kubelet、Kube-Proxy 以及其它一些可选的组件。

- **命名空间（Namespace）**

命名空间为 Kubernetes 集群提供虚拟的隔离作用。Kubernetes 集群初始有 3 个命名空间，分别是默认命名空间 default、系统命名空间 kube-system 和 kube-public，除此以外，管理员可以创建新的命名空间以满足需求。

- **容器组（Pod）**

Pod 是 Kubernetes 部署应用或服务的最小的基本单位。一个 Pod 封装多个应用容器（也可以只有一个容器）、存储资源、一个独立的网络 IP 以及管理控制容器运行方式的策略选项。

- **副本控制器（Replication Controller，RC）**

RC 确保任何时候 Kubernetes 集群中有指定数量的 Pod 副本在运行。通过监控运行中的 Pod 来保证集群中运行指定数目的 Pod 副本。指定的数目可以是多个也可以是 1 个；少于指定数目，RC 就会启动运行新的 Pod 副本；多于指定数目，RC 就会终止多余的 Pod 副本。

- **副本集（ReplicaSet，RS）**

ReplicaSet（RS）是 RC 的升级版本，唯一区别是对选择器的支持，RS 能支持更多种类的匹配模式。副本集对象一般不单独使用，而是作为 Deployment 的理想状态参数使用。

- **工作负载（Workload）**

工作负载是在 Kubernetes 上运行的应用程序。工作负载包括以下几种类型：

工作负载类型	**描述**
无状态工作负载（Deployment）	无状态工作负载表示对 Kubernetes 集群的一次更新操作。适用于运行完全独立、功能相同应用的场景。
有状态工作负载（StatefulSet）	有状态工作负载支持应用部署、扩容、滚动升级时有序进行。如果希望使用存储卷为工作负载提供持久存储，可以使用 StatefulSet 作为解决方案的一部分。
守护进程集（DaemonSet）	守护进程集确保全部（或者某些）节点上运行一个 Pod。与 Deployment 不同，DaemonSet 会在指定的节点上都部署定义的 Pod，确保这些节点都运行守护进程 Pod。适用集群的日志、监控等部署场景。
任务（Job）	Job 指运行一次性的任务。您可以使用 Job 以并行的方式运行多个 Pod。
定时任务（CronJob）	CronJob 指根据规划时间周期性地运行反复的任务。适用于执行数据备份或者发送邮件的场景。
自定义资源（CustomResourceDefinitions，CRD）	在庞大的 Kubernetes 生态系统中，您可以通过 CRD 添加第三方工作负载资源。CRD 资源允许您定义定制资源。

- **标签（Label）**

Labels 的实质是附着在资源对象上的一系列 Key/Value 键值对，用于指定对用户有意义的对象的属性，标签对内核系统是没有直接意义的。标签可以在创建一个对象的时候直接赋予，也可以在后期随时修改，每一个对象可以拥有多个标签，但 key 值必须唯一。

- **服务（Service）**

Service 是 Kubernetes 的基本操作单元，是真实应用服务的抽象，每一个服务后面都有很多对应的容器来提供支持，通过 Kube-Proxy 的 ports 和服务 selector 决定服务请求传递给后端的容器，对外表现为一个单一访问接口。

- **路由（Ingress）**

Ingress 是授权入站连接到达集群服务的规则集合。您可以通过 Ingress 配置提供外部可访问的 URL、负载均衡、SSL、基于名称的虚拟主机等。通过 POST Ingress 资源到 API Server 的方式来请求 Ingress。Ingress Controller 负责实现 Ingress，通常使用负载均衡器，它还可以配置边界路由和其他前端，这有助于以高可用的方式处理流量。

- **配置项（ConfigMap）**

配置项可用于存储细粒度信息如单个属性，或粗粒度信息如整个配置文件或 JSON 对象。您可以使用配置项保存不需要加密的配置信息和配置文件。

- **保密字典（Secret）**

保密字典用于存储在 Kubernetes 集群中使用一些敏感的配置，例如密码、证书等信息。

- **卷（Volume）**

和 Docker 的存储卷有些类似，Docker 的存储卷作用范围为一个容器，而 Kubernetes 的存储卷的生命周期和作用范围是一个 Pod。每个 Pod 中声明的存储卷由 Pod 中的所有容器共享。

- **存储卷（Persistent Volume，PV）**

PV 是集群内的存储资源，类似节点是集群资源一样。PV 独立于 Pod 的生命周期，可根据不同的 StorageClass 类型创建不同类型的 PV。

- **存储卷声明（Persistent Volume Claim，PVC）**

PVC 是资源的使用者。类似 Pod 消耗节点资源一样，而 PVC 消耗 PV 资源。

- **存储类（StorageClass）**

存储类可以实现动态供应存储卷。通过动态存储卷，Kubernetes 将能够按照用户的需要，自动创建其所需的存储。

- **弹性伸缩（Autoscaling）**

弹性伸缩是根据业务需求和策略，经济地自动调整弹性计算资源的管理服务。典型的场景包含在线业务弹性、大规模计算训练、深度学习 GPU 或共享 GPU 的训练与推理、定时周期性负载变化等。支持的弹性伸缩服务如下表。

弹性伸缩维度	弹性伸缩分类	描述
调度层弹性	容器水平伸缩（HPA）	容器水平伸缩基于 CPU 使用率自动扩缩 Pod 数量。适用于 Deployment、StatefulSet 等实现了 scale 接口的对象。

	容器定时伸缩（CronHPA）	应对资源浪费的场景，提供 kubernetes-cronhpa-controller 组件，实现资源定时扩容。适用于 Deployment、StatefulSet 等实现了 scale 接口的对象。此外 CronHPA 提供了 HPA 对象的兼容能力，您可以同时使用 CronHPA 与 HPA。
	容器垂直伸缩（VPA）	容器垂直伸缩会基于 Pod 的资源使用情况自动为集群设置资源占用的限制，从而让集群将 Pod 调度到有足够资源的最佳节点上。容器垂直伸缩也会保持最初容器定义中资源 request 和 limit 的占比。适用于无法水平扩展的应用，通常是在 Pod 出现异常恢复时生效。
资源层弹性	节点自动伸缩	自动伸缩能力是通过节点自动伸缩组件实现的，支持多可用区、多实例规格、多种伸缩模式，满足不同的节点伸缩场景。全场景支持，适合在线业务、深度学习、大规模成本算力交付等。

- **可观测性（Observability）**

Kubernetes 可观测性体系包含监控和日志两部分，监控可以帮助开发者查看系统的运行状态，而日志可以协助问题的排查和诊断。

- **Helm**

Helm 是 Kubernetes 包管理平台。Helm 将一个应用的相关资源组织成为 Charts，然后通过 Charts 管理程序包。

- **节点亲和性（nodeAffinity）**

节点亲和性指通过 Worker 节点的 Label 标签控制 Pod 部署在特定的节点上。

- **污点（Taints）**

污点和节点亲和性相反，它使节点能够排斥一类特定的 Pod。

- **容忍（Tolerations）**

应用于 Pod 上，允许（但并不要求）Pod 调度到带有与之匹配的污点的节点上。

- **应用亲和性（podAffinity）**

应用亲和性决定应用 Pod 可以和特定 Pod 部署在同一拓扑域。例如，对于相互通信的服务，可通过应用亲和性调度，将其部署到同一拓扑域（例如同一个主机）中，以减少它们之间的网络延迟。

- **应用反亲和性（podAntiAffinity）**

应用反亲和性决定应用 Pod 不与特性 Pod 部署在同一拓扑域。例如，将一个服务的 Pod 分散部署到不同的拓扑域（例如不同主机）中，以提高服务本身的稳定性。

- **服务网格（Istio）**

Istio 是一个提供连接、保护、控制以及观测服务的开放平台，兼容社区 Istio 开源服务网格，用于简化服务的治理，包括服务调用之间的流量路由与拆分管理、服务间通信的认证安全以及网格可观测性能力。

1.7 使用限制

1.7.1 概述

CCSE 实例的使用限制主要包括功能限制、配额限制以及底层资源限制三大类。

1.7.2 功能限制

使用 CCSE 实例之前需要注意以下一些限制：

- 购买实例之前需要实名认证。
- 实例创建之后，暂不支持以下项：

变更集群 VPC；变更集群管理节点数量；变更集群网络插件；集群主机无法远程登录。

1.7.3 配额限制

管理节点数量	单节点最大 Pod 数	例外申请方式
1000	110	不可申请

1.7.4 底层资源限制

限制大类	限制项	用户限制
云服务器	操作系统	ctyunos2.0.1
	内核版本	5.4

2 计费说明

2.1 计费项及其计费方式

一套容器集群（专有版）实例包括集群管理、IaaS 云资源（如：弹性云主机、云硬盘、ELB 等），其中涉及计费的资源如下表所示。

计费项	计费项说明	计费方式
容器集群管理	管理 50 节点（非高可用）	包年/包月、按量计费
	管理 50 节点（高可用）	
	管理 1000 节点（高可用）	
IaaS 云资源（参照天翼云一类节点计费说明）	计算资源（vCPU 和内存）	
	云硬盘（GB）	
	ELB（绑定 API Server 使用）	

2.2 价格

容器集群管理费：

名称	计费方式	价格（按月）元/月	价格（按需）元/小时
集群管理（1 个控制节点）	按需/包周期	421.00	0.970000
集群管理（3 个控制节点）	按需/包周期	1263.00	2.910000
集群管理（5 个控制节点）	按需/包周期	4671.00	11.070000

计算资源（包含：Master 节点/Worker 节点）：

名称	CPU：vCPU	内存：GB	主机类型	计费方式	价格（按月）元/月	价格（按需）元/小时
s6.xlarge.2	4	8	s6	按需/包周期	452	0.619178
s6.xlarge.4	4	16	s6	按需/包周期	580	0.794521
s6.2xlarge.2	8	16	s6	按需/包周期	760	1.041096
s6.2xlarge.4	8	32	s6	按需/包周期	1016	1.391781
s6.4xlarge.2	16	32	s6	按需/包周期	1376	1.884932
s6.4xlarge.4	16	64	s6	按需/包周期	1888	2.586301
s7.xlarge.2	4	8	s7	按需/包周期	457	0.626027
s7.xlarge.4	4	16	s7	按需/包周期	588	0.805479
s7.2xlarge.2	8	16	s7	按需/包周期	770	1.054795
s7.2xlarge.4	8	32	s7	按需/包周期	1031	1.412329
s7.4xlarge.2	16	32	s7	按需/包周期	1397	1.913699

s7.4xlarge.4	16	64	s7	按需/包周期	1919	2.628767
s7.8xlarge.2	32	64	s7	按需/包周期	2650	3.630137
s7.8xlarge.4	32	128	s7	按需/包周期	3694	5.060274
c6.xlarge.2	4	8	c6	按需/包周期	564	0.772603
c6.xlarge.4	4	16	c6	按需/包周期	644	0.882192
c6.2xlarge.2	8	16	c6	按需/包周期	984	1.347945
c6.2xlarge.4	8	32	c6	按需/包周期	1144	1.567123
c6.3xlarge.2	12	24	c6	按需/包周期	1404	1.923288
c6.3xlarge.4	12	48	c6	按需/包周期	1644	2.252055
c6.4xlarge.2	16	32	c6	按需/包周期	1824	2.498630
c6.4xlarge.4	16	64	c6	按需/包周期	2144	2.936986
c6.6xlarge.2	24	48	c6	按需/包周期	2664	3.649315
c6.6xlarge.4	24	96	c6	按需/包周期	3144	4.306849
c6.8xlarge.2	32	64	c6	按需/包周期	3504	4.800000
c6.8xlarge.4	32	128	c6	按需/包周期	4144	5.676712

c6.16xlarge.2	64	128	c6	按需/包周期	6864	9.402740
c6.16xlarge.4	64	256	c6	按需/包周期	8144	11.156164
c7.xlarge.2	4	8	c7	按需/包周期	548	0.750685
c7.xlarge.4	4	16	c7	按需/包周期	660	0.904110
c7.2xlarge.2	8	16	c7	按需/包周期	952	1.304110
c7.2xlarge.4	8	32	c7	按需/包周期	1176	1.610959
c7.3xlarge.2	12	24	c7	按需/包周期	1356	1.857534
c7.3xlarge.4	12	48	c7	按需/包周期	1692	2.317808
c7.4xlarge.2	16	32	c7	按需/包周期	1760	2.410959
c7.4xlarge.4	16	64	c7	按需/包周期	2208	3.024658
c7.6xlarge.2	24	48	c7	按需/包周期	2568	3.517808
c7.6xlarge.4	24	96	c7	按需/包周期	3240	4.438356
c7.8xlarge.2	32	64	c7	按需/包周期	3376	4.624658
c7.8xlarge.4	32	128	c7	按需/包周期	4272	5.852055
c7.12xlarge.2	48	96	c7	按需/包周期	4992	6.838356

c7.12xlarge.4	48	192	c7	按需/包周期	6336	8.679452
c7.16xlarge.2	64	128	c7	按需/包周期	6608	9.052055
c7.16xlarge.4	64	256	c7	按需/包周期	8400	11.506849
c7.24xlarge.2	96	192	c7	按需/包周期	9840	13.479452
c7.24xlarge.4	96	384	c7	按需/包周期	12528	17.161644
m6.xlarge.8	4	32	m6	按需/包周期	804	1.101370
m6.2xlarge.8	8	64	m6	按需/包周期	1464	2.005479
m6.3xlarge.8	12	96	m6	按需/包周期	2124	2.909589
m6.4xlarge.8	16	128	m6	按需/包周期	2784	3.813699
m6.6xlarge.8	24	192	m6	按需/包周期	4104	5.621918
m6.8xlarge.8	32	256	m6	按需/包周期	5424	7.430137
m6.16xlarge.8	64	512	m6	按需/包周期	10704	14.663014
m7.xlarge.8	4	32	m7	按需/包周期	826	1.131507
m7.2xlarge.8	8	64	m7	按需/包周期	1507	2.064384
m7.3xlarge.8	12	96	m7	按需/包周期	2189	2.998630

m7.4xlarge.8	16	128	m7	按需/包周期	2871	3.932877
m7.6xlarge.8	24	192	m7	按需/包周期	4234	5.800000
m7.8xlarge.8	32	256	m7	按需/包周期	5598	7.668493
m7.12xlarge.8	48	384	m7	按需/包周期	8325	11.404110
m7.16xlarge.8	64	512	m7	按需/包周期	11052	15.139726
m7.24xlarge.8	96	768	m7	按需/包周期	16505	22.609589
GPU 直通 16C64G	16	64	pi7	按需/包周期	4591	6.289041
GPU 直通 32C128G	32	128	pi7	按需/包周期	9039	12.382192
GPU 直通 64C256G	64	256	pi7	按需/包周期	17934	24.567123
GPU 虚拟化 8C32G	8	32	g7	按需/包周期	2177	2.982192
GPU 虚拟化 16C64G	16	64	g7	按需/包周期	4211	5.768493
GPU 虚拟化 32C128G	32	128	g7	按需/包周期	8277	11.338356
hc1.xlarge.2	4	8	hc1	按需/包周期	612	0.838356
hc1.xlarge.4	4	16	hc1	按需/包周期	724	0.991781
hc1.2xlarge.2	8	16	hc1	按需/包周期	1080	1.479452

hc1.2xlarge.4	8	32	hc1	按需/包周期	1304	1.786301
hc1.4xlarge.2	16	32	hc1	按需/包周期	2016	2.761644
hc1.4xlarge.4	16	64	hc1	按需/包周期	2464	3.375342
hc1.8xlarge.2	32	64	hc1	按需/包周期	3888	5.326027
hc1.8xlarge.4	32	128	hc1	按需/包周期	4784	6.553425
hm1.xlarge.8	4	32	hm1	按需/包周期	948	1.298630
hm1.2xlarge.8	8	64	hm1	按需/包周期	1752	2.400000
hm1.4xlarge.8	16	128	hm1	按需/包周期	3360	4.602740
hs1.xlarge.2	4	8	hs1	按需/包周期	452	0.619178
hs1.xlarge.4	4	16	hs1	按需/包周期	580	0.794521
hs1.2xlarge.2	8	16	hs1	按需/包周期	760	1.041096
hs1.2xlarge.4	8	32	hs1	按需/包周期	1016	1.391781
hs1.4xlarge.2	16	32	hs1	按需/包周期	1376	1.884932
hs1.4xlarge.4	16	64	hs1	按需/包周期	1888	2.586301
kc1.xlarge.2	4	8	kc1	按需/包周期	680	0.931507

kc1.xlarge.4	4	16	kc1	按需/包周期	800	1.095890
kc1.2xlarge.2	8	16	kc1	按需/包周期	1216	1.665753
kc1.2xlarge.4	8	32	kc1	按需/包周期	1456	1.994521
kc1.4xlarge.2	16	32	kc1	按需/包周期	2288	3.134247
kc1.4xlarge.4	16	64	kc1	按需/包周期	2768	3.791781
kc1.8xlarge.2	32	64	kc1	按需/包周期	4432	6.071233
kc1.8xlarge.4	32	128	kc1	按需/包周期	5392	7.386301
km1.xlarge.8	4	32	km1	按需/包周期	1040	1.424658
km1.2xlarge.8	8	64	km1	按需/包周期	1936	2.652055
km1.4xlarge.8	16	128	km1	按需/包周期	3728	5.106849
km1.8xlarge.8	32	256	km1	按需/包周期	7312	10.016438
ks1.xlarge.2	4	8	ks1	按需/包周期	472	0.646575
ks1.xlarge.4	4	16	ks1	按需/包周期	592	0.810959
ks1.2xlarge.2	8	16	ks1	按需/包周期	800	1.095890
ks1.2xlarge.4	8	32	ks1	按需/包周期	1040	1.424658

ks1.4xlarge.2	16	32	ks1	按需/包周期	1456	1.994521
Ks1.4xlarge.4	16	64	ks1	按需/包周期	1936	2.652055
fc1.xlarge.2	4	8	fc1	按需/包周期	564	0.772603
fc1.xlarge.4	4	16	fc1	按需/包周期	644	0.882192
fc1.2xlarge.2	8	16	fc1	按需/包周期	984	1.347945
fc1.2xlarge.4	8	32	fc1	按需/包周期	1144	1.567123
fc1.4xlarge.2	16	32	fc1	按需/包周期	1824	2.498630
fc1.4xlarge.4	16	64	fc1	按需/包周期	2144	2.936986
fm1.xlarge.8	4	32	fm1	按需/包周期	832	1.139726
fm1.2xlarge.8	8	64	fm1	按需/包周期	1520	2.082192
fm1.4xlarge.8	16	128	fm1	按需/包周期	2896	3.967123
fs1.xlarge.2	4	8	fs1	按需/包周期	452	0.619178
fs1.xlarge.4	4	16	fs1	按需/包周期	580	0.794521
fs1.2xlarge.2	8	16	fs1	按需/包周期	760	1.041096
fs1.2xlarge.4	8	32	fs1	按需/包周期	1016	1.391781

fs1.4xlarge.2	16	32	fs1	按需/包周期	1376	1.884932
fs1.4xlarge.4	16	64	fs1	按需/包周期	1888	2.586301
physical.s5.2xlarge1	2 路 28 核	512	physical.s5.2xlarge1	包周期	15696	21.501370
physical.s5.2xlarge4	2 路 28 核	512	physical.s5.2xlarge4	包周期	22912	31.386301
physical.pi7.2xlarge1	2 路 32 核	768	physical.pi7.2xlarge1	包周期	30334.42	41.554000

备注：以上单节点主机均包含 1 个系统盘费用：超高 IO（SSD）120GB

数据盘：

名称	磁盘类型	计费方式	价格（按月）元/月	标准价格（按小时）元/小时
普通 IO	SATA	按需/包周期	0.3	0.000500
高 IO	SAS	按需/包周期	0.4	0.000900
超高 IO	SSD	按需/包周期	1.2	0.001700

ELB：

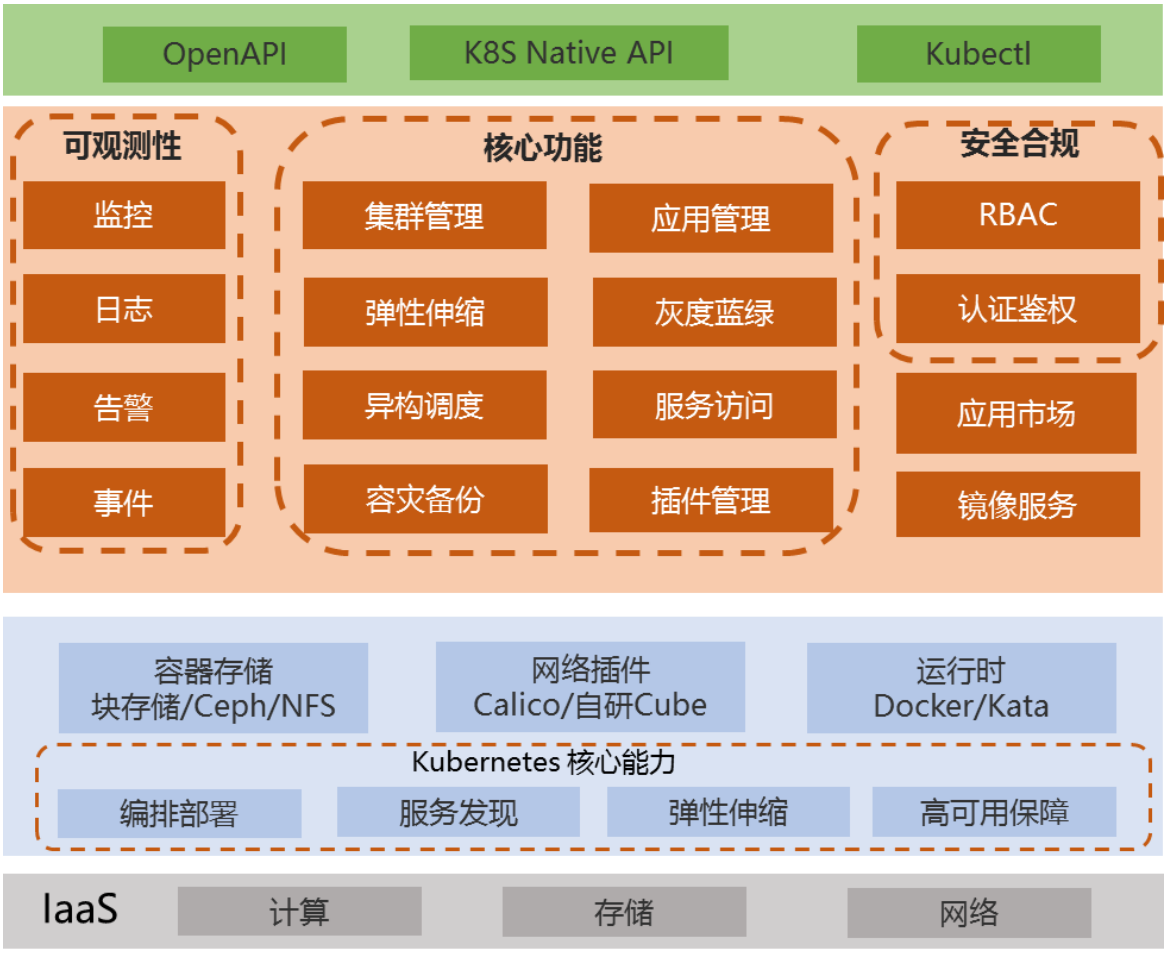
名称	计费方式	价格（按月）元/月	标准价格（按小时）元/小时
负载均衡-标准型 I	按需/包周期	360	0.493151
负载均衡-标准型 II	按需/包周期	720	0.986301
负载均衡-增强型 I	按需/包周期	1000	1.369863
负载均衡-增强型 II	按需/包周期	1300	1.780822

负载均衡-高阶型 I	按需/包周期	1800	2.465753
负载均衡-高阶型 II	按需/包周期	2500	3.424658
负载均衡-超强型 I	按需/包周期	4500	6.164384
负载均衡-超强型 II	按需/包周期	10000	13.698630

3 产品功能

3.1 产品架构

3.1.1 产品功能架构



3.1.2 技术特性

- **强大的集群管理**

高性能自研网络插件，VM 与容器直连互通，性能提升 20%

丰富的集群插件，开箱即用

基于 ebpf 实现无侵入监控，实现集群拓扑，网络性能感知

- **一站式容器应用管理**

支持原生 5 种类型工作负载

内置应用模板，支持一键部署 Helm 应用

支持灰度发布，蓝绿发布，应用弹性伸缩

- **极致弹性&高效调度**

支持 HPA/CronHPA 伸缩策略

支持基于历史指标/事件驱动的弹性伸缩

提供负载感知调度，解决原生 k8s 调度不均问题

- **企业级的安全稳定**

支持 3Master 高可用，镜像服务高可用能力

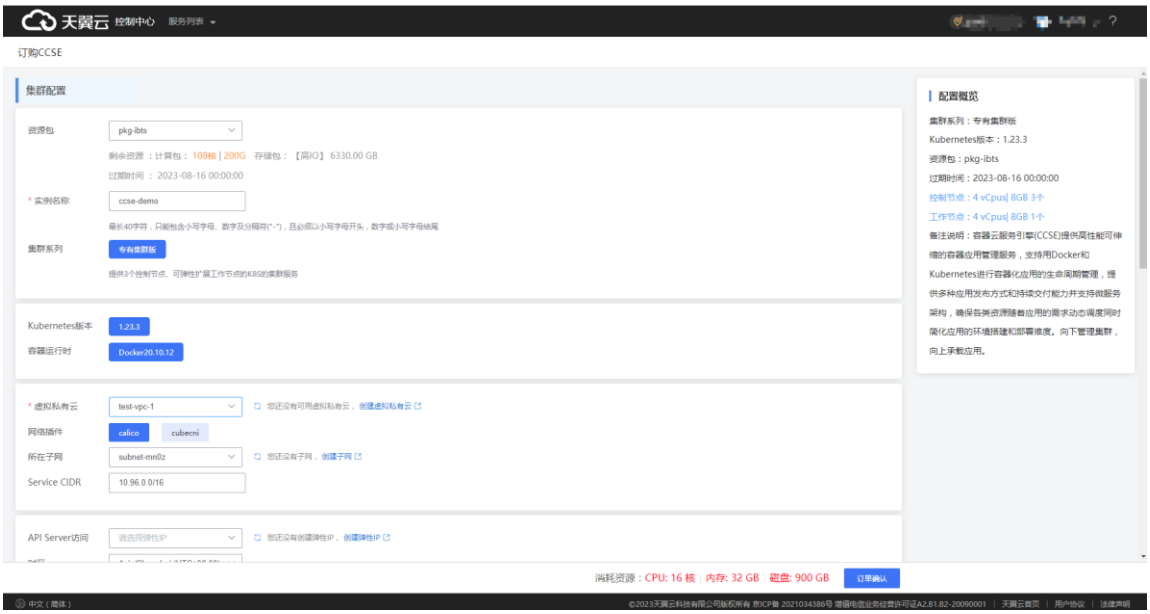
提供集群备份恢复插件，支持多种存储介质

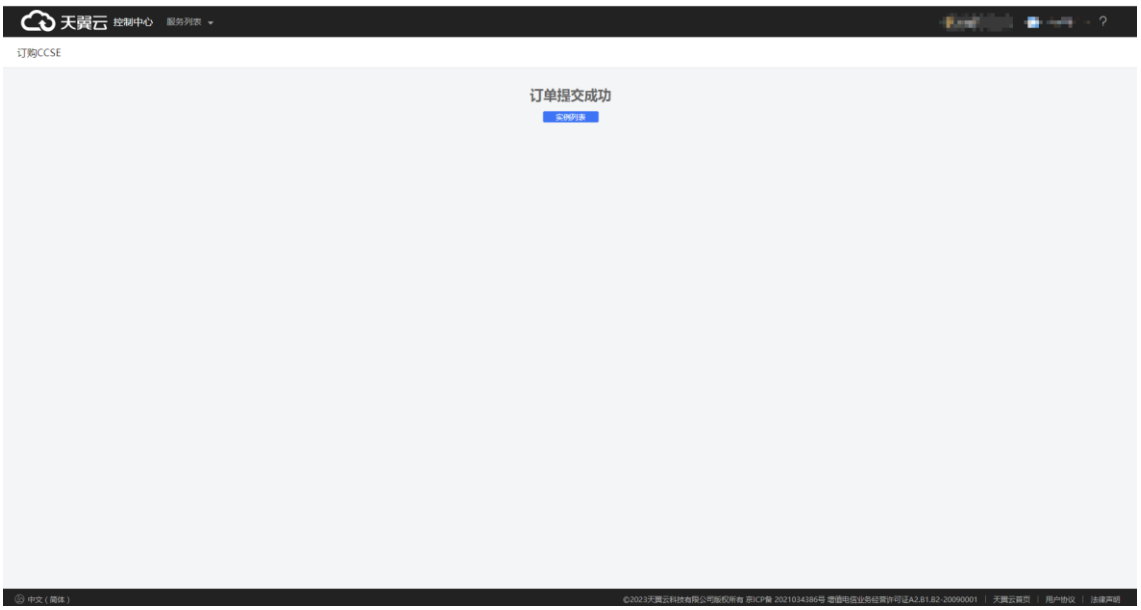
支持安全容器，提供镜像签名和镜像扫描

3.2 产品功能

3.2.1 开通

未开通 CCSE 时，初次进入 CCSE，会自动跳转到开通页面，如下图所示：

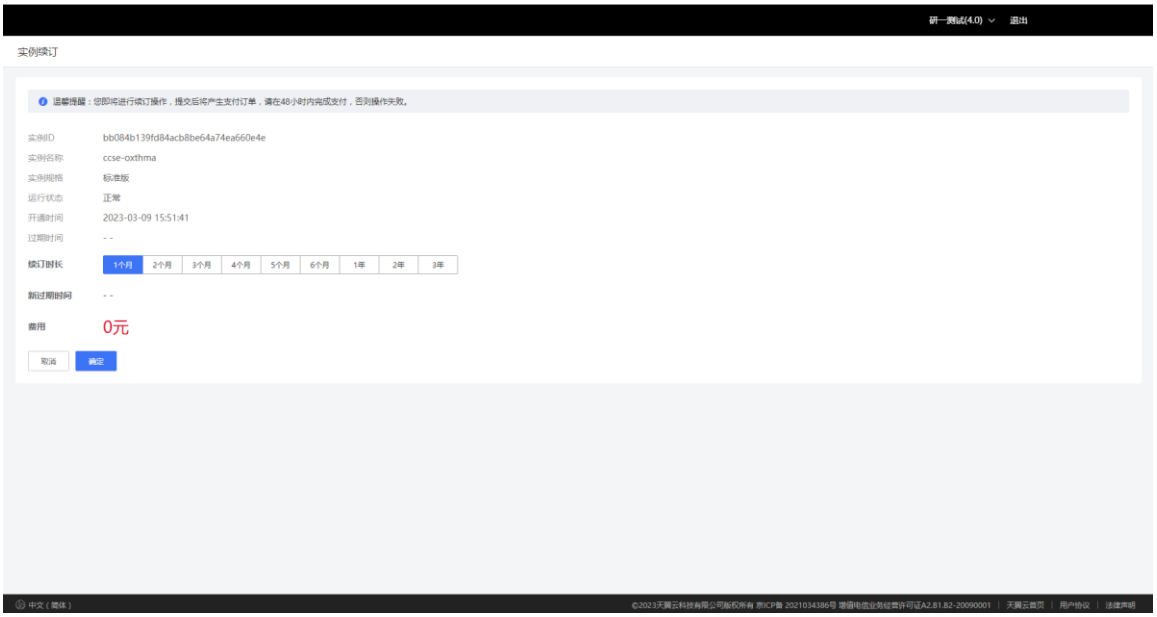




生成订单，成功付款后，再此进入 CCSE 控制台，在总览页面可以查看到开通成功信息。

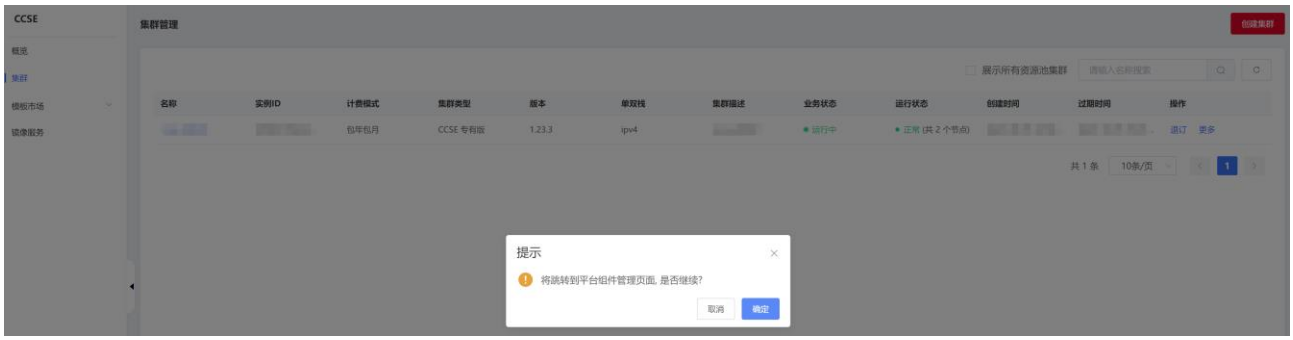
3.2.2 续订

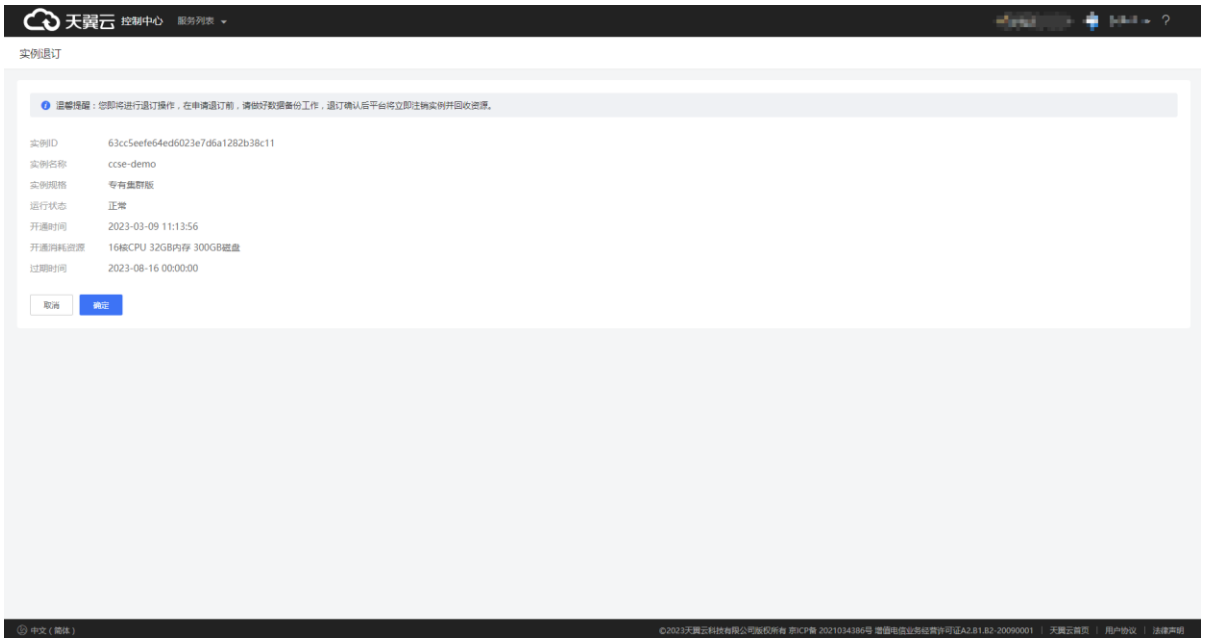
进入 CCSE 控制台页面，找到对应的集群实例，点击“更多”，可以进行实例的续订（示例）。

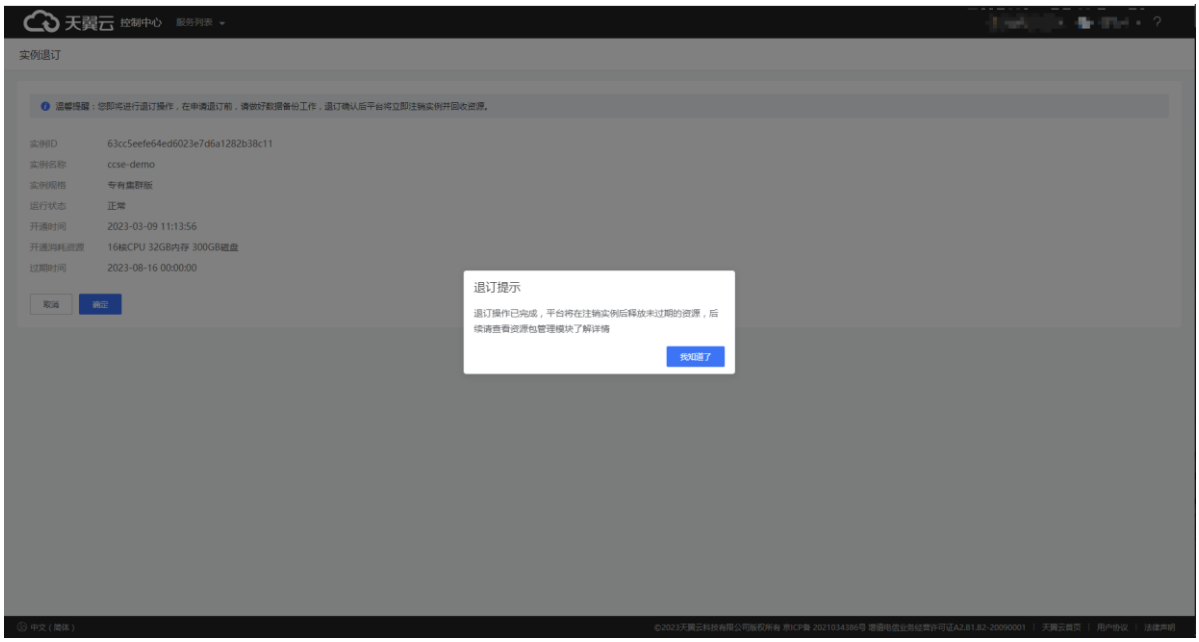


3.2.3 退订

进入 CCSE 控制台页面，找到对应的集群实例，点击“更多”，可以进行实例的退订。







3.2.4 CCSE 集群

Kubernetes 是主流的开源容器编排平台，用于管理容器化应用和服务。本文介绍 CCSE 集群的功能、类型和使用限制等。

云容器引擎 CCSE 面向用户提供 Kubernetes 集群服务，当前主要包括：

CCSE 集群：适合大多数业务场景，是一种最通用的 Kubernetes 集群。

3.2.4.1 使用限制

使用天翼云云容器引擎 CCSE 产品前，需要注意以下使用限制：

- 购买云容器引擎 CCSE 实例前，需要注册天翼云账号。
- 在创建 CCSE 集群以后，暂不支持以下项：

变更集群的 VPC；变更容器网络插件；在不同命名空间下迁移应用。

- CCSE 集群中 ECS 实例的限制如下：
  - 由于 ECS 等底层依赖产品配额及库存限制，创建、扩容集群，或者自动弹性扩容集群时，可能只有部分节点创建成功。
  - 因为 ECS 配额及库存限制可能导致创建失败，已创建出来的包年包月实例无法释放，因此只能加入已有集群进行使用。
  - ECS 规格要求：CPU 大于等于 4 核，且内存大于等于 8 GiB。

CCSE 集群配额限制

集群类型	单账号最大集群	单集群最大节点数	单节点最大 Pod 数	例外申请方式
CCSE 专有版	无限制	1000	128	不可申请

备注：单节点最大 Pod 数仅适用于 Calico 网络模式且不支持申请提高。如果集群使用 Cubecni 网络，单节点最大 Pod 数由该节点可分配 IP 总数决定。

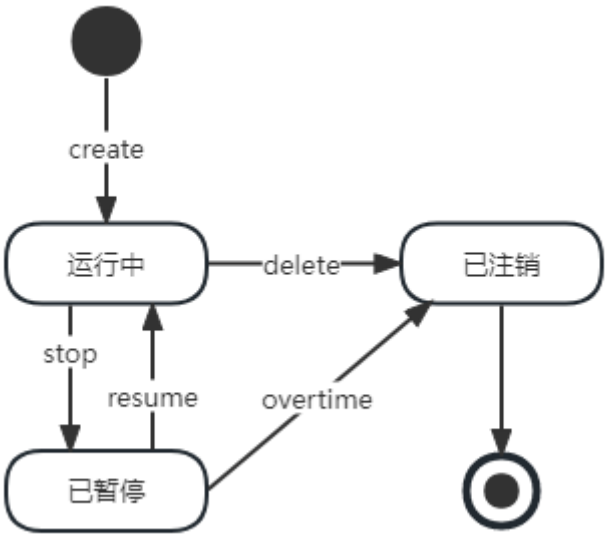
3.2.4.2 集群生命周期

CCSE 集群在不同状态下的含义和集群的状态流转图如下。

集群状态说明

状态	说明
运行中（1）	成功申请集群云资源。
已暂停（2）	集群欠费后处于停机状态，无法正常访问。
已注销（3）	集群主动退订或者未在规定时间内续费。

集群状态流转



3.2.4.3 功能入口

CCSE 集群支持的功能如下表

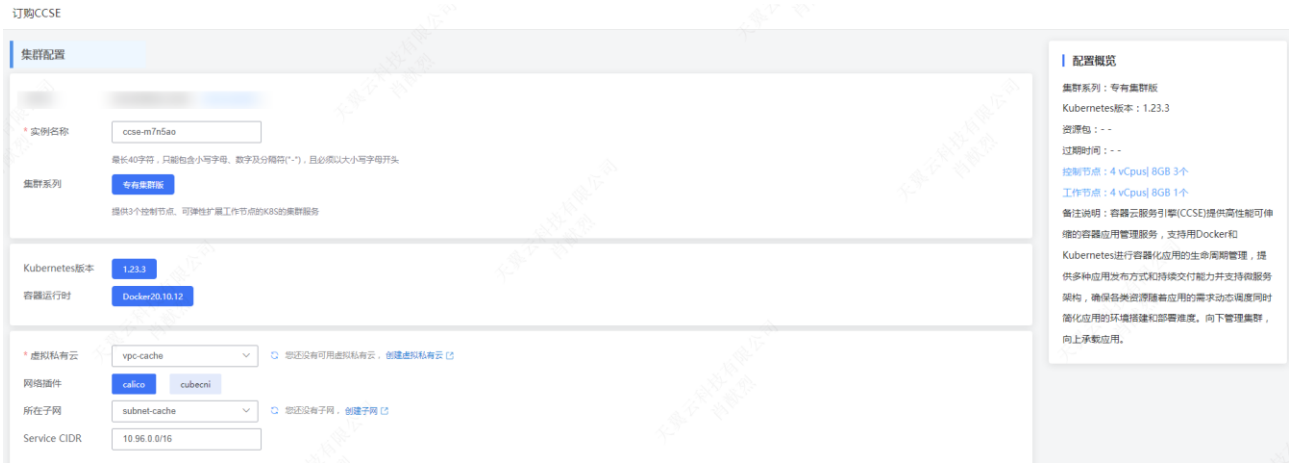
功能	描述
集群管理	■ 集群创建：您可根据需求创建 CCSE 集群，选择类型丰富的工作节

	点，并进行灵活的自定义配置。 ■ 弹性伸缩：通过控制台垂直扩缩容来快速应对业务波动，同时支持服务级别的亲和性策略和横向扩展。 ■ 授权管理：支持子账号授权和 RBAC 权限管理。
应用管理	■ 应用创建：支持多种类型应用，从镜像、模版的创建，支持环境变量、应用健康、数据盘、日志等相关配置。 ■ 应用全生命周期：支持应用查看、更新、删除，应用历史版本回滚、应用事件查看、应用滚动升级、应用替换升级。 ■ 应用调度：支持节点间亲和性调度、应用间亲和性调度、应用间反亲和性调度三种策略。 ■ 应用伸缩：支持手动伸缩应用容器实例，HPA 自动伸缩策略。 ■ 应用发布：支持灰度发布和蓝绿发布。 ■ 应用备份和恢复：支持对 Kubernetes 应用进行备份和恢复。
存储	■ 存储插件：支持 Flexvolume 以及 CSI 存储插件。 ■ 存储卷和存储声明： ■ 支持创建存储卷。 ■ 支持持久化存储卷声明（PVC）挂接存储卷。 ■ 支持存储卷的动态创建和迁移。 ■ 支持以脚本方式查看和更新存储卷和存储声明。
网络	■ 支持 Calico 容器网络和 Cubecni 容器网络。

	■ 支持定义 Sevice 和 Pod 的 CIDR。 ■ 支持 NetworkPolicy。 ■ 支持路由 Ingress。 ■ 支持服务发现 DNS。
运维与安全	■ 可观测性： ■ 监控：支持集群、节点、应用、容器实例层面的监控；支持 prometheus 插件。 ■ 日志：支持集群日志查看；支持应用日志采集；支持容器实例日志查看。 ■ 安全中心：支持运行时刻的安全策略管理，应用安全配置巡检和运行时刻的安全监控和告警，提升容器安全整体纵深防御能力。
开发者服务	■ API：提供 OpenAPI 和社区原生 API。

3.2.4.3.1 集群创建

可根据需求创建 CCSE 集群，并进行灵活的自定义配置。控制节点及工作节点均支持选择多种规格的虚机实例；集群支持多种网络插件及自定义集群网络，容器网络可以灵活配置。更多信息可参考创建一个集群。

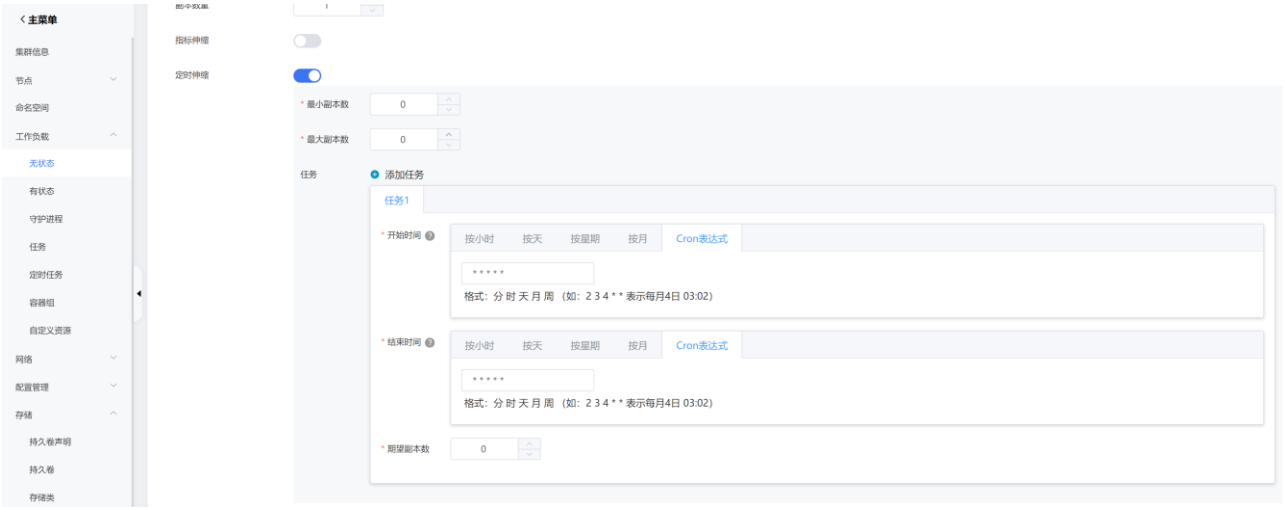


3.2.4.3.2 弹性伸缩

CCSE 集群支持两个层次的弹性伸缩，包括：

- 通过控制台垂直扩缩容来快速应对业务波动，其中由分为手动和自动两种方式。
- 支持服务级别的自动伸缩，其中由分为 HPA、定时等方式

HPA



定时伸缩

3.2.4.3.3 授权管理

支持 Kubernetes 证书管理及 RBAC 权限管理，用户获取证书后可通过 kubectl 直接操作集群，主用户可对子用户权限进行管理，控制子用户可访问的资源。详细见“授权”。

3.2.4.3.4 应用创建

支持多种类型应用，支持 Kubernetes Deployment、StatuefulSet 等多种资源；支持应用灵活配置，用户可自定义环境变量、应用监控、数据盘挂载、安全上下文等多种配置。应用支持从模板市场快速创建，支持用户上传模板到模板市场。主要包括以下操作：

- 创建方式，包括 YAML，负载配置界面，模板
- 支持工作负载类型，包括 Deployment、StatefulSet、DaemonSet、Job、CornJob 等
- 配置项：环境变量、应用监控、数据盘挂载、安全上下文等。


主菜单
集群管理 / 无状态
新增
新增YAML
批量操作
命名空间
请输入工作负载名称进行搜索
名称
类型
运行/期望Pod数量
镜像
创建时间
操作
暂无数据
共 0 条
10条/页
集群信息
节点
命名空间
工作负载
无状态
有状态
守护进程
任务
定时任务
容器组
自定义资源


主菜单
集群管理 / 无状态
配置项 / 新增Yaml
取消
保存
apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-deployment
labels:
app: nginx
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
spec:
containers:
- name: nginx
image: nginx:1.14.2
ports:
- containerPort: 80

YAML


CCSE
模板市场 / 模板市场
上传模板
镜像服务实例:
请选择
命名空间:
全部命名空间
请输入模板名称搜索
名称
命名空间
创建时间
操作
暂无数据
共 0 条
10条/页
前往
页
概览
集群
模板市场
模板实例
镜像服务

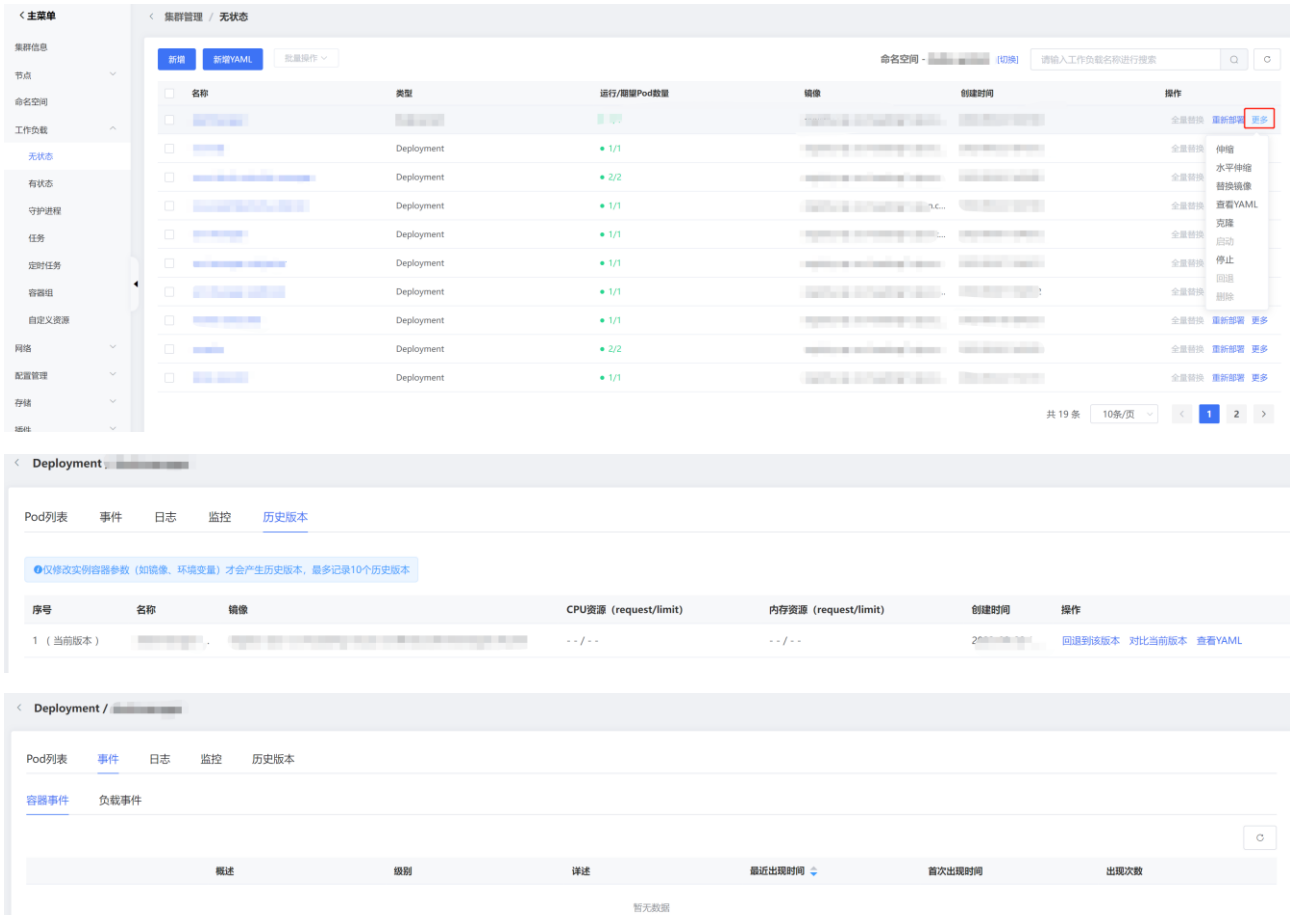
模板


主菜单
新建 Deployment
* Deployment名称
请输入
最长63个字符，由小写字母、数字、"-"组成，以字母开头，以字母或数字结尾
数据卷（选填）
卷名称
卷配置
暂无数据
添加数据卷
为容器提供存储，目前支持临时路径、主机路径、配置文件、本地卷（Local PV）、NFS、Ceph，还需挂载到容器的指定路径中。
副本数量
1
指标伸缩
定时伸缩
集群信息
节点
命名空间
工作负载
无状态
有状态
守护进程
任务
定时任务
容器组
自定义资源
网络

负载界面配置

3.2.4.3.5 应用全生命周期

CCSE 集群支持应用查看、更新、删除，应用历史版本回滚、应用事件查看、应用滚动升级、应用替换升级。

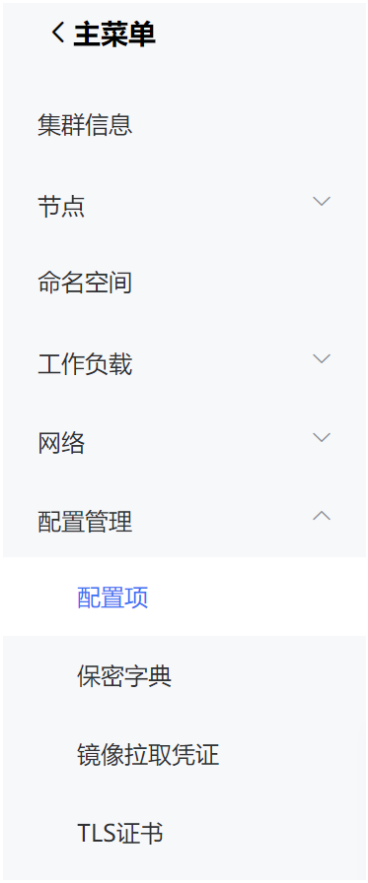


3.2.4.3.6 应用调度

支持节点间亲和性调度、应用间亲和性调度、应用间反亲和性调度三种策略

3.2.4.3.7 应用配置

CCSE 集群支持创建配置项或保密字典形式的配置，配置支持可视化和 YAML 两种编辑形式。



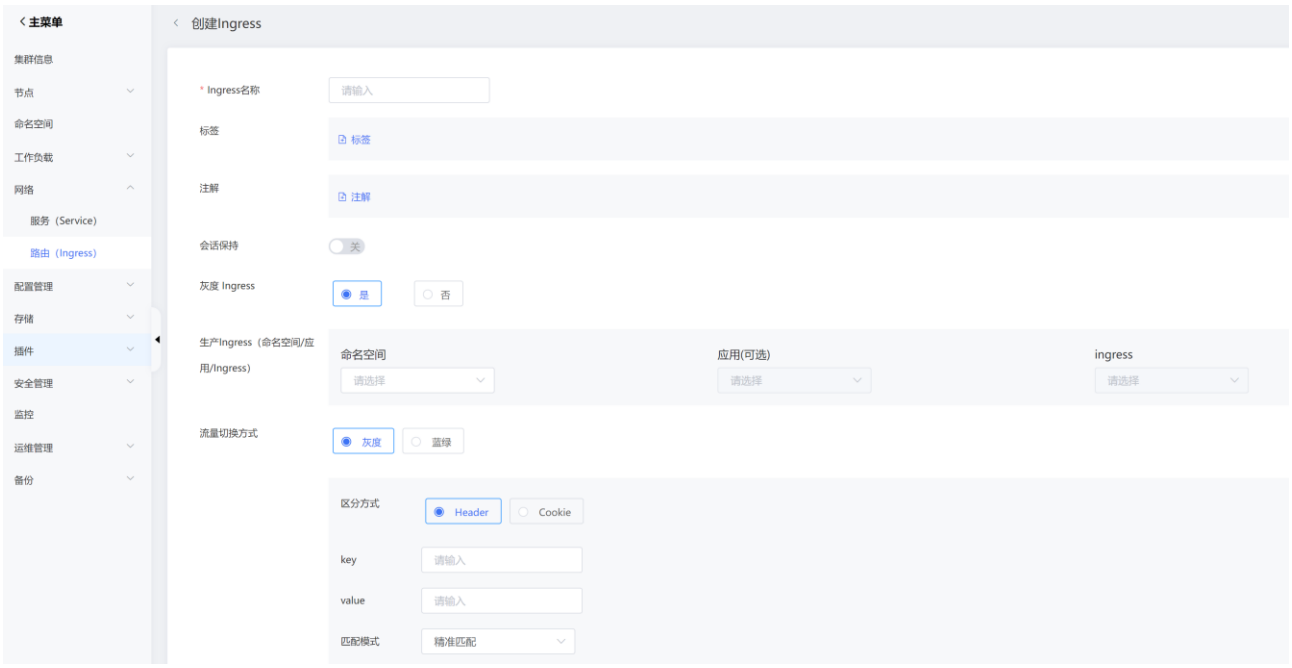
配置支持以数据见的形式挂载到容器目录或导入成容器的环境变量。

3.2.4.3.8 应用伸缩

CCSE 支持用户手动伸缩应用容器实例，HPA 自动伸缩策略和定时自动伸缩。详见“弹性伸缩”。

3.2.4.3.9 应用发布

CCSE 集群支持灰度发布和蓝绿发布，目前通过 Ingress 来实现。



3.2.4.3.10 应用备份和恢复

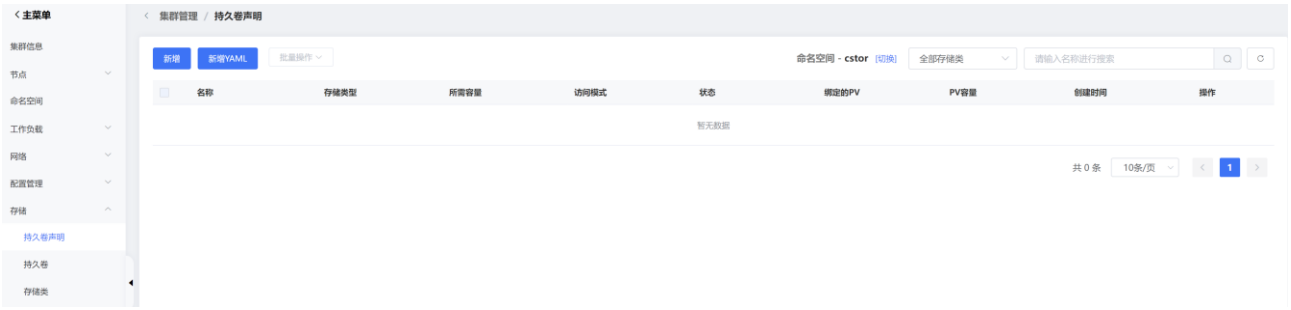
CCSE 集群的应用备份是通过集群备份实现的，支持到命名空间维度和资源类型。

3.2.4.3.11 存储插件

CCSE 集群支持 Flexvolume 以及 CSI 存储插件。当前 CCSE 集群尚未提供自研的插件，用户上传存储提供商的 chart 包到模板市场，并通过模板市场发布，以此达到使用外部存储的能力。

3.2.4.3.12 存储卷和存储声明

CCSE 集群支持持久卷声明、持久卷和存储类的管理，支持包括新增、查看、更改和删除的操作。



3.2.4.3.13 Service 管理

CCSE 集群支持用户创建和管理不同类型的 Service，实现 4 层的流量转发，详见文档“网络”。

3.2.4.3.14 Ingress 管理

CCSE 集群支持用户创建和管理 Ingress，实现 4-7 层的流程转发，详见文档“网络”。

3.2.4.3.15 NetworkPolicy 管理

CCSE 集群支持用户设置 NetworkPolicy，支持增加、查看和删除等操作。

3.2.4.3.16 监控

CCSE 集群提供集群、节点、应用等维度的监控的能力。详见可观性性体系。

3.2.4.3.17 日志

CCSE 集群对接了 ARMS-日志，ARMS-日志提供采集、存储、检索等能力。详见可观测性体系。

3.2.4.3.18 策略管理

CCSE 基于 OPA Gatekeeper 实现了官方的策略管理插件，代替已被淘汰的 Pod Security Policy（PSP）。详见安全体系。

3.2.4.4 开源项目

开源项目扩展了 Kubernetes 集群的功能。本文介绍天翼云云容器引擎 Kubernetes 版主要使用的开源项目。

项目分类	项目名称	项目简介	项目地址	参考文档
核心组件	cloud-provider	cloud-provider 是 Kubernetes 官方提供的一种接入云产商平台的方式，云产商只需要专注于与云产品对接的逻辑，它允许你将 K8S 集群连接到云提供商的 API 之上，并将与该云平台交互的组件同与你的集群交互的组件分离开来	cloud-provider	cloud-controller-manager
网络	NGINX Ingress Controller	作为反向代理服务器，提供 4 层和 7 层负载均衡能力。		
资源优化				

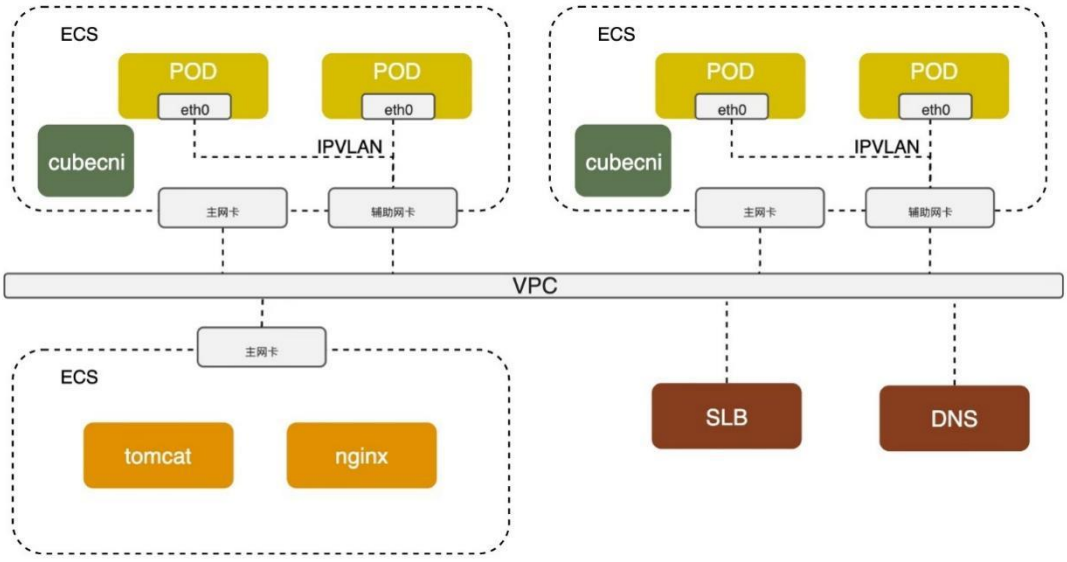
弹性	Cluster Autoscaler	Cluster Autoscaler是一个满足特定条件下能自动扩缩容集群节点的工具，能在资源不足时自动扩容，节点资源空闲时自动缩容	cluster-autoscaler	cluster-autoscaler document
安全	Open Policy Agent Gatekeeper	与 OPA Constraint Framework 集成在一起，使用户能够通过配置（而不是代码）自定义控制许可，用来实施基于 CRD 的策略，并可以可靠地共享已完成声明配置的策略	gatekeeper	gatekeeper document
调度	Scheduler Plugins	基于 scheduler framework 的	scheduler-plugins	scheduler-plugins

		Kubernetes的调度插件		document

3.2.5 网络

天翼云云容器引擎 Kubernetes 版 CCSE 集成 Kubernetes 网络、天翼云 VPC、天翼云 ELB，提供稳定高性能的容器网络。本文介绍 CCSE 集群网络及天翼云网络底层基础设施的重要概念，如容器网络 CNI、Service、Ingress、提供服务发现能力的 DNS 等。您可以通过了解这些概念，更合理地设计应用部署模型和网络访问的方式。

3.2.5.1 容器网络 CNI



容器化应用会在同一个节点上部署多个业务，而每个业务都需要自己的网络空间。为避免与其他业务网络冲突，需要 Pod 有自己独立的网络空间，而 Pod 中应用需要和其他网络进行通信，就需要 Pod 能够跟不同的网络互相访问。可见容器网络特点如下：

- 每个 Pod 都拥有自己独立的网络空间和 IP 地址。不同 Pod 的应用可以监听同样的端口而不冲突。
- Pod 可以通过各自的 IP 地址互相访问。

  集群中 Pod 可以通过它独立的 IP 地址与其他应用互通：

  - 同一个集群中，Pod 之间相互访问。
  - Pod 直接访问同一个 VPC 下的 ECS。
  - 同一个 VPC 下的 ECS 直接访问 Pod。

1、Cubecni

Cubecni 网络模式采用的是云原生的网络方案，直接基于天翼云的虚拟化网络中的弹性网卡资源来构建的容器网络。Pod 会通过弹性网卡资源直接分配 VPC 中的 IP 地址，而不需要额外指定虚拟 Pod 网段。

Cubecni 网络模式的特点是：

- 容器和虚拟机在同一层网络，便于业务云原生化迁移。
- 不依赖封包或者路由表，分配给容器的网络设备本身可以用来通信。
- 集群节点规模不受路由表或者封包的 FDB 转发表等配额限制。
- 不需要额外为容器规划 Overlay 的网段，多个集群容器之间只要设置安全组开放端口就可以互相通信。
- 可以直接把容器挂到 SLB 后端，无需在节点上使用 NodePort 进行转发。

- NAT 网关可以对容器做 SNAT，无需节点上对容器网段做 SNAT：容器访问 VPC 内资源，所带的源 IP 都是容器 IP，便于审计；容器访问外部网络不依赖 conntrack SNAT，降低失败率。
- Cubecni 网络模式支持通过网络策略（NetworkPolicy）配置 Pod 间网络访问的规则。

  网络策略（NetworkPolicy）是一种关于 Pod 间及 Pod 与其他网络端点间所允许的通信规则的规范。NetworkPolicy 资源使用标签选择 Pod，并定义选定 Pod 所允许的通信规则。

2、Calico

Calico 网络模式中 Pod 的网段独立于 VPC 的网段。

Calico 网络模式的特点是：

- 性能几乎无损。
- Pod 网段是独立于 VPC 的虚拟网段。

3.2.5.2 CCSE 常见网络能力

分类	常见网络能力	网络插件	
		Cubecni	Calico
网络配置管理	IPv4/IPv6 双栈	不支持	支持
	节点维度的网络配置	不支持	不支持

	Pod 维度的网络配置	支持	支持
	Pod 固定 IP	不支持	不支持
	Pod QoS	不支持	不支持
	网络策略 Network Policy	支持	支持
	设置 Pod 安全组	不支持	不支持
	扩容 Pod 网段	不支持	不支持
	使用 VPC 的多路由表功能	支持	不支持
南北向访问	Pod 访问公网	支持	支持
	从公网直接访问 Pod	不支持	不支持
	使用 LoadBalancer 服务	支持	不支持
	使用 Ingress	支持	支持

3.2.5.3 Service

由于云原生的应用，通常需要敏捷的迭代和快速的弹性，且单一容器和其相关的网络资源的生命非常短暂，所以需要固定的访问地址，以及自动负载均衡实现快速的业务弹性。CCSE 采用 Service 方式为一组容器提供固定的访问入口，并对这一组容器做负载均衡。实现原理如下：

- 创建 Service 对象时，CCSE 会分配一个相对固定的 Service IP 地址。
- 通过字段 selector 选择一组容器，以将这个 Service IP 地址和端口负载均衡到这一组容器 IP 和端口上。

Service 网络支持以下多种模式分别对接不同来源和类型的客户端的访问：

- ClusterIP

ClusterIP 类型的 Service 用于集群内部的应用间访问，如果您的应用需要暴露到集群内部提供服务，需使用 ClusterIP 类型的 Service 暴露。

- NodePort

NodePort 类型的 Service 将集群中部署的应用向外暴露，通过集群节点上的一个固定端口暴露出去，这样在集群外部就可以通过节点 IP 和这个固定端口来访问。

- LoadBalancer

LoadBalancer 类型的 Service 同样是将集群内部部署的应用向外暴露，不过它是通过天翼云的负载均衡进行暴露的，相对于 NodePort 方式，有更高的可用性和性能。

- Headless Service

Headless Service 类型的 Service 是在 Service 属性中指定 clusterIP 字段为 None 类型。采用 Headless Service 类型后，Service 将没有固定的虚拟 IP 地址，客户端访问 Service 的域名时会通过 DNS 返回所有的后端 Pod 实例的 IP 地址，客户端需要采用 DNS 负载均衡来实现对后端的负载均衡。

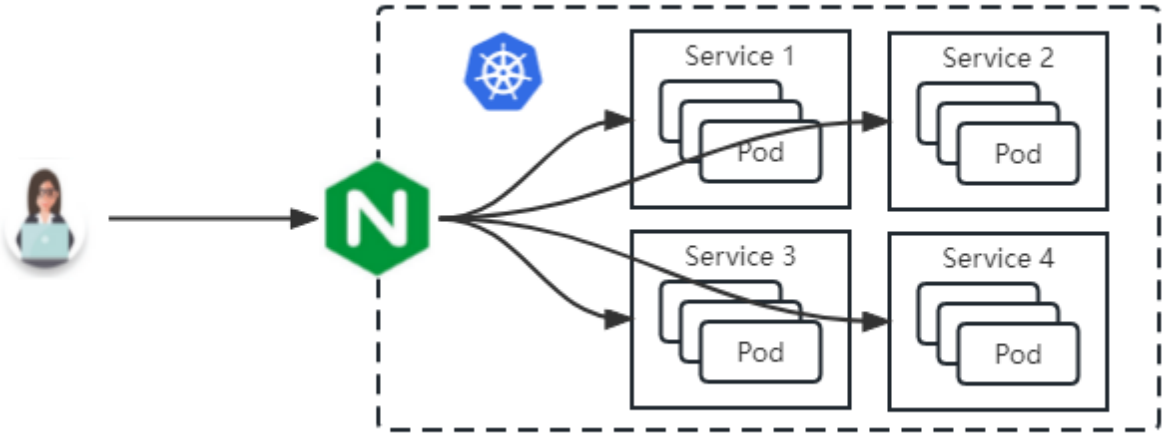
- ExternalName

ExternalName 类型的 Service 将集群外部的域名映射到集群内部的 Service 上，例如将外部的数据库域名映射到集群内部的 Service 名，那么就能在集群内部通过 Service 名直接访问。

3.2.5.4 Ingress

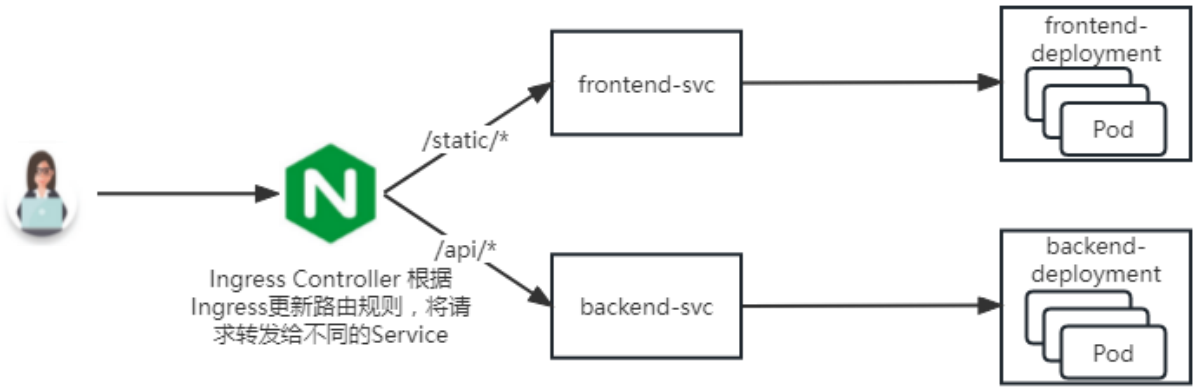
在 CCSE 集群中，与 Service 的 4 层负载均衡不同，Ingress 是集群内 Service 对外暴露 7 层的访问接入点。您可以通过 Ingress 资源来配置不同的 7 层的转发规则，例如通过域名或者访问路径来路由到不同的 Service 上，从而达到 7 层的负载均衡作用。

Ingress 和 Service 的关系：



示例：

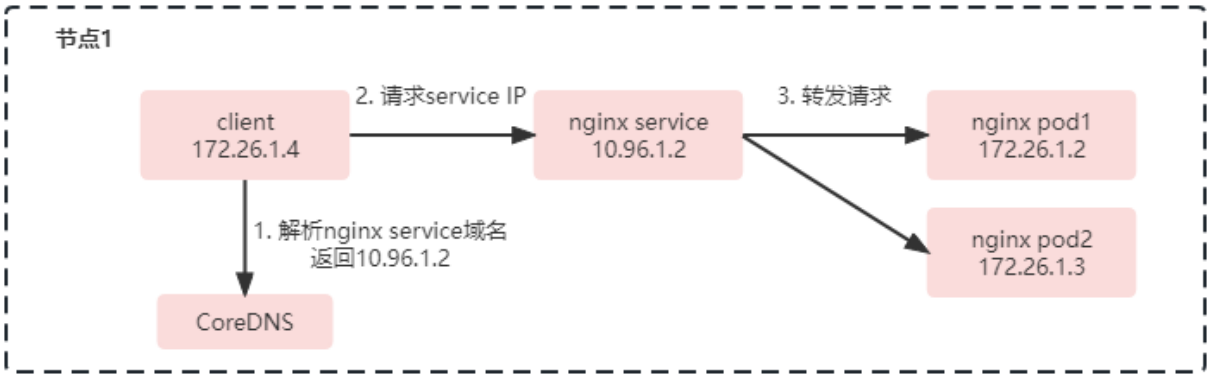
常见的前后端分离的架构方式中，前后端的访问地址分别使用不同的访问路径。对应这种场景，可以采用 Ingress，根据 7 层的访问路径负载到不同的应用实例上。



3.2.5.5 服务发现 DNS

CCSE 使用 DNS 来实现应用的服务发现能力，例如客户端应用可以通过 Service 的服务名解析出它的 ClusterIP 访问、可以通过 StatefulSet 的 Pod 名解析出 Pod 的 IP 地址。采用 DNS 服务发现的能力让集群中应用间的调用与 IP 地址和部署环境相解耦。

集群的 Coredns 会将 Service 名自动转换成对应的 Service 的 IP 地址，来实现不同部署环境中同样的访问入口。



3.2.5.6 网络底层基础设施

1、VPC

天翼云专有网络 VPC（Virtual Private Cloud）是基于天翼云创建的自定义私有网络，不同的专有网络之间彻底逻辑隔离。您可以在专有网络内创建和管理云产品实例，例如云服务器、云数据库和负载均衡等。

2、私网网段

在创建专有网络时，您需要以 CIDR 地址块的形式指定专有网络使用的私网网段。

您可以使用下表中标准的私网网段及其子网作为 VPC 的私网网段。

网段	说明
10.0.0.0/8-24	可用私网 IP 数量（不包括系统保留地址）：16,777,216
172.16.0.0/12-24	可用私网 IP 数量（不包括系统保留地址）：1,048,576
192.168.0.0/16-24	可用私网 IP 数量（不包括系统保留地址）：65,536

3、ELB

天翼云负载均衡 ELB（Server Load Balancer）通过设置虚拟服务地址，将添加的 ECS 实例虚拟成一个高性能、高可用的应用服务池，并根据转发规则，将来自客户端的请求分发给云服务器池中的 ECS 实例。

负载均衡默认检查云服务器池中的 ECS 实例的健康状态，自动隔离异常状态的 ECS 实例，消除了单台 ECS 实例的单点故障，提高了应用的整体服务能力。

负载均衡由以下三个部分组成：

负载均衡实例：一个负载均衡实例是一个运行的负载均衡服务，用来接收流量并将其分配给后端服务器。要使用负载均衡服务，您必须创建一个负载均衡实例，并至少添加一个监听和两台 ECS 实例。

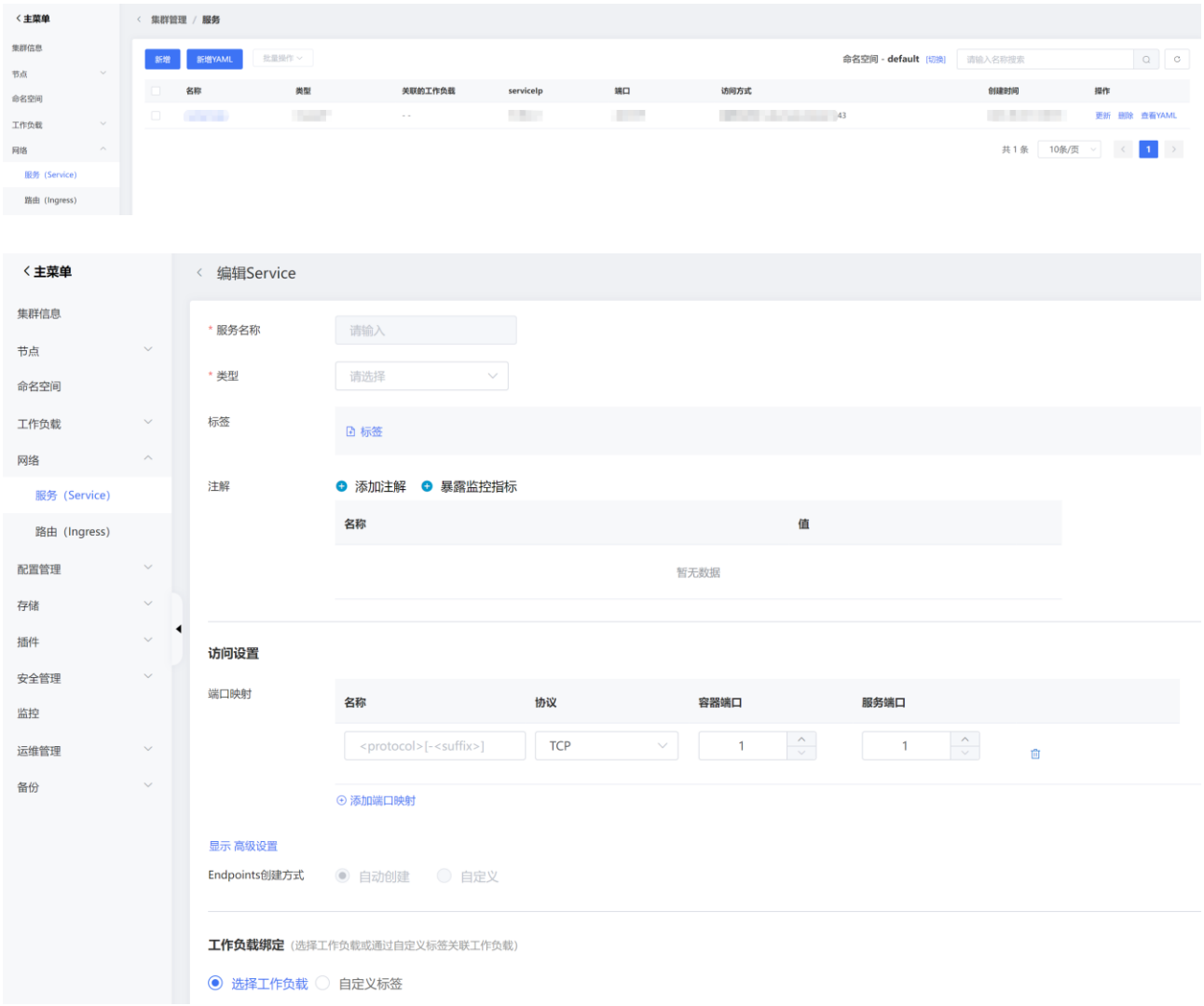
监听：监听用来检查客户端请求并将请求转发给后端服务器。监听也会对后端服务器进行健康检查。

后端服务器：一组接收前端请求的 ECS 实例。

3.2.5.7 功能入口

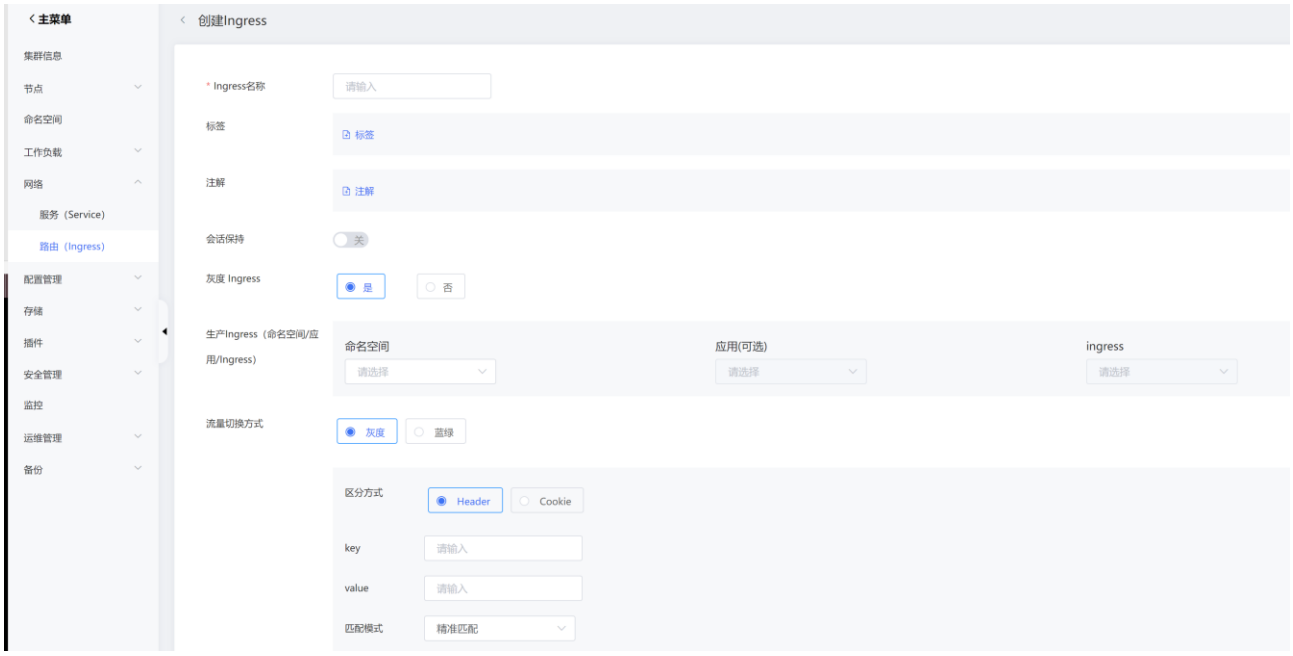
3.2.5.7.1 Service 管理

Service 的功能入口在一级菜单“集群”的二级菜单“网络”中，支持 ClusterIP、NodePort、LoadBalance 三种类型 Service 的创建、查看、修改和删除。



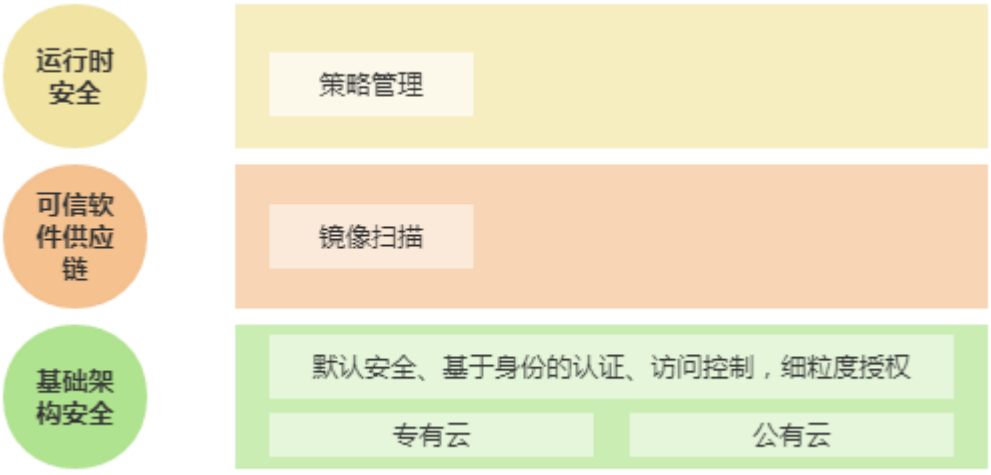
3.2.5.7.2 Ingress 管理

Ingress 的功能入口在一级菜单“集群”的二级菜单“网络”中，支持用户创建、查看、修改和删除等操作，也支持用户基于 Ingress 实现灰度和蓝绿发布的能力。



3.2.6 安全体系

本文从运行时安全、可信软件供应链和基础架构安全三个维度介绍天翼云容器服务 Kubernetes 版的安全体系，包括安全巡检、策略管理、运行时监控和告警、镜像扫描、镜像签名、云原生应用交付链、默认安全、身份管理、细粒度访问控制等。



3.2.6.1 可信软件供应链

镜像扫描：容器镜像服务支持所有基于 Linux 的容器镜像安全扫描，可以识别镜像中已知的漏洞信息。您可以收到相应的漏洞信息评估和相关的漏洞修复建议，为您大幅降低使用容器的安全风险。

3.2.6.2 基础架构安全

默认安全：天翼云云容器引擎 CCSE 集群内所有系统组件均依据容器安全最佳实践进行了组件配置上的加固，同时保证系统组件镜像没有严重级别的 CVE 漏洞。每个新建的集群需指定一个与之对应的安全组。创建的集群默认不允许公网 SSH 连入。

身份管理：CCSE 集群内所有组件之间的通讯链路均需要 TLS 证书校验，保证全链路通讯的数据传输安全。子账号用户均可以通过控制台或 OpenAPI 的方式获取连接指定集群 API Server 的 Kubeconfig 访问凭证，具体操作，请参见获取集群 KubeConfig 接口。CCSE 负责维护访问凭证中签发的身份信息，对于可能泄露的已下发 Kubeconfig，可以及时进行吊销操作。

细粒度访问控制：基于 Kubernetes RBAC 实现了对 CCSE 集群内 Kubernetes 资源的访问控制，它是保护应用安全的一个基本且必要的加固措施。CCSE 在控制台的授权管理页面中提供了命名空间维度的细粒度 RBAC 授权能力，主要包括以下几点：

1）根据企业内部不同人员对权限需求的不同，系统预置了管理员、运维人员、开发人员等对应的 RBAC 权限模板，降低了 RBAC 授权的使用难度。

2）支持角色扮演用户的授权。

3）支持绑定用户在集群中自定义的 ClusterRole。

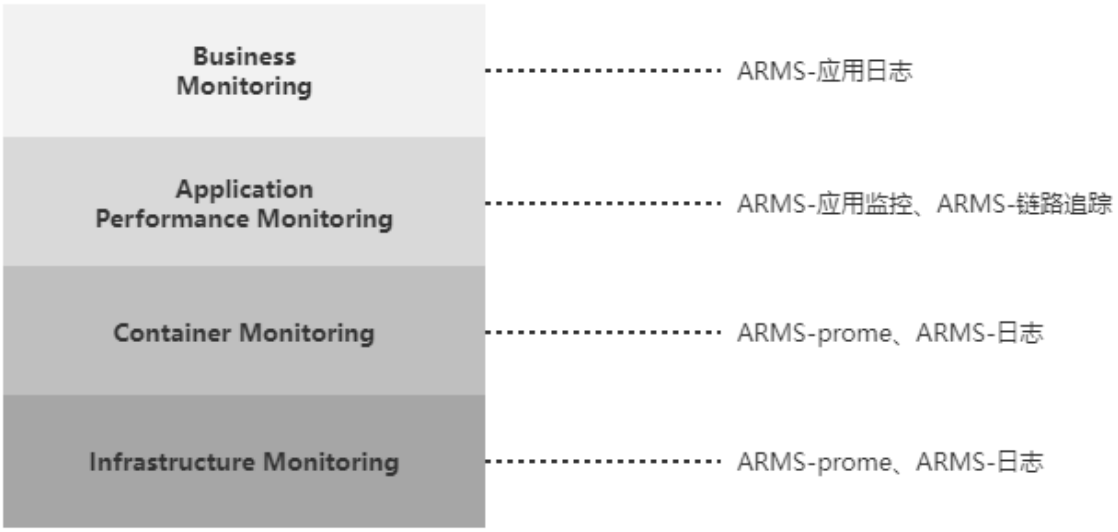
CCSE 同时支持以组件管理的方式安装 Gatekeeper 组件，提供基于 OPA 策略引擎的细粒度访问控制能力。

3.2.7 可观测性体系

可观测性指如何从外部输出推断及衡量系统内部状态。Kubernetes 可观测性体系包含监控和日志两部分，监控可以帮助开发者查看系统的运行状态，而日志可以协助问题的排查和诊断。本文介绍天翼云云容器引擎 CCSE 可观测性生态分层和各层的可观测能力，以帮助您更好地对容器服务可观测性生态有一个全面的认识。

3.2.7.1 容器服务可观测生态概述

从可观测性的角度，以 CCSE 为基础的系统架构可以粗略分为 4 个层次。自下而上分别是：基础设施层、容器性能层、应用性能层、用户业务层。



3.2.7.2 基础设施层可观测性

指容器服务 CCSE 所依赖的底层资源的可观测场景：定位 Pod 与节点组成的资源池的调用链路，可视化拓扑关系，以及基础设施监控，例如宿主机节点、网络基础组件的性能监控等。

解决方案	方案介绍	适用场景
架构可视化感知方案	Kubernetes 集群中的业务是运行在节点组成的资源池上，使得定位 Pod 的调用链路以及拓扑关系非常复杂。那么如何以可视化的方式监控 Kubernetes 中的负载状态，及更好地可视化集群中流量的吞吐是非常重要的问题。	适用全部场景
基础设施	资源监控是 Kubernetes 中最常见的底层资源监控方	适用

指标监控方案	式，通过资源监控可以快速查看负载的 CPU、内存、网络等指标的使用率。在天翼云云容器引擎中，资源监控已经与云监控互通。	全部场景

3.2.7.3 容器层可观测性

指基于云容器引擎 CCSE 构建系统的容器抽象层的可观测场景，包括集群的性能、事件等监控，容器的性能，以及容器组件等监控。

解决方案	方案介绍	适用场景
天翼云 AMS 的监控方案	Prometheus 是社区官方的容器场景云原生指标可观测方案。天翼云 AMS 监控全面对接开源 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。借助天翼云 AMS 监控，您无需自行搭建 Prometheus 监控系统，因而无需关心底层数据存储、数据展示、系统运维等问题。推荐使用天翼云 AMS 云产品。	适用全部场景

3.2.7.4 应用性能层可观测性

指基于云容器引擎 CCSE 构建系统的具体应用场景，包括应用指标性能（Metric）、系统调用链（Tracing）、日志监控（Logging）等，例如基于容器服务构建一个 Java 应用，JAVA 应用的线程数指标等。

解决方案	方案介绍	适用场景

无侵入应用监控APM监控方案	推荐使用天翼云应用性能监控 ARMS（Application Realtime Monitor Service）作为应用性能层监控方案，ARMS 是一款天翼云应用性能管理（APM）类监控产品。只要为 CCSE 集群安装 cubems 插件，在部署其中的 Java 打标，您无需修改任何代码，就能借助 ARMS 对 Java 应用进行全方位监控，以便您更快速地定位出错接口和慢接口、重现调用参数、检测内存泄漏、发现系统瓶颈，从而大幅提升线上问题诊断问题的效率。	适用部分场景，包括 JAVA 应用的应用监控，方案接入支持无侵入方式，无需进行代码改造。

3.2.7.5 用户业务可观测性

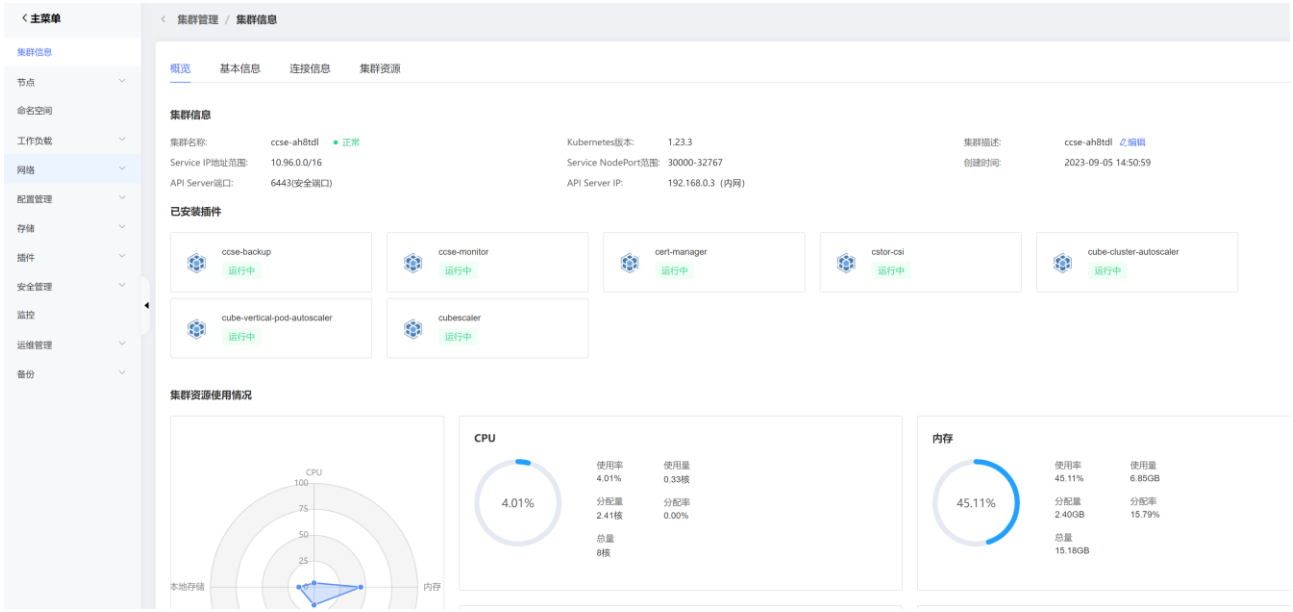
基于云容器引擎 CCSE 构建的业务系统的具体业务场景，例如基于容器服务构建一套高可用可扩展的网站，网站的业务运营数据 PV、UV 等，例如应用的成本审计场景等。

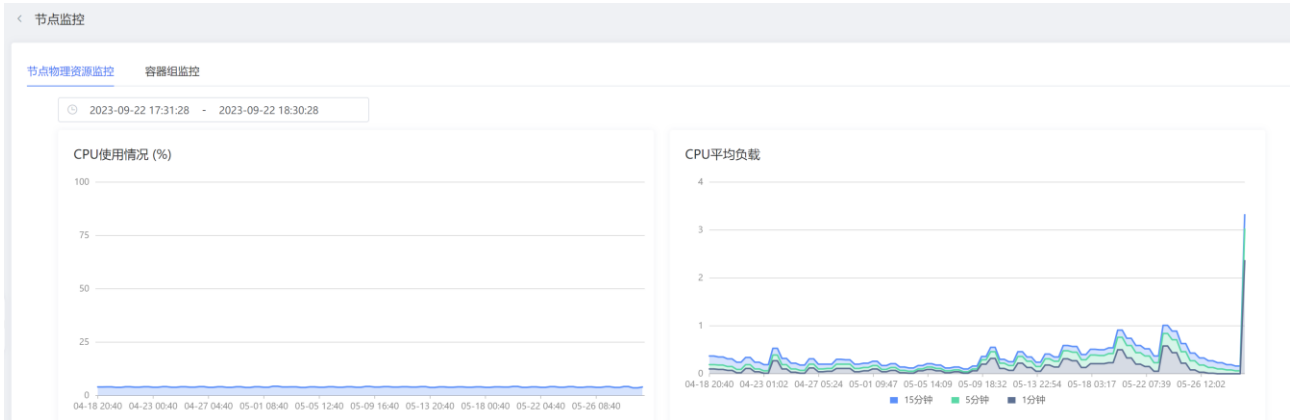
解决方案	方案介绍	适用场景
日志监控方案	推荐使用添一员 ARMS-应用日志作为自定义指标的观测方案。您通过日志服务收集，并在日志服务中配置业务大盘，观测自己的业务情况。	适用全部场景。

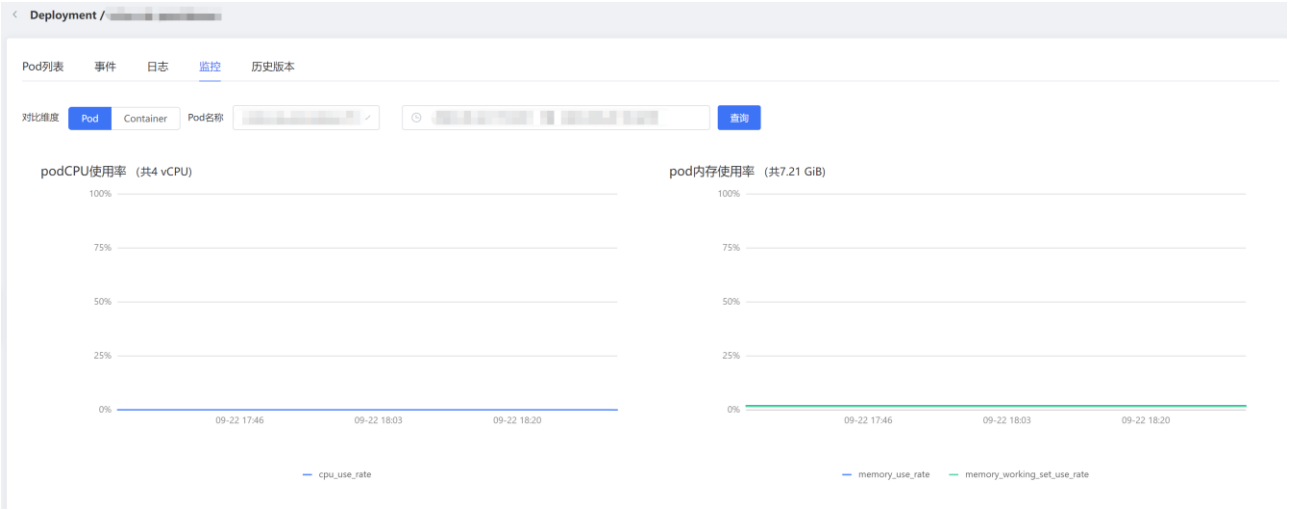
3.2.7.6 功能入口

3.2.7.6.1 监控

CCSE 集群提供集群、节点、应用等维度的监控的能力。







3.2.7.6.2 日志

CCSE 集群对接了 ARMS-日志，ARMS-日志提供采集、存储、检索等能力。

3.2.8 弹性伸缩

弹性伸缩是根据业务需求和策略，经济地自动调整弹性计算资源的管理服务。本文介绍弹性伸缩的背景信息和弹性伸缩涉及的组件。

3.2.8.1 背景介绍

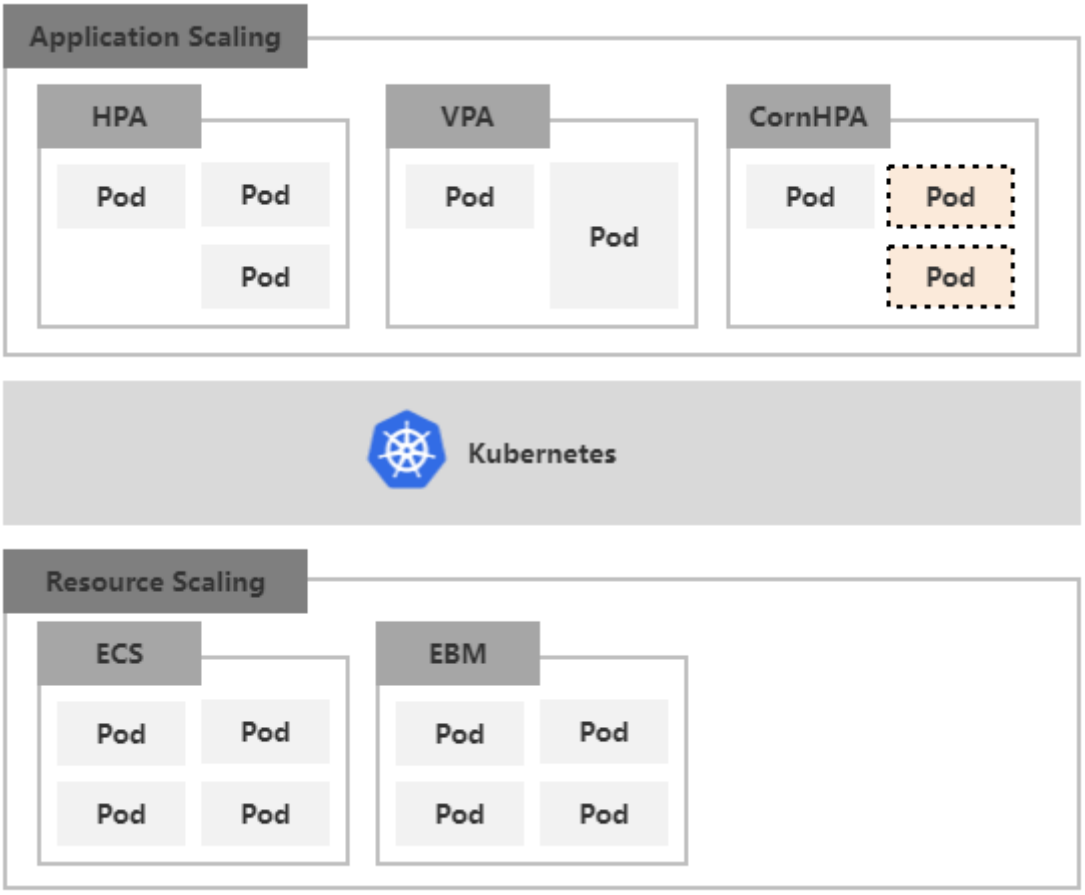
弹性伸缩是 CCSE 被广泛采用的功能，典型的场景包含在线业务弹性、大规模计算训练、深度学习 GPU 或共享 GPU 的训练与推理、定时周期性负载变化等。弹性伸缩分为两个维度：

调度层弹性，主要是负责修改负载的调度容量变化。例如，HPA 是典型的调度层弹性组件，通过 HPA 可以调整应用的副本数，调整的副本数会改变当前负载占用的调度容量，从而实现调度层的伸缩。

资源层弹性，主要是集群的容量规划不能满足集群调度容量时，会通过弹出 ECS 等资源的方式进行调度容量的补充。

两层的弹性组件与能力可以分开使用，也可以结合在一起使用，并且两者之间是通过调度层面的容量状态进行解耦。

3.2.8.2 CCSE 弹性伸缩组件介绍



3.2.8.3 调度层弹性组件介绍

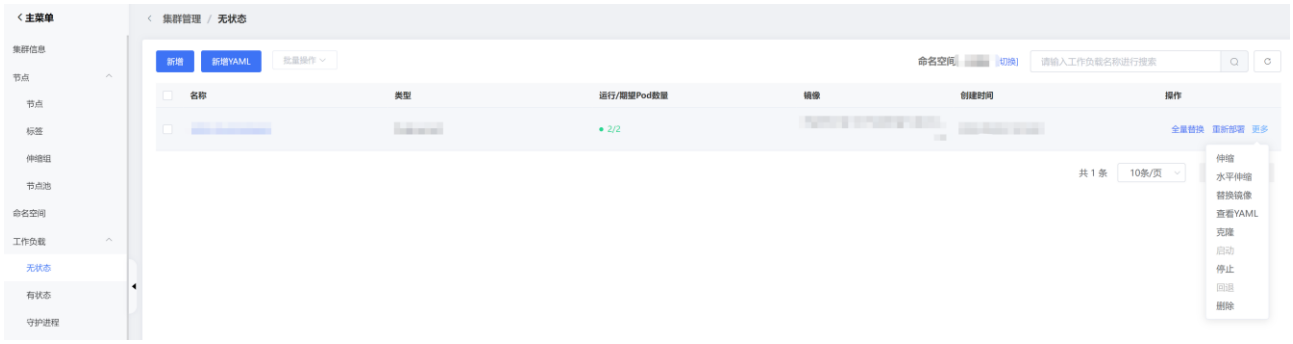
组件名称	组件介绍	适用场景	使用限制

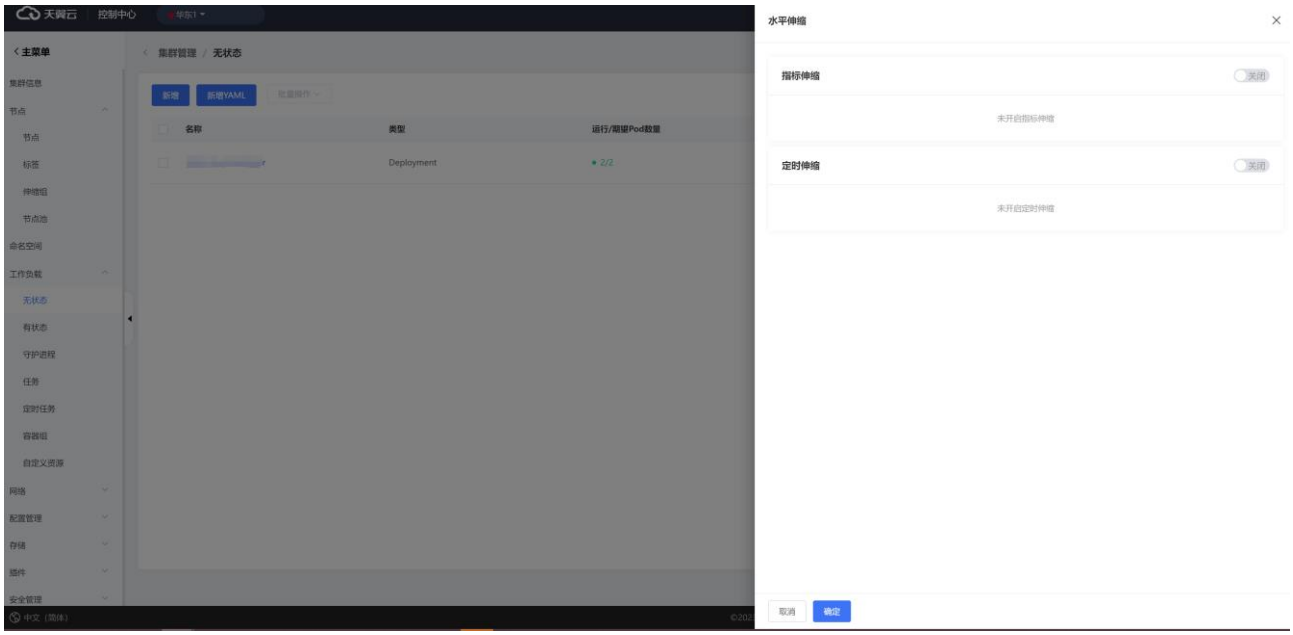
HPA	Kubernetes 内置组件，主要面向在线业务。	在线业务	适用于 Deployment、StatefulSet 等实现 scale 接口的对象。
VPA（alpha）	开源社区组件，主要面向大型单体应用。	大型单体应用	适用于无法水平扩展的应用，通常是在 Pod 出现异常恢复时生效。
Cubescaler	CCSE 自研插件，主要面向应用资源使用率存在周期性变化的场景。	周期性负载业务	适用于 Deployment、StatefulSet 等，实现了 scale 接口的对象。

3.2.8.4 功能入口

3.2.8.4.1 应用调度伸缩

CCSE 支持用户手动伸缩应用容器实例，HPA 自动伸缩策略和定时自动伸缩。

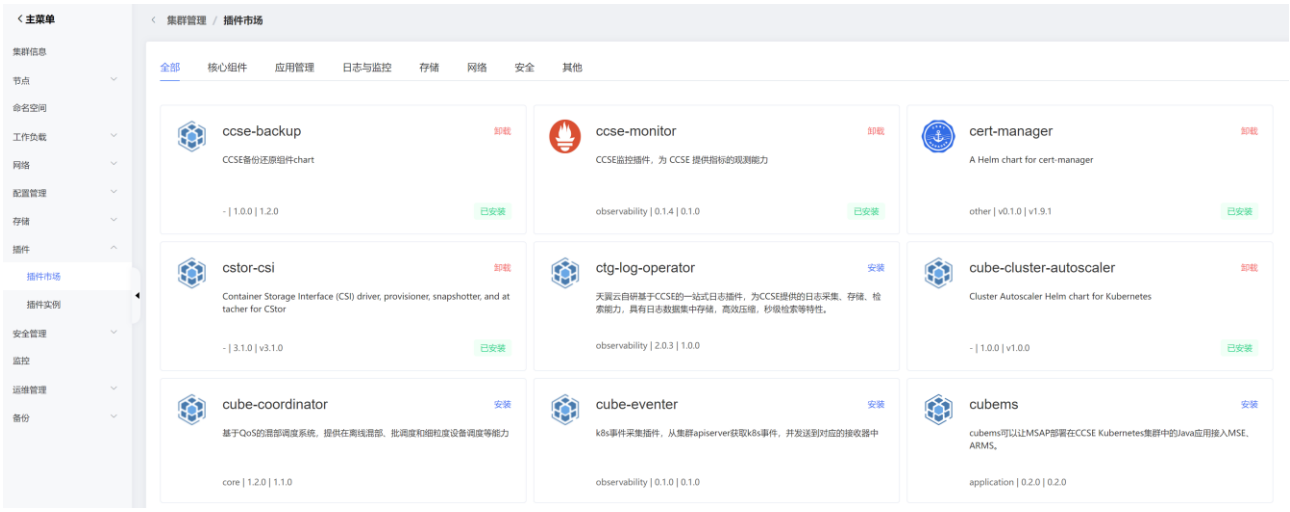




3.2.9 插件管理

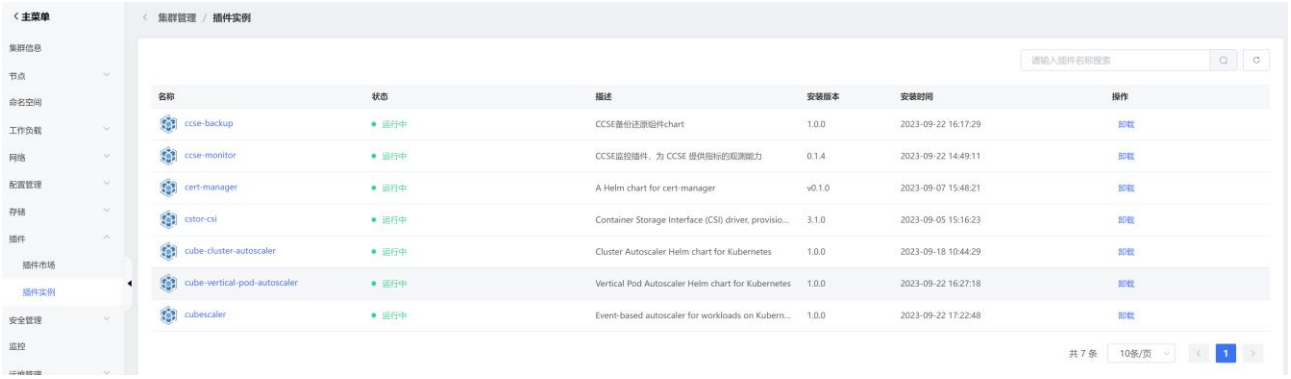
3.2.9.1 插件安装

插件市场提供多种插件，如应用管理、日志与监控、储存、网络等，可选择安装插件增强集群能力。



3.2.9.2 插件管理

支持对插件实例进行管理，可查看集群已安装的插件，插件使用的资源列表、参数列表，对插件进行卸载操作。



3.2.10 镜像管理

CCSE 集群对接了镜像服务，支持从 CCSE 控制台跳转到镜像服务控制台。


CCSE
概览
集群
模板市场
镜像服务

4 快速入门

4.1 创建一个集群

4.1.1 订购集群

在 CCSE 控制台，选择集群 > 新增，进入集群订购界面

在集群订购页面，在集群配置中填写集群实例名称、选择集群 VPC 及子网、选择 API Server 访问方式

实例名称：集群的实例名称

虚拟私有云：集群所在的 VPC

网络插件：集群使用的网络插件，目前可选择 calico 插件及自研 cubecni 插件

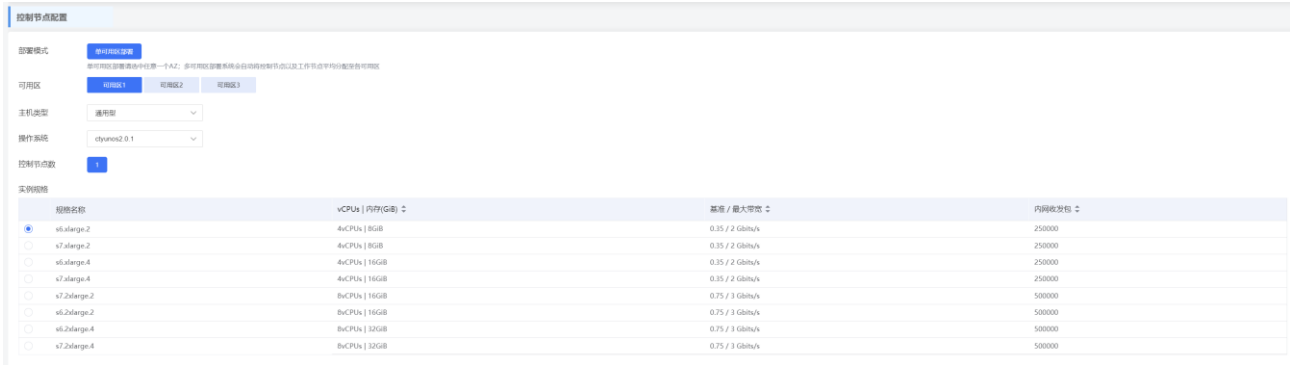
Service CIDR：为集群内容器分配在容器网络地址范围内的 IP 地址

所在子网：集群所在的子网

在控制节点配置中选择控制节点的数量和规格

控制节点数：集群 master 节点的数量

实例规格：集群 master 节点的规格

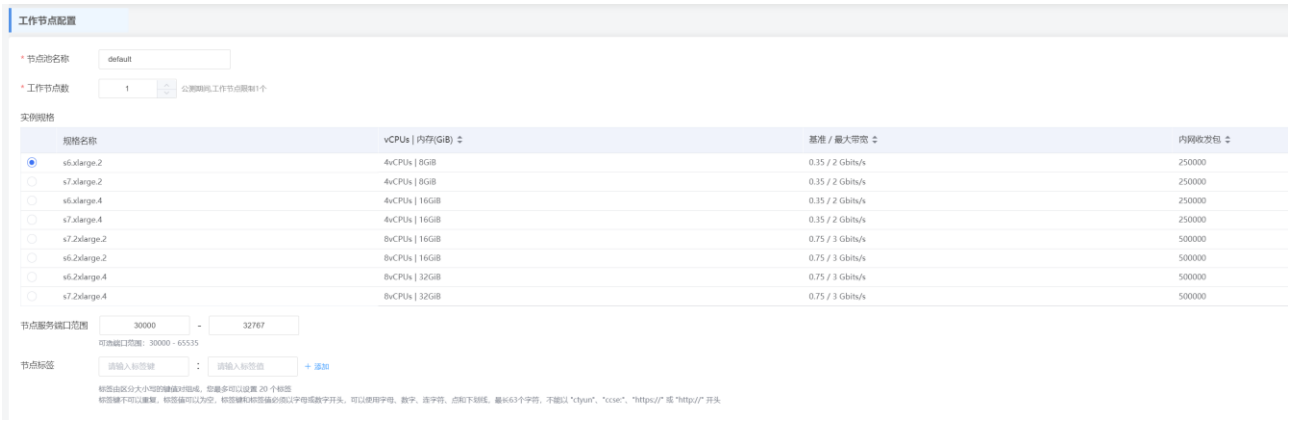


在工作节点配置中选择工作节点数量及规格、选择节点服务端口的范围

工作节点数：集群 worker 节点的数量

实例规格：集群 worker 节点的规格

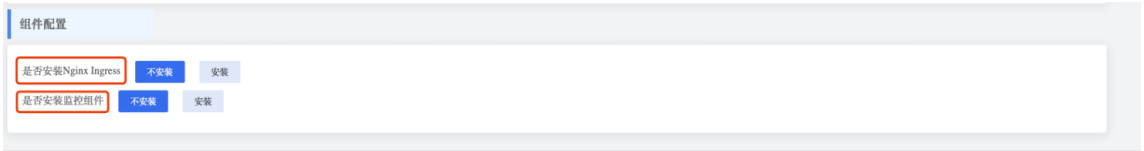
节点端口范围：k8s 能够使用节点的端口范围



在组件配置中选择是否安装监控插件几 Nginx Ingress 插件

监控插件：开源监控插件，为 CCSE 提供指标的观测能力

Nginx Ingress 插件：为 CCSE 提供创建 nginx ingress 的能力



提交订单

4.1.2 查看集群

进入 CCSE 控制台，在集群选项卡中可以看到新创建的集群。点击集群 ID 进入集群详情界面，可以查看集群相关信息

4.1.3 退订集群

在 CCSE 控制台，在集群选项卡可以点击 更多 > 退订对集群实例进行退订

4.2 创建一个无状态工作负载

4.2.1 创建工作负载及服务

进入 CCSE 控制台，在集群选项卡中选择一个集群进入集群详情界面。选择工作负载 > 无状态 > 新增 选择 default 命名空间，创建一个 Deployment

配置项说明：

配置项	说明
Deployment 名称	工作负载的名称
数据卷（选填）	为容器提供存储，目前支持临时路径、主机路径、配置文件、本地卷（Local PV）、NFS、Ceph，还需挂载到容器的指定路径中。
实例数量	工作负载的副本数，可选择手动设置或自动伸缩
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	容器实例的名称
镜像及镜像版本	支持选择容器镜像服务企业版、容器镜像服务个人版的镜像
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。

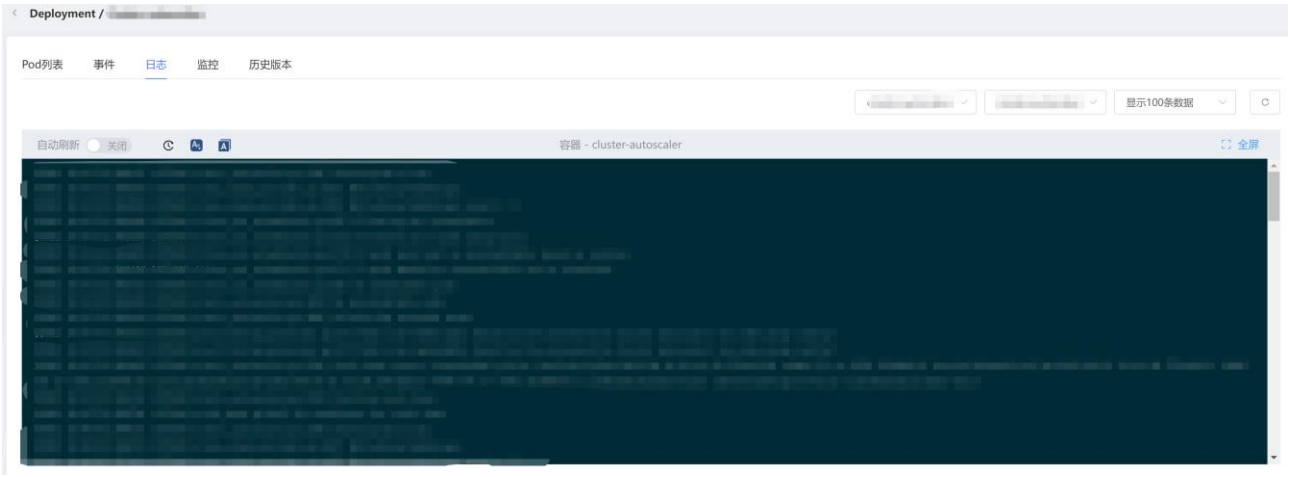
启动执行（选填）	启动执行-命令：对应镜像的 ENTRYPOINT 命令，将会覆盖镜像的 ENTRYPOINT 命令；每个输入框仅输入一个命令或参数 启动执行-参数：对应镜像的 CMD 命令，将会覆盖镜像的 CMD 命令；每个输入框仅输入一个命令或参数
启动后处理	容器启动后执行，注意由于是异步执行，无法保证一定在 ENTRYPOINT 之后运行；每个输入框仅输入一个命令或参数
停止前处理	容器停止前执行，常用于资源清理；每个输入框仅输入一个命令或参数
容器健康检查	存活检查：检查容器是否正常，不正常则重启实例 就绪检查：检查容器是否就绪，不就绪则停止转发流量到当前实例
特权级容器	容器开启特权级，将拥有宿主机的 root 权限
Container 安全上下文	为 Container 设置安全上下文，仅适用于该 Container：若 Pod、Container 层面都设置了用户、用户组、Selinux 上下文，Container 的设置会覆盖 Pod 的设置
访问设置	配置 Service 访问负载

4.2.2 查看容器实例事件

在工作负载实例详情的界面中，选择事件可以查看负载事件及容器事件

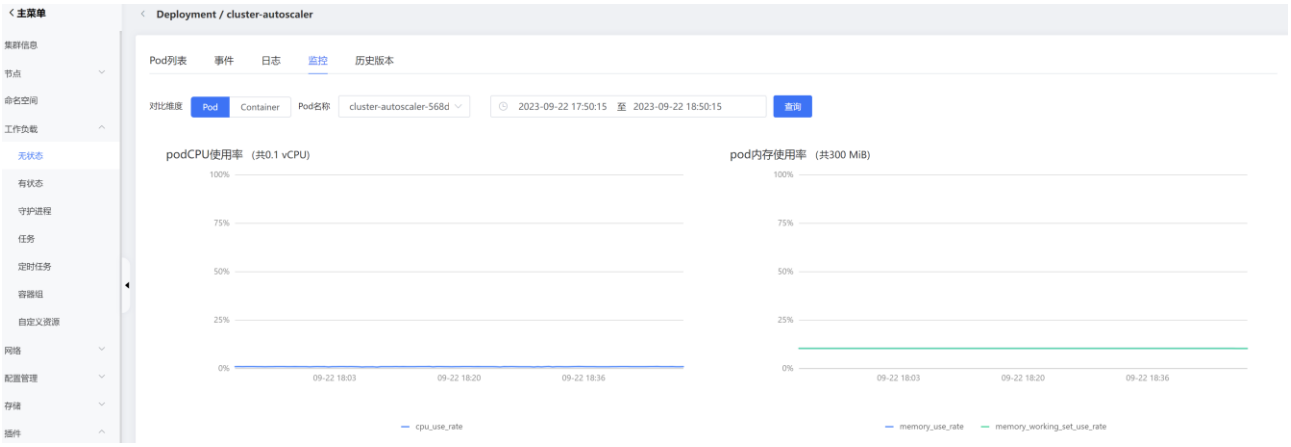
4.2.3 查看容器实例日志

在工作负载实例详情的界面中，选择日志可以查看容器实例的日志



4.2.4 查看容器监控

在工作负载实例详情的界面中，选择监控可以查看工作负载及容器的监控数据


主菜单
集群信息
节点
命名空间
工作负载
无状态
有状态
守护进程
任务
定时任务
容器组
自定义资源
网络
配置管理
存储
插件
Deployment / cluster-autoscaler
Pod列表
事件
日志
监控
历史版本
对比维度
Pod
Container
Pod名称
cluster-autoscaler-568d
2023-09-22 17:50:15 至 2023-09-22 18:50:15
查询
podCPU使用率 （共0.1 vCPU）
100%
75%
50%
25%
0%
09-22 18:03
09-22 18:20
09-22 18:36
cpu_use_rate
pod内存使用率 （共300 MiB）
100%
75%
50%
25%
0%
09-22 18:03
09-22 18:20
09-22 18:36
memory_use_rate
memory_working_set_use_rate

5 API 参考

查看官网 API 参考。

6 最佳实践

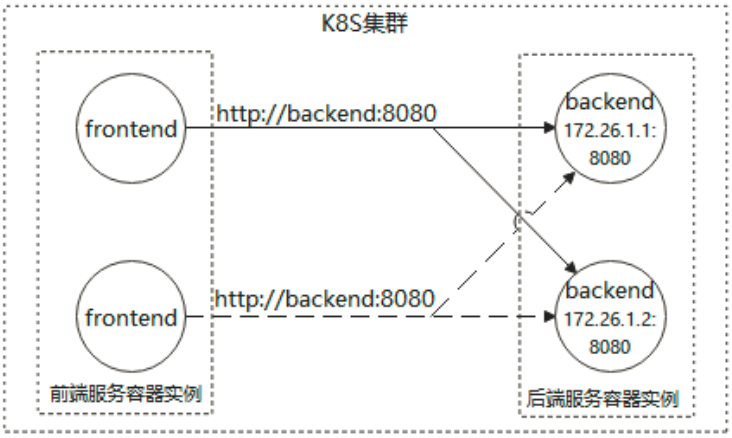
6.1 负载均衡最佳实践

一、集群内访问

假设我们在一个 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，那么有以下两种方式。

1.1 使用应用名进行访问

假设在 K8S 集群中发布的后端应用的名称为 backend，它监听 8080 端口，那么在前端应用的容器中，可以直接使用后端应用的名称加端口 http://backend:8080 进行访问，K8S 会自动转发到后端的某个容器实例。如下所示：

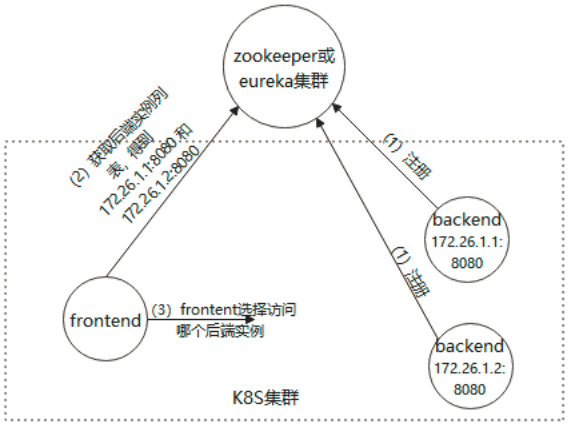


注意，上面的转发其实是四层转发，即如果后端应用是 mysql 类型的，可使用 tcp://backend:8080 进行访问，如果后端应用为 web 应用，可使用 http://backend:8080 进行访问。

另外，上面的访问方式要求前端应用和后端应用要在 K8S 集群的同一个命名空间内。如果二者不在同一个命名空间，比如后端应用在命名空间 ns2 中，那么前端应用应该使用 http://backend.ns2:8080 进行访问（即应用名后面要加上命名空间）

1.2 使用微服务架构自己的注册机制

如果业务开发者使用的微服务架构有注册机制，那么可以直接使用微服务框架的注册机制来做负载均衡。如下：



后端应用容器实例把自己的 IP 与端口注册到 zookeeper 或 eureka，前端应用容器实例去注册中心获取后端应用的实例列表，然后前端应用的实例自己决定访问哪一个实例。

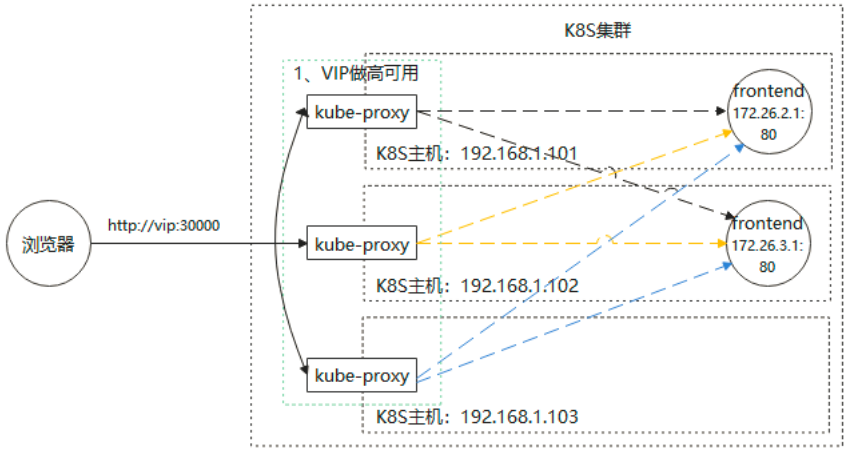
注册中心 zookeeper/eureka 集群可以部署在 K8S 集群内，也可以部署在集群外，这个需要业务方自己去部署。

二、集群外访问

假设我们要从浏览器上访问上面的前端应用，下面我们来介绍几种方法

2.1 NodePort

最简单的方法就是使用 NodePort。我们只需要为前端应用在 K8S 集群中创建一个 Service 对象，在这个对象中指定一个主机端口比如为 30000，那么，每台主机上的 kube-proxy（部署 K8S 集群时已部署好）便会监听 30000 端口。当访问主机的 30000 端口时，kube-proxy 便会转发到前端应用的容器实例去。如下：

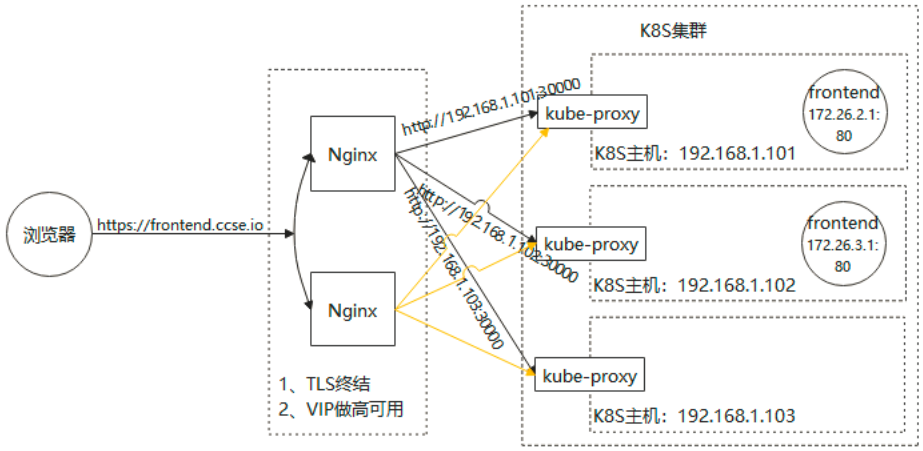


由于 CCSE 在 K8S 集群的 Master 主机上安装了 keepalived，绑定了一个 VIP，所以我们可以通过 http://vip:30000 来访问，保证了访问入口的高可用。

2.2 NodePort+Nginx

在上面的 NodePort 方案中，kube-proxy 处是没有办法设置 HTTPS 证书的，所以上面的方案只支持 HTTP，不支持 HTTPS。

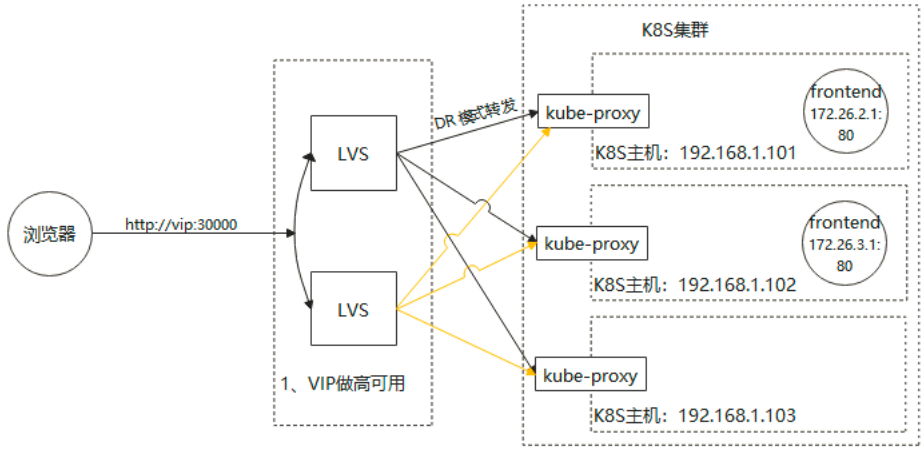
如果要支持 HTTPS，则需要在 kube-proxy 的前面手动搭建 Nginx 集群，在 Nginx 处手动配置 HTTPS 证书，手动配置转发规则；两个 Nginx 主机上还要安装 Keepalived 绑定一个 VIP，以保证高可用。如下：



上面的 Nginx 可以搭建在 K8S 集群的外面，也可以搭建在 K8S 集群的主机上（是物理部署在主机上，不是部署在容器内），但不要部署在 K8S 集群的 Master 主机上，因为 Master 主机上已经部署了 keepalived，以免和 Nginx 的 keepliaved 冲突 。

2.3 NodePort+LVS

NodePort 的方案中，VIP 只在一台主机上，可能承受不了大流量的访问。此时，我们可以在 NodePort 的前面手动搭建 LVS，如下：

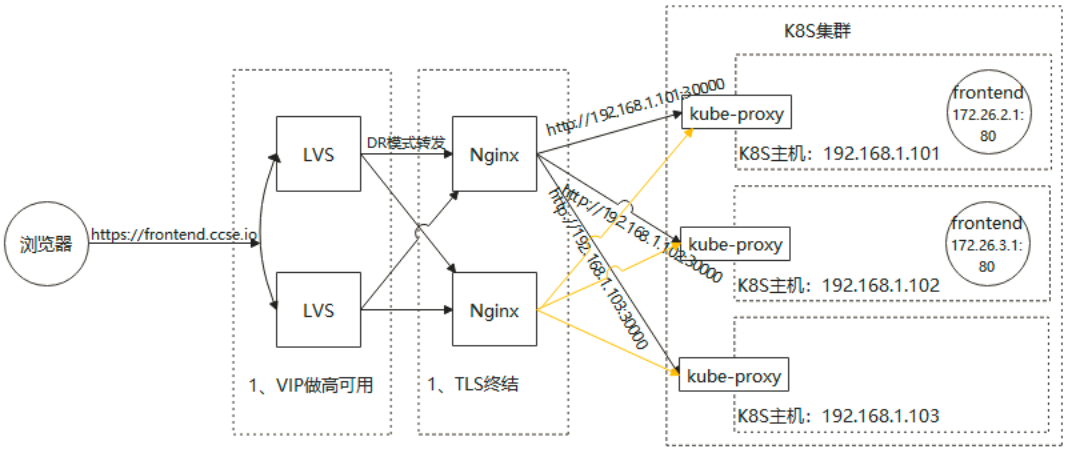


1、LVS 需要手动搭建，LVS 主机上还要安装 keepalived 绑定 VIP 以保证入口的高可用 2、LVS 配置 DR 模式转发到多台 K8S 主机上 3、kube-proxy 处每开放一个主机端口，在 LVS 处也需要配置这个端口 4、该方案不支持 HTTPS，因为 LVS 处是四层转发，不支持配置 HTTPS

2.4 NodePort+Nginx+LVS

如果流量大，还需要支持 HTTPS，则需要使用该种方案。

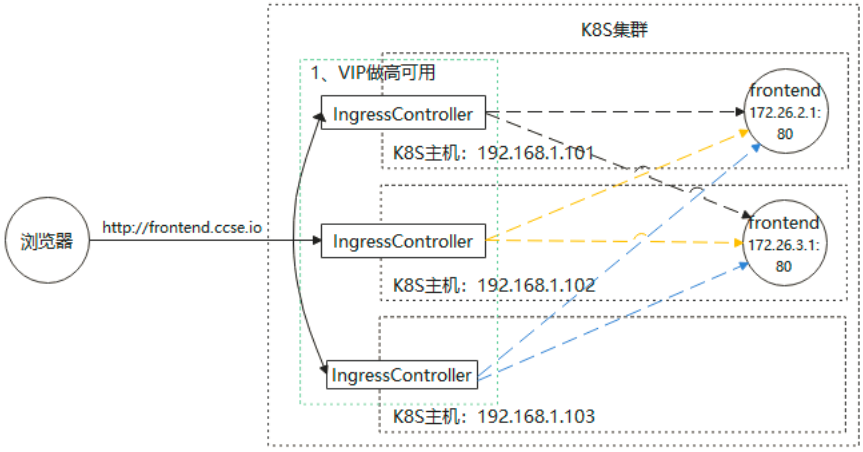
在 kube-proxy 的前面手动搭建 Nginx，在 Nginx 的前面手动搭建 LVS，如下：



1、LVS 需要手动安装，同时需要安装 keepalived 绑定 VIP，以保证访问入口的高可用 2、Nginx 需要手动安装，手动配置 HTTPS 证书 3、LVS 处只需要配置一个端口，转发到 Nginx 的同端口；Nginx 只需要监听一个端口，通过不同的域名，转发到 kube-proxy 上不同的端口；kube-proxy 根据不同的端口，转发到的集群内的不同的应用

2.5 Ingress

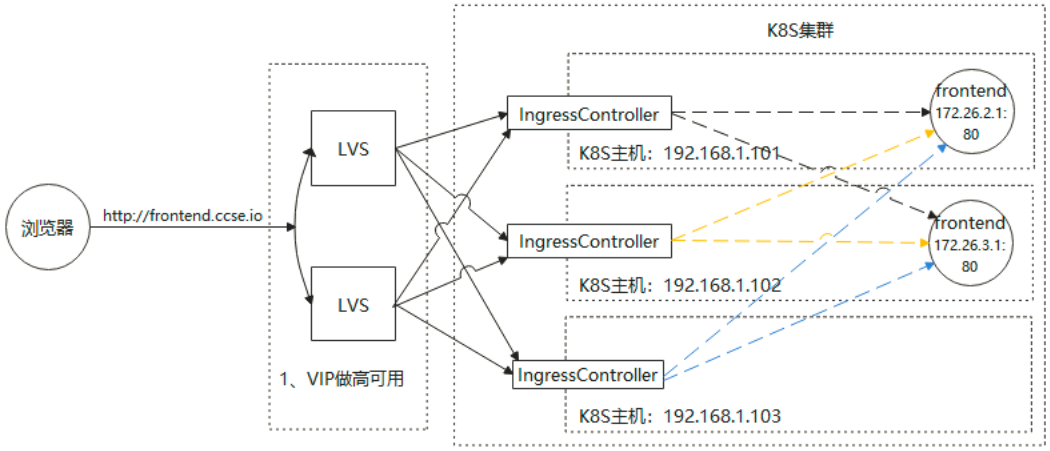
Ingress 是一种不同于 NodePort 的方案，如下



1、IngressController 只监听一个端口，通过不同的域名转发到 K8S 集群中不同的应用；所以该方案只支持域名访问，不支持 IP 访问。 2、只有 K8S 的 Master 主机上才会有 IngressController，Master 主机上已经有 VIP，所以 DNS 服务器把域名解析为 VIP，即可保证入口的高可用。 3、在 CCSE 管理台页面上，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。 4、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。

2.6 Ingress+LVS

上面的 Ingress 方案，由于 VIP 只在一台主机上，所以不适合大访问量的场景。如果访问量较大，可以使用 Ingress+LVS，如下：



1、LVS 需要手动安装与配置，LVS 主机上需要安装 keepalived 绑定 VIP 以保证入口高可用 2、LVS 只需要监听一个端口（IngressController 监听的端口） 3、浏览器通过域名访问，DNS 服务器把域名解析为 LVS 的 VIP，LVS 把请求转给其中某个 IngressController，IngressController 通过域名转发到不同的集群内的应用 4、该方案与上面方案一样，目前不支持 HTTPS，后续会支持。

2.7 对比

方案	四层或七层转发	支持高可用	支持 HTTPS	优缺点	适用场景
NodePort	四层	是	否	优点：使用简单，无须额外配置	TCP 或 HTTP 服务、访问量不大
NodePort+Nginx	七层	是	是	缺点：需要额外手动部	HTTPS 服务，访问量

				署与配置 Nginx	不大
NodePort+LVS	四层	是	否	缺点：需要额外手动部署与配置 LVS	TCP 或 HTTP 服务，访问量大
NodePort+Nginx+LVS	七层	是	是	缺点：需要额外手动部署与配置 Nginx 和 LVS	HTTPS 服务，访问量大
Ingress	七层	是	当前不支持，后续会支持	优点：使用简单；缺点：只支持域名访问，不支持 IP 访问，需要自行配置 DNS 服务器	HTTP 服务(后续支持 HTTPS 服务)，访问量不大
Ingress+LVS	七层	是	当前不支持，后续会支持	缺点：需要额外部署 LVS，只支持域名访问，不支持 IP 访问，需	HTTP 服务(后续支持 HTTPS 服务)，访问量大

				自行配置 DNS 服务器	

6.2 会话保持最佳实践

- **不使用会话保持**

1、前提条件

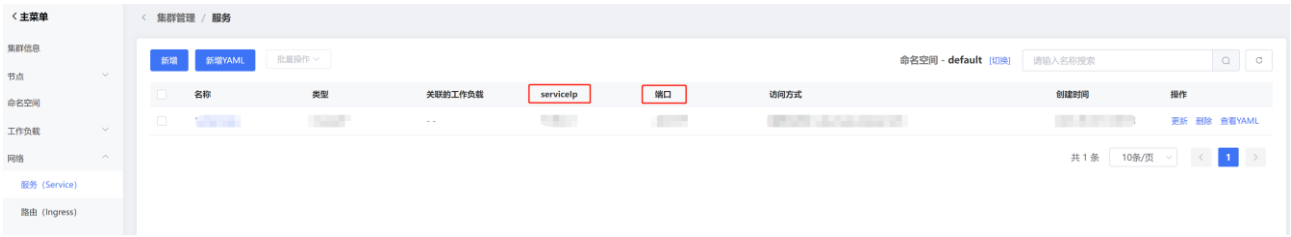
创建一个 nginx 工作负载，并确保工作负载的实例个数大于 1，工作负载不需要其他的额外特殊配置

创建一个 ClusterIP 类型的服务（Service）并关联到上述 nginx 工作负载，注意 Session Affinity 不需要设置，保持默认值即可

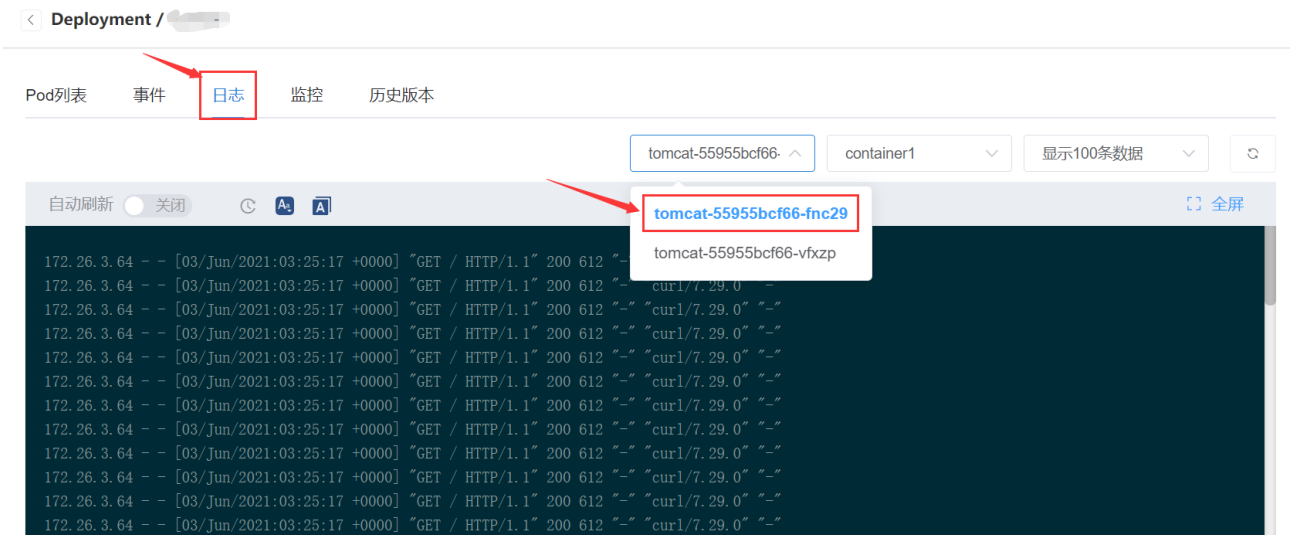
2、测试验证

发起服务调用，在集群节点上执行这个命令发起对服务的 100 次调用 > for i in {1..100};do curl 10.96.116.221:80;done;

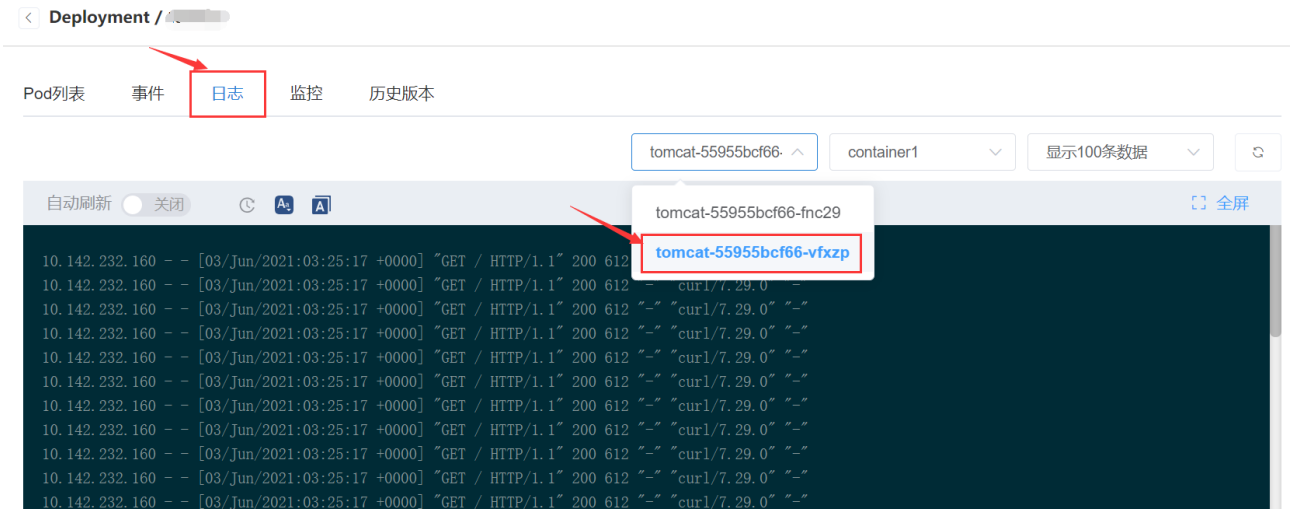
上述 curl 命令中的 IP 和端口来自如下地方：



观察工作负载日志，第一个 Pod 实例的日志输出：



第二个 Pod 实例的日志输出：



结论：服务请求会随机的转发到任一个 Pod 实例

- **集群内请求会话保持**

1、前提条件

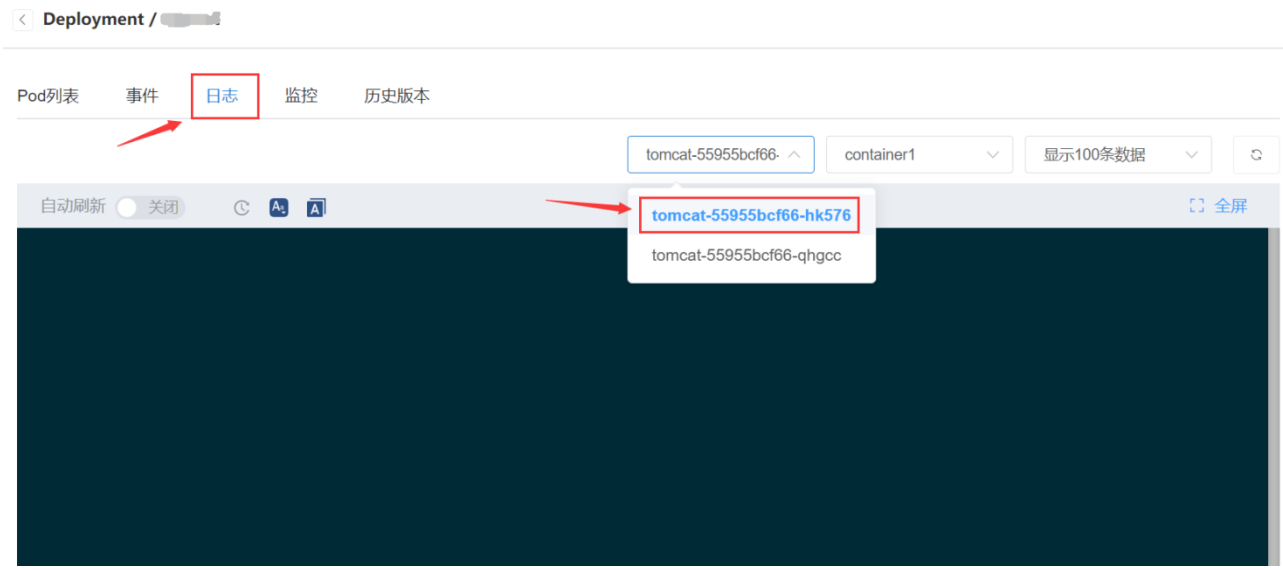
创建一个 nginx 工作负载，同上。

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

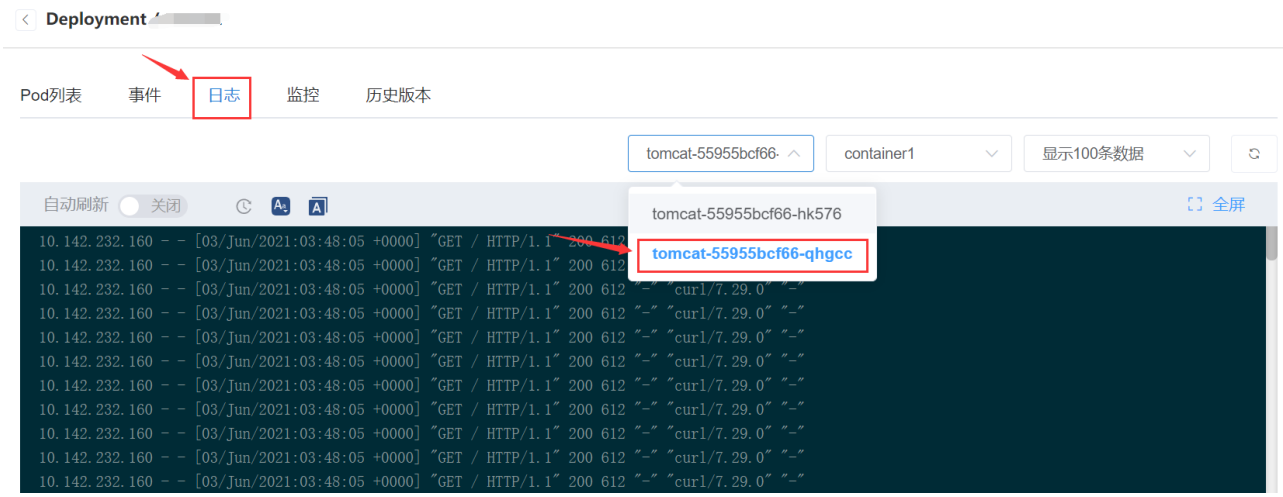
2、测试验证

发起服务调用，同上。观察工作负载日志，

第一个 Pod 实例的日志输出：



第二个 Pod 实例的日志输出：

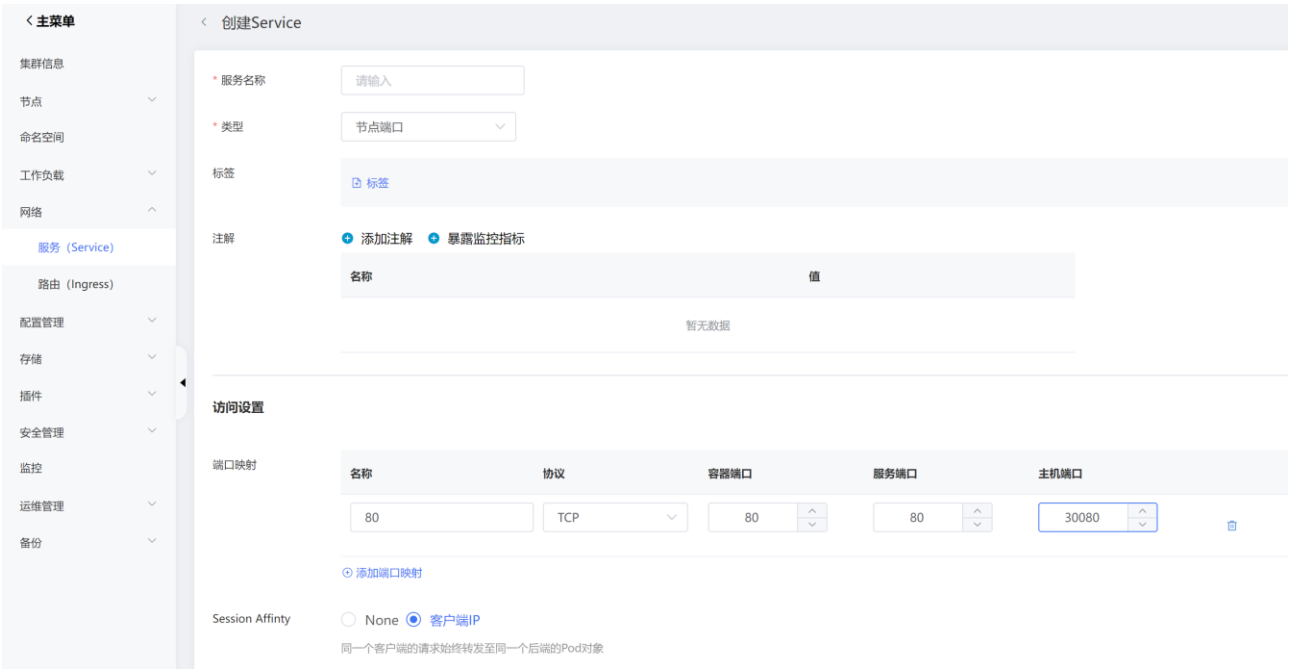


结论：服务请求会全部转发到某一个 Pod 实例，进行会话保持。

- **集群外 NodePort 访问请求会话保持**

1、前提条件

创建一个 nginx 工作负载，同上。创建一个 NodePort 类型的服务（Service）并关联到上述 nginx 工作负载，指定一个合法的主机端口，注意需要展开高级设置，并设置 Session Affinity 为客户端 IP

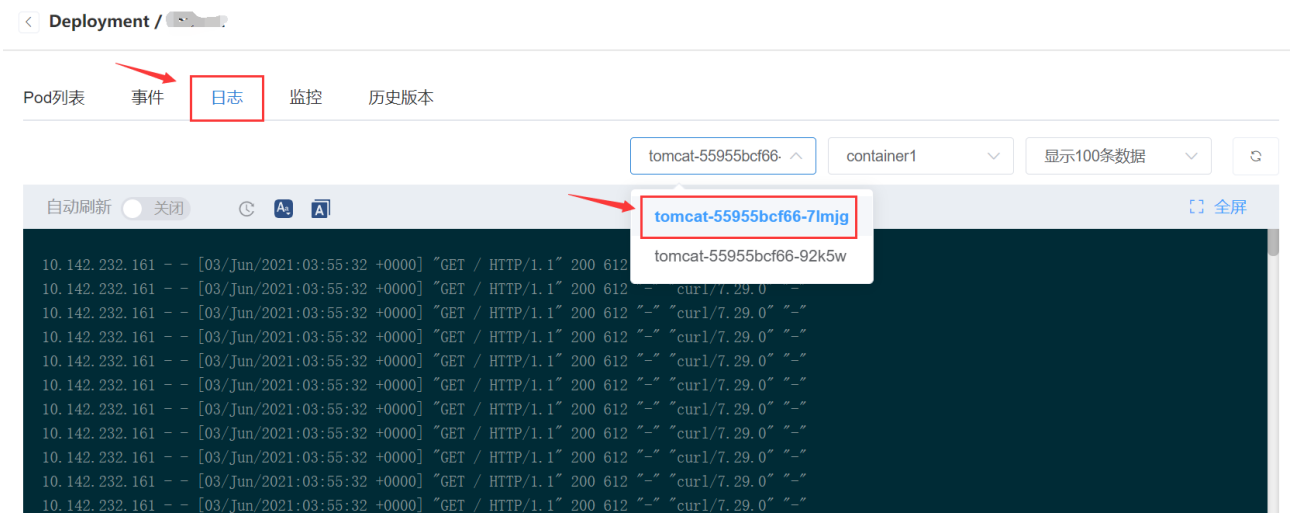


2、测试验证

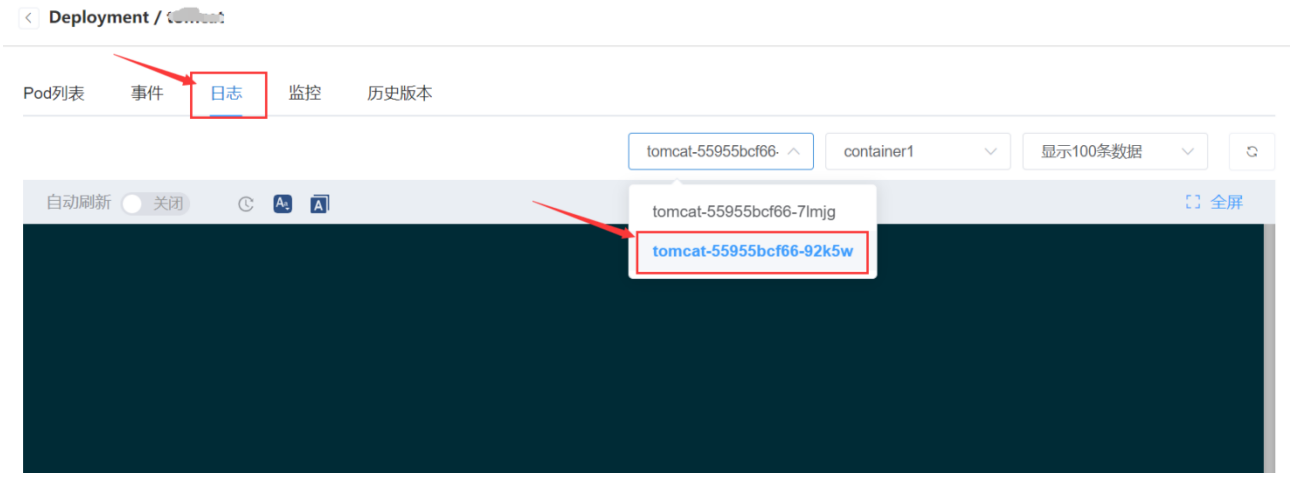
发起服务调用；在集群外执行这个命令发起对服务的 100 次调用 > for i in {1..100};do curl 10.142.232.160:30080;done;

上述 curl 命令中的 IP 可以是集群的 vip 或者集群任意节点 IP，端口是服务（Service）中指定的主机端口。

观察工作负载日志，第一个 Pod 实例的日志输出：



第二个 Pod 实例的日志输出：



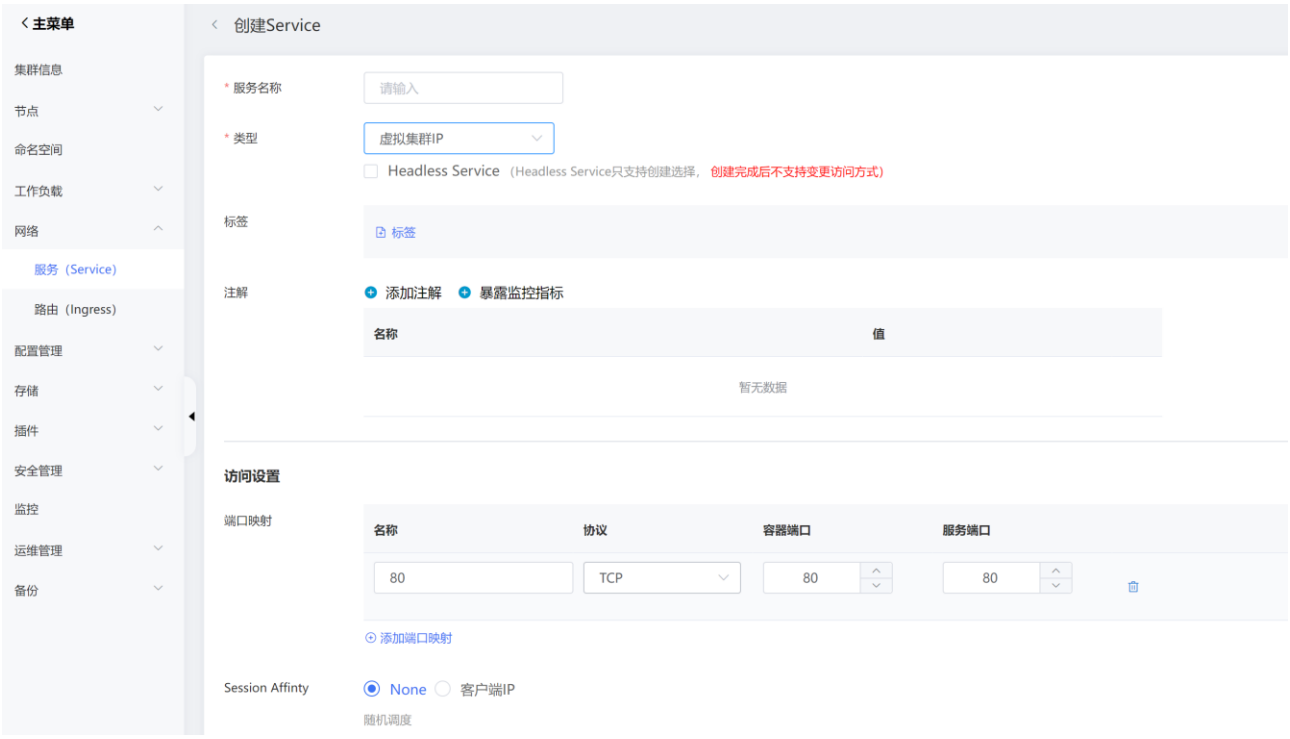
结论：集群外的请求会全部转发到某一个 Pod 实例，进行会话保持。

- **集群外 Ingress 访问请求会话保持**

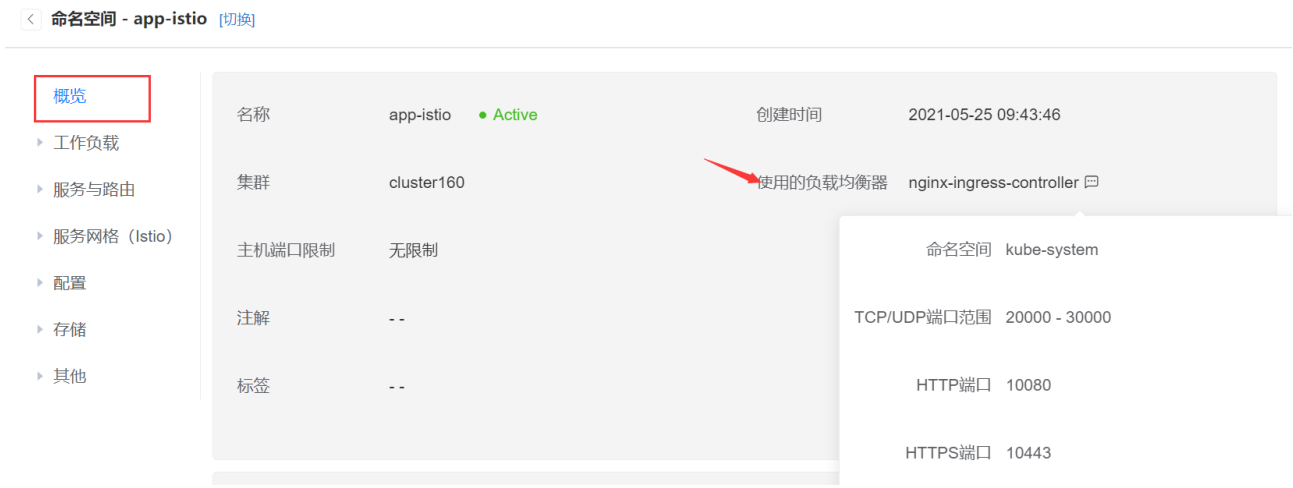
1、前提条件

创建一个 nginx 工作负载，同上

创建一个 ClusterIP 类型的服务（Service）并关联到上述 nginx 工作负载，注意 Session Affinity 不需要设置，保持默认值即可



确保当前命名空间已经绑定到一个负载均衡器：



创建一个生产路由（Ingress）并关联到上述服务（Service）

手动编辑 Ingress 资源文件，添加如下注解：# kubectl edit ingresses nginx-http -napp-istio annotations:
nginx.ingress.kubernetes.io/affinity: cookie

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 10.142.232.160 nginx.ccse.io

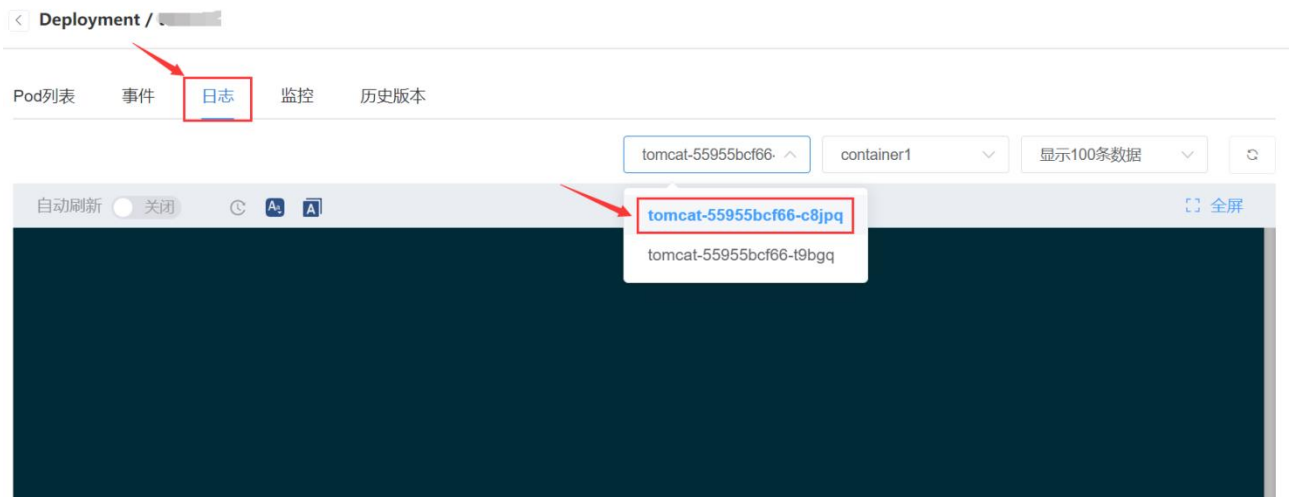
2、发起服务调用

在浏览器中多次发起对服务的请求 > http://nginx.ccse.io:10080/

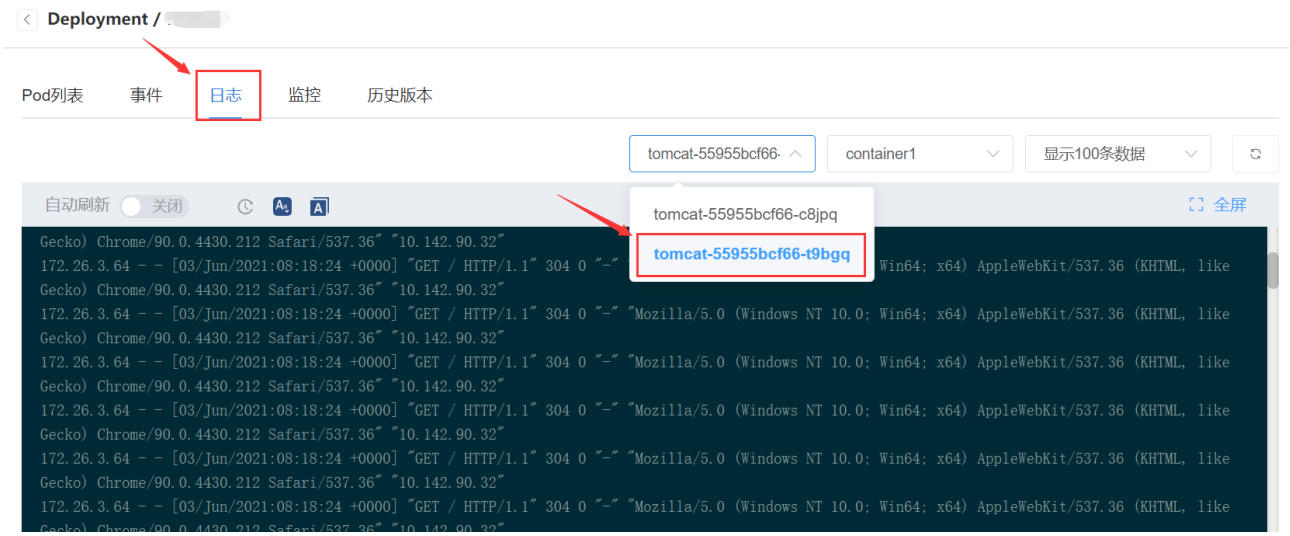
这里没法通过 curl 来测试验证，因为 curl 请求时没法保持 Cookie

3、观察工作负载日志

第一个 Pod 实例的日志输出：



第二个 Pod 实例的日志输出：



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

- **集群外 TCP/UDP 访问**

结论：集群外通过 TCP/UDP 没法进行会话保持

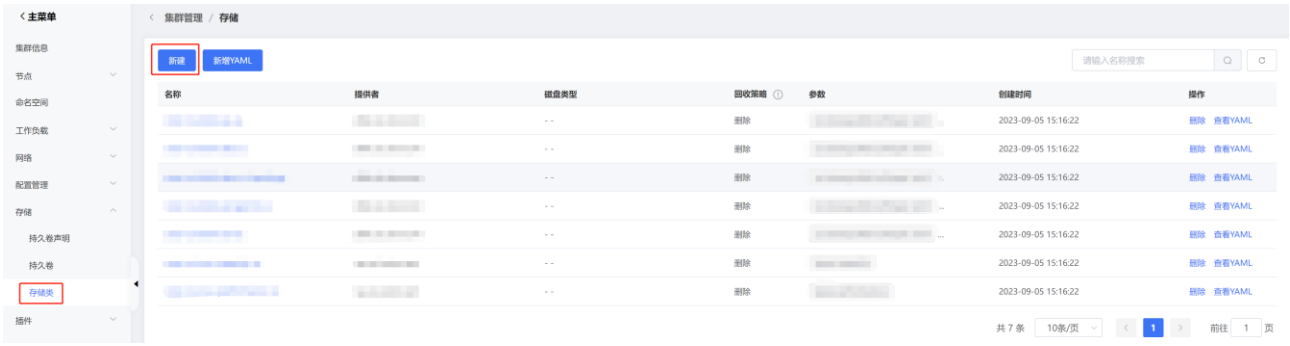
2. 参考

https://github.com/kubernetes/ingress-nginx/tree/master/docs/examples/affinity/cookie

6.3 容器化 Web 访问 MySQL（LocalPV）实践

本篇以实际场景介绍如何发布无状态工作负载，并连接有状态数据库，并配合使用持久存储，在本片教程中，我们使用 Local PV 作为持久存储介质，在生产使用中建议使用 Ceph 存储。

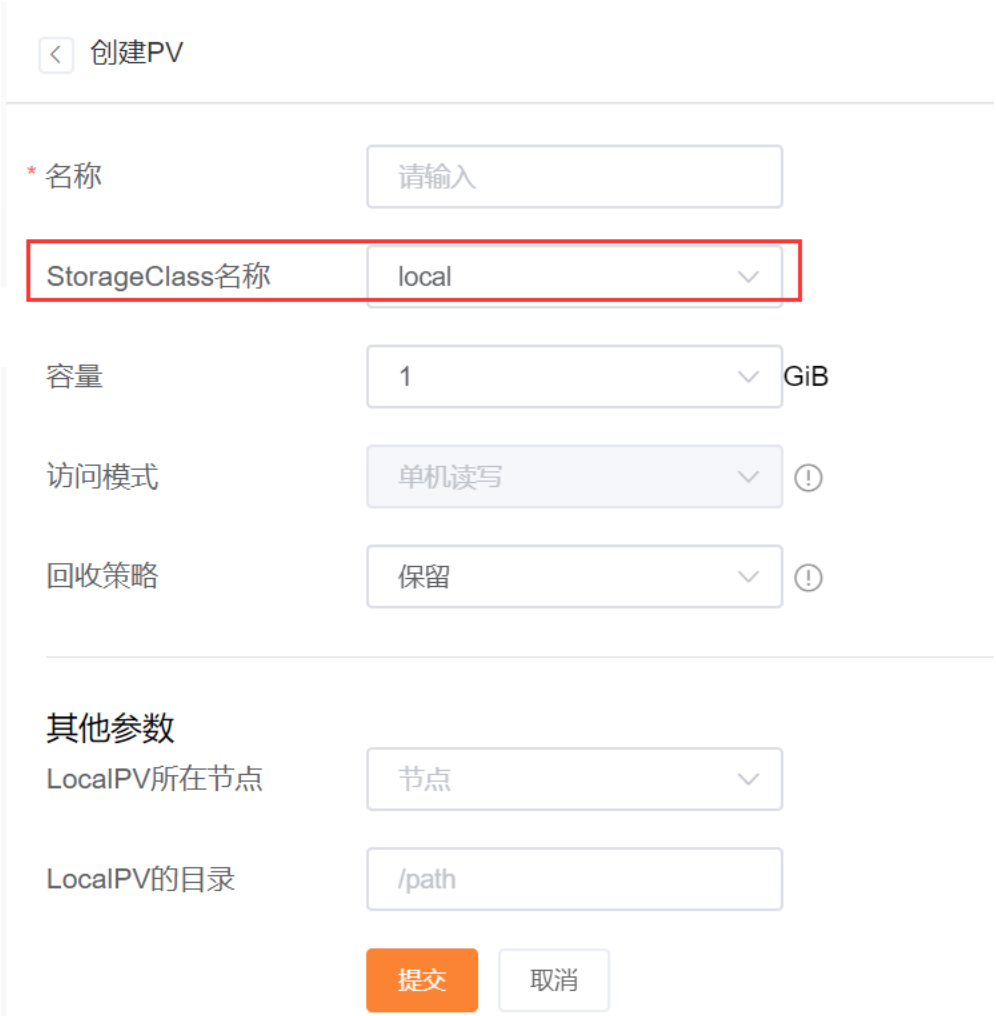
- **创建持久存储类**



在 CCSE 界面依次点击上图按钮，出现新建存储类界面，我们选择 Local 存储，如下：

创建持久存储卷：

在 CCSE 界面依次点击上图按钮，出现新建存储卷界面，我们选择第一步中创建的持久存储类，如下：



选择 LocalPV 所在节点以及 LocalPV 的目录，这里 LocalPV 所在节点即想要把容器数据持久化到哪一个 k8s 节点上，LocalPV 的目录即是我们所选节点上的指定目录。（注意：此目录需要在我们所选择节点上存在，若不存在需要手动在改节点上创建目录）

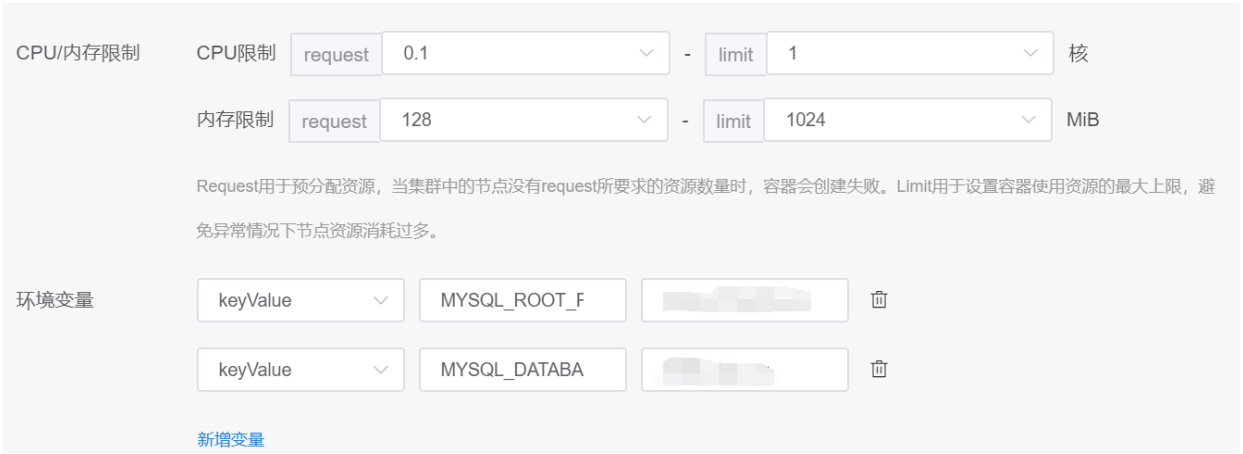
创建 MySQL 有状态服务并使用持久存储卷：

在 CCSE 界面依次点击上图按钮，出现新建存储卷界面，我们创建 MySQL 的流程如下：

点击新建 PVC 后，我们需要格外注意下图中的注意点。

然后我们开始设置 MySQL 容器的参数，依次输入 MySQL 容器名称、选择 MySQL 镜像及版本号，需要格外注意挂载点的名称和容器路径。

设置容器的资源参数及环境变量，MySQL 容器正常运行我们必须要设置 MYSQL_ROOT_PASSWORD 这个环境变量，设置 root 用户密码，同时如果需要在 MySQL 容器启动后帮我们创建一个 database，我们可以使用 MYSQL_DATABASE 这个环境变量。



最后一步，我们为 MySQL 容器配置集群内访问方式，所以我们选择类型为 ClusterIP。

- **创建 web 无状态工作负载并连接数据库**

1、web 工作负载配置

我们使用的 web 工作负载使用的配置文件如下：

```
spring:
  jpa:
    show-sql: false
    open-in-view: true
  datasource:
    driver-class-name: "com.mysql.cj.jdbc.Driver"
```

```
jdbc-url:
"jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/test?autoReconnect=true&useUnicode=true&characterEncoding=UTF-8&allowMultiQueries=true&useSSL=false"
username: "${MYSQL_USERNAME}"
password: "${MYSQL_PASSWORD}"
```

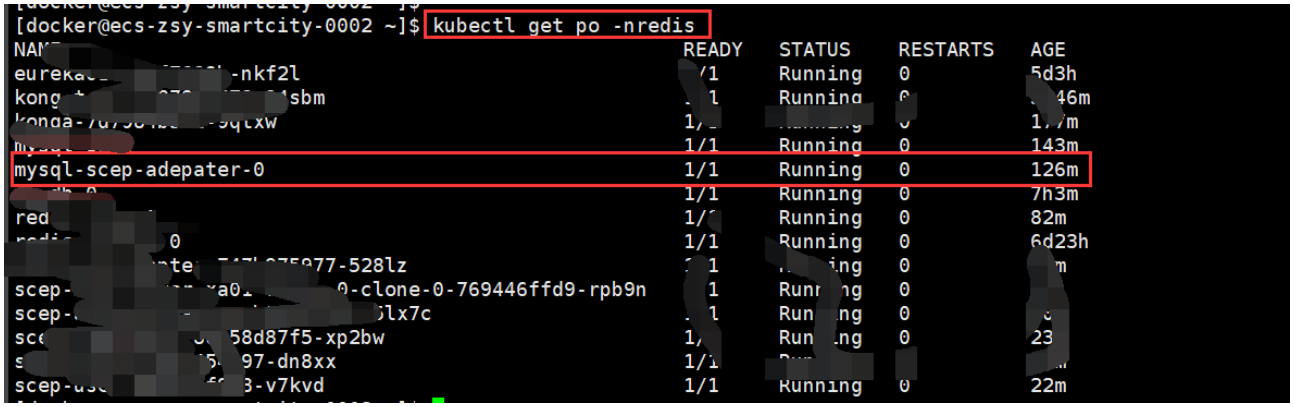
在上述配置中，我们通过读取环境变量来设置应用所需的 MySQL 主机 IP、端口、用户名、密码。

2、创建无状态工作负载

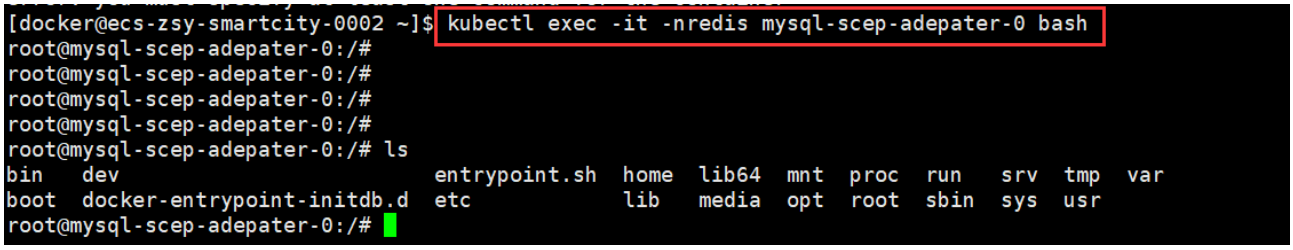
这里我们只介绍无状态工作负载的环境变量怎么设置，其他的参数配置与 MySQL 的类似：

- **查看 MySQL 容器（可选）**

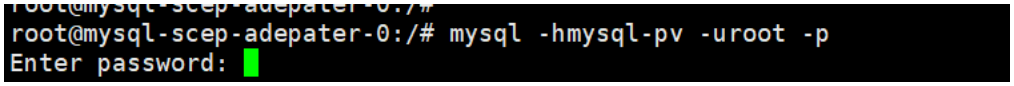
进入 k8s 集群，通过命令查看我们刚刚创建出来的 MySQL 容器



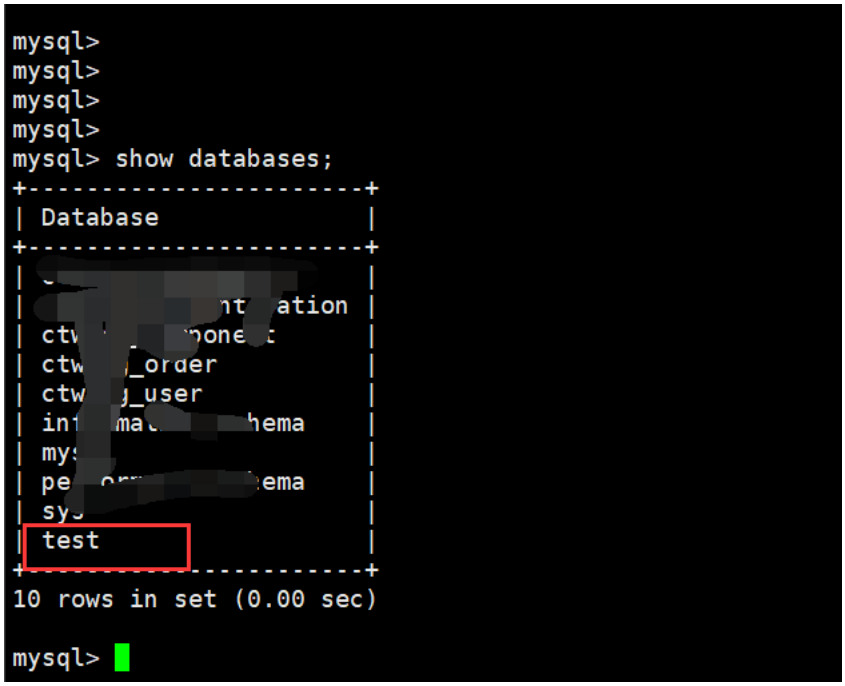
执行命令进入这个 MySQL 容器



执行 MySQL 命令连接 MySQL 客户端，此处的参数分别为：-h 后面的参数为我们在 3.1 步骤中创建的 MySQL 工作负载名称，-u 指定用户为 root 用户，-p 参数指定 root 用户的密码，指定的密码为我们在 3.1 中创建 MySQL 容器时指定的 MYSQL_ROOT_PASSWORD 这个环境变量的值。

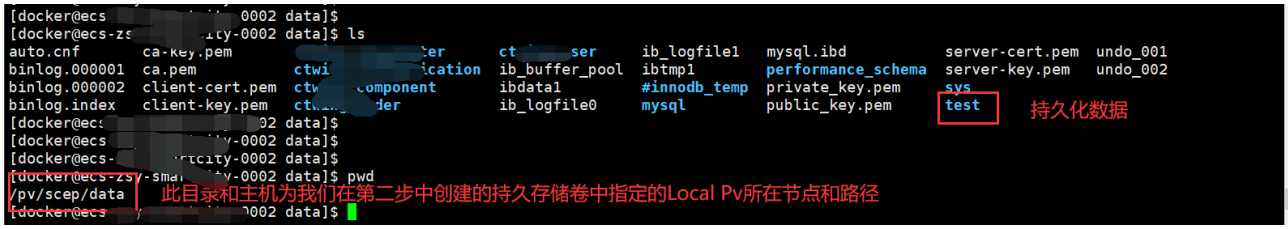


查看 MySQL 中的 database，即可看到我们在 3.1 中通过 MYSQL_DATABASE 这个环境变量创建的 database。



- **验证 MySQL 容器的持久存储（可选）**

创建一个 database 容器化 Web 访问 MySQL（LocalPV）连续指定两次 exit 命令分别退出 MySQL 客户端和 MySQL 容器



用于验证，我们可以在 CCSE 界面上重新部署 MySQL 容器，然后查看数据是否仍然存在，如下图：

6.4 服务与网络

6.4.1 CCSE 集群网络规划

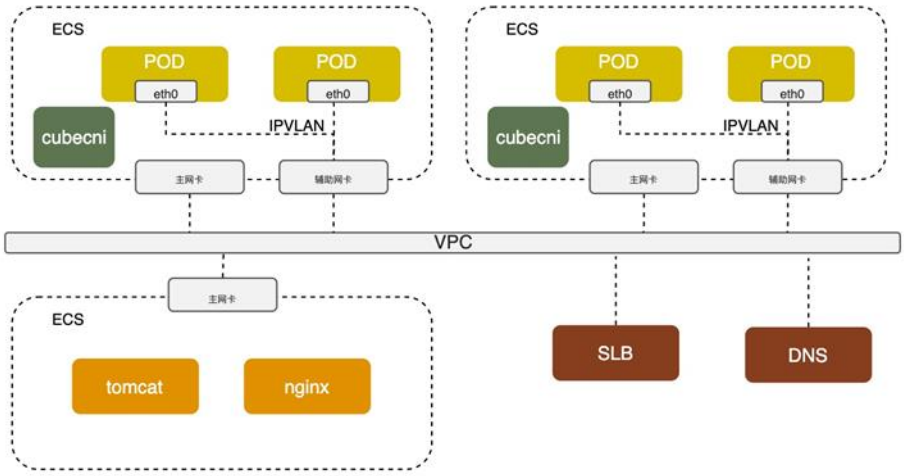
在创建 CCSE Kubernetes 集群时，您需要指定专有网络 VPC、Pod 网络 CIDR/Pod 子网（地址段）和 Service CIDR（地址段）。因此建议您提前规划 ECS 地址、Kubernetes Pod 地址和 Service 地址。本文将介绍天翼云专有网络 VPC 环境下 CCSE Kubernetes 集群里各种地址的作用，以及地址段该如何规划。

6.4.1.1 专有网络 VPC 网段和 Kubernetes 网段关系

专有网络 VPC（下文简称为 VPC 或专有网络）的网段规划包含 VPC 自身网段和子网网段，Kubernetes 网段规划包含 Pod 地址段和 Service 地址段。CCSE 网络支持 Cubecni 和 Calico 两种网络模式。

6.4.1.1.1 Cubecni 网络模式

Cubecni 网络模式图



配置 Cubecni 模式网络时，需要设置的参数及参数网段配置的注意事项如下：

专有网络：您在创建 VPC 时需要选择网段，建议从 10.0.0.0/16~20、172.16.0.0/16~20、192.168.0.0/16~20 三者中选择一个。

Pod 子网：Pod 地址从该子网分配，用于 Pod 网络通信。Pod 是 Kubernetes 内的概念，每个 Pod 具有一个 IP 地址。在 VPC 里创建子网时指定的网段，必须是当前 VPC 网段的子集。配置网段时，请注意：Cubecni 网络模式下，Pod 分配的 Pod IP 就是从这个子网网段内获取的。

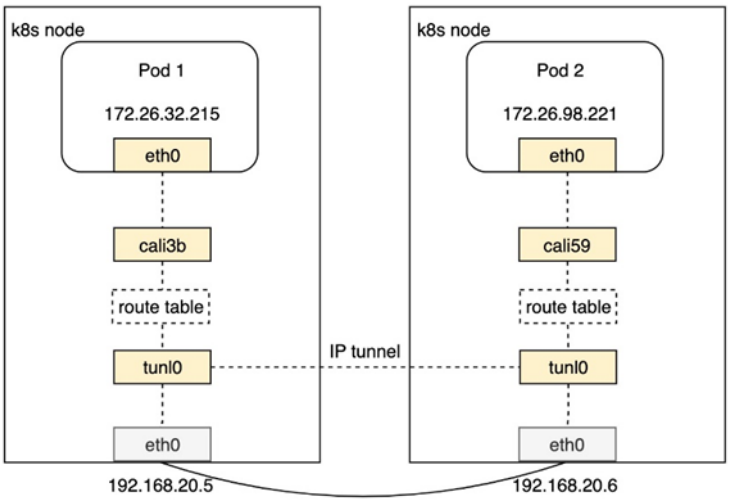
该地址段不能和 Service CIDR 网段重叠。

Service CIDR：Service 地址段。Service 是 Kubernetes 内的概念，对应的是 Service 类型为 ClusterIP 时 Service 使用的地址，每个 Service 有自己的地址。配置网段时，请注意：Service 地址只在 Kubernetes 集群内使用，不能在集群外使用。

Service 地址段不能和 Pod 子网地址段重叠。

6.4.1.1.2 Calico 网络模式

Calico 网络模式图



配置 Calico 模式网络时，需要设置的参数及参数网段配置的注意事项如下：

专有网络：您在创建 VPC 时需要选择网段，建议从 10.0.0.0/16~20、172.16.0.0/16~20、192.168.0.0/16~20 三者中选择一个。

Pod 网络 CIDR：Pod 网络 CIDR，Pod 地址从该地址段分配，用于 Pod 网络通信。Pod 是 Kubernetes 内的概念，每个 Pod 具有一个 IP 地址。配置网段时，请注意：非 VPC 子网，为虚拟网段。

该地址段不能和 Service CIDR 网段重叠。

例如，VPC 网段用的是 172.16.0.0/12，Kubernetes 的 Pod 地址段就不能使用 172.16.0.0/16、172.17.0.0/16 等，因为这些地址都包含在 172.16.0.0/12 里。

Service CIDR：Service 地址段。Service 是 Kubernetes 内的概念，对应的是 Service 类型为 ClusterIP 时 Service 使用的地址，每个 Service 有自己的地址。配置网段时，请注意：

Service 地址只在 Kubernetes 集群内使用，不能在集群外使用。

Service 地址段不能和 Pod 网络 CIDR 地址段重叠。

6.4.1.2 网络规划

在天翼云环境下使用 CCSE 支持的 Kubernetes 集群，首先需要根据业务场景、集群规模进行网络规划。您可以按下表规格进行规划（未包含场景，请根据实际需要自行调整）。

6.4.1.2.1 VPC 网络规划

集群节点规模	目的	VPC 规划	可用区
小于 100 个节点	一般性业务	单 VPC	1 个
任意	需要多可用区	单 VPC	3 个及以上
任意	对可靠性有极致要求、需要多地域	多 VPC	3 个及以上

6.4.1.2.2 容器网络规划

本文针对 Cubecni 和 Calico 网络场景，规划容器网络：

Cubecni 配置示例：Cubecni Pod IPVlan 模式

专有网络网段	Pod CIDR 网段	Service CIDR 网段	最大可分配 Pod 地址数
192.168.0.0/16	192.168.0.0/20	10.96.0.0/16	4090

Calico 配置示例

专有网络网段	Pod CIDR 网段	Service CIDR 网段	最大可分配 Pod 地址数
192.168.0.0/16	172.16.0.0/16	10.96.0.0/16	65536

6.4.1.3 如何选择地址段？

- 场景 1：单 VPC+单 Kubernetes 集群

  这是最简单的情形。VPC 地址在创建 VPC 的时候就已经确定，创建 Kubernetes 集群时，选择的 Pod 及 Service 地址网段和当前 VPC 不一样的地址段即可。

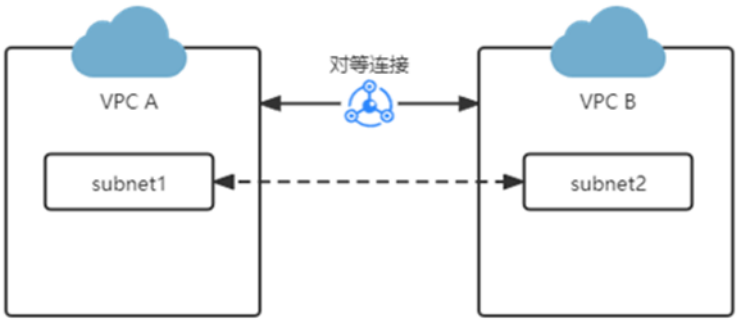
- 场景 2：单 VPC+多 Kubernetes 集群

  一个 VPC 下创建多个 Kubernetes 集群。

  - VPC 地址是在创建 VPC 时已经确定。创建 Kubernetes 集群时，每个集群内的 VPC 地址段、Service 地址段和 Pod 地址段彼此间不能重叠。
  - 所有 Kubernetes 集群之间的 Pod 地址段（Cubecni 模式下）不能重叠，但 Service 地址段可以重叠。

- 场景 3：VPC 互联

  两个 VPC 网络互联的情况下，可以通过对等连接使 VPC 互联。



在这种情况下，VPC A 和 VPC B 里创建的 Kubernetes 集群有以下限制：

- 不能和 VPC A 的地址段重叠
- 不能和 VPC B 的地址段重叠
- 不能和其他集群的地址段重叠
- 不能和 Pod 的地址段重叠
- 不能和 Service 的地址段重叠

6.4.2 使用 Cubecni 网络插件

Cubecni 是天翼云开源的基于专有网络 VPC 的容器网络接口 CNI（Container Network Interface）插件，支持基于 Kubernetes 标准的网络策略来定义容器间的访问策略。您可以通过使用 Cubecni 网络插件实现 Kubernetes 集群内部的网络互通。本文介绍如何使用天翼云云容器引擎 Kubernetes 版 CCSE 集群的 Cubecni 网络插件。

6.4.2.1 背景信息

Cubecni 网络插件是 CCSE 自研的网络插件，将原生的弹性网卡及子网 IP 分配给 Pod 实现 Pod 网络，支持基于 Kubernetes 标准的网络策略（Network Policy）来定义容器间的访问策略。

在 Cubecni 网络插件中，每个 Pod 都拥有自己网络栈和 IP 地址。同一台 ECS 内的 Pod 之间通信，直接通过机器内部的转发，跨 ECS 的 Pod 通信、报文通过 VPC 的弹性网卡直接转发。由于不需要使用 VxLAN 等的隧道技术封装报文，因此 Cubecni 模式网络具有较高的通信性能。

Cubecni 与 Calico 对比

在创建集群时，CCSE 提供 Cubecni 和 Calico 两种网络插件：

Cubecni：是天翼云云容器引擎 CCSE 自研的网络插件。CCSE 将天翼云的弹性网卡和子网 IP 分配给容器，支持基于 Kubernetes 标准的网络策略来定义容器间的访问策略。Cubecni 拥有更为灵活的 IPAM（容器地址分配）策略，避免地址浪费。如果您不需要使用实现容器与虚机互访，可以选择 calico，其他情况建议选择 Cubecni。

Calico：使用社区 Calico CNI 插件的 IP IP 模式。

6.4.2.2 Cubecni 与 Calico 对比

对比项	Cubecni	Calico
性能	性能接近弹性网卡	IP IP 解封包损耗
安全	支持使用网络策略 Network Policy。	支持使用网络策略 Network Policy。
地址管理	无需按节点分配地址段，随用随分配，地址无浪费。支持配置安全组。	每个节点提前分配虚拟地址段
ELB	ELB 后端不能直接对接 Pod，需要通过 NodePort 转发。	ELB 后端不能直接对接 Pod，需要通过 NodePort 转发。

6.4.2.3 配置

6.4.2.3.1 步骤一：规划和准备集群网络

在创建 CCSE Kubernetes 集群时，您需要指定专有网络 VPC、Pod 子网（地址段）和 Service CIDR（地址段）。

您可以使用以下网段配置，以快速搭建 Cubecni 网络。

专有网络网段	Pod 子网	Service CIDR
192.168.0.0/16	192.168.0.0/20	172.21.0.0/20

本文使用上述推荐网段，说明如何创建一个专有网络和 Pod 子网。

登录专有网络管理控制台。

在顶部菜单栏处，选择专有网络的地域，然后单击创建专有网络。

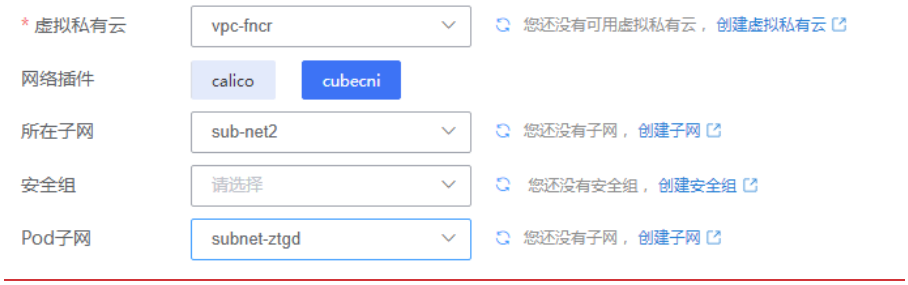
在创建专有网络页面，设置名称为 vpc_192_168_0_0_16，在 IPv4 网段文本框中，输入 192.168.0.0/16。

单击+添加，设置第二个交换机名称为 pod_switch_192_168_32_0_19，选择可用区，设置 IPv4 网段为 192.168.32.0/20。

单击确定。

6.4.2.3.2 步骤二：配置 Cubecni 网络

为 Cubecni 网络插件配置集群网络的关键参数配置说明如下：



虚拟私有云：选择步骤一：规划和准备集群网络中创建的专有网络。

网络插件：选择 Cubecni

Pod 子网：选择步骤一：规划和准备集群网络中创建的 Pod 子网。

Service CIDR：保留默认值。

6.4.2.4 Cubecni IPVlan 模式

在创建集群时，如果选择 Cubecni 网络插件，则使用 Cubecni IPvlan 模式。Cubecni IPvlan 模式采用 IPvlan 虚拟化实现高性能的 Pod 和 Service 网络：

不同于默认的 Terway 的网络模式，IPvlan 模式主要在 Pod 网络、Service、网络策略（NetworkPolicy）做了性能的优化：

- Pod 的网络直接通过 ENI 网卡的 IPvlan L2 的子接口实现，大大简化了网络在宿主机上的转发流程，让 Pod 的网络性能几乎与宿主机的性能无异，延迟相对传统模式降低 30%。
- Pod 的网络策略（NetworkPolicy）采用 eBPF 替换掉原有的 iptables 的实现，不需要在宿主机上产生大量的 iptables 规则，降低网络策略对网络性能的影响。

6.4.3 通过注解配置负载均衡

通过 Service YAML 文件中的 Annotation（注解），可以实现丰富的负载均衡功能。本文从 ELB、监听和后端服务器组三种资源维度介绍通过注解可以对 ELB 进行的常见配置操作。

6.4.3.1 注解使用说明

- 注解的内容是区分大小写。
- annotations 字段格式 service.beta.kubernetes.io/xxx

6.4.3.2 ELB

ELB 的典型操作

- 指定负载均衡的规格

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ctyun-loadbalancer-spec: "${负载均衡的规格 ID}"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
```

```
    protocol: TCP
    targetPort: 443
  selector:
    name: nginx
  type: LoadBalancer
```

- 使用一个已有的负载均衡

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ctyun-loadbalancer-id: "${负载均衡 ID}"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    name: nginx
```

```
type: LoadBalancer
```

6.4.3.3 常用注解

- ELB 常用注解

注解	类型	描述	默认值	支持的 CCM 版本
service.beta.kubernetes.io/ctyun-loadbalancer-address-type	String	创建公网类型或者私网类型的 ELB，取值可以是 internet 或者 intranet	intranet	v1.0.0 及以上
service.beta.kubernetes.io/ctyun-loadbalancer-id	String	指定已有负载均衡实例的 ID。删除 service 时该 ELB 不会被删除。	无	v1.0.0 及以上
service.beta.kubernetes.io/ctyun-loadbalancer-spec	String	负载均衡的规格	无	v1.0.0 及以上

service.beta.kubernetes.io/ctyun-loadbalancer-resource-pack-id	String	开通 ELB 所使用的资源包 ID	无	v1.0.0 及以上

6.4.4 基于 DNS 的服务发现

DNS 为 Kubernetes 集群内的工作负载提供域名解析服务。本文主要介绍 Kubernetes 集群中 DNS 域名解析原理和 CCSE 集群中默认内置的 DNS 服务器 CoreDNS。

6.4.4.1 Kubernetes 集群中 DNS 域名解析原理

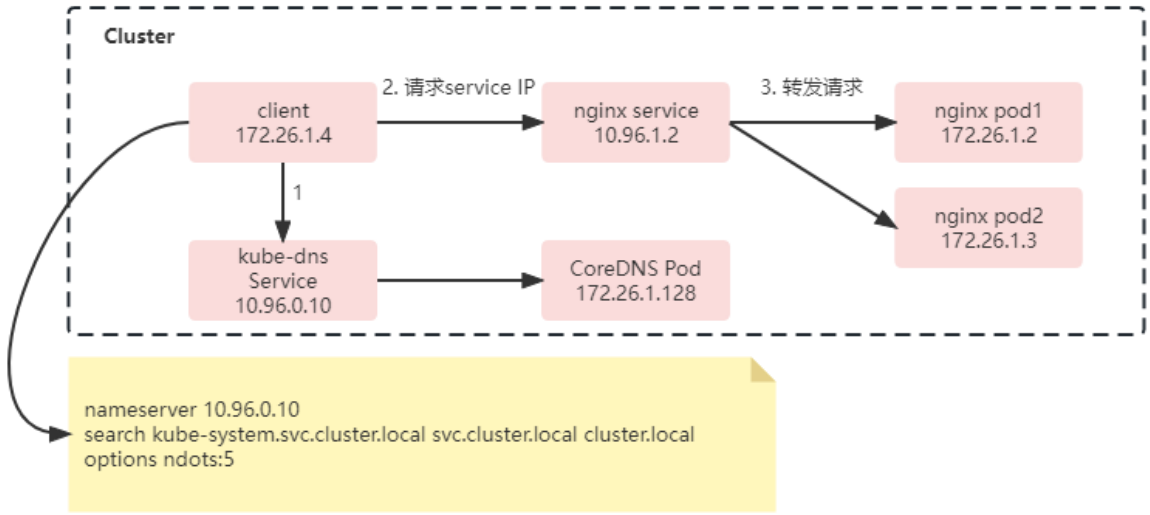
CCSE 集群中 kubelet 的 config.yaml 有 clusterDNS 和 clusterDomain，这两个参数分别被用来设置集群 DNS 服务器的 IP 地址和主域名后缀。

CCSE 集群默认部署了一套 CoreDNS 工作负载，并通过 kube-dns 的服务名暴露 DNS 服务。CCSE 部署的 CoreDNS 工作负载后端是两个名为 coredns 的 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。Pod 内的 DNS 域名解析配置文件为/etc/resolv.conf，文件内容如下。

```
nameserver 10.xx.x.xx

search kube-system.svc.cluster.local svc.cluster.local cluster.local

options ndots:5
```



序号	描述
1	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 10.96.1.2 的 IP 地址。
2	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。

6.4.4.2 CoreDNS 介绍

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，请参见 CoreDNS: DNS and Service Discovery。

CCSE 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。

6.5 存储与安全

6.5.1 存储基础知识

天翼云云容器引擎 CCSE 使用 Kubernetes 编排系统作为集群、应用、存储、网络等模块的管理平台。本文为您介绍 CCSE 容器存储相关的基础知识，以便在使用容器服务的存储能力时，了解相应模块的基础知识和使用原则。

6.5.1.1 数据卷（Volume）

因为容器中的文件在磁盘上是临时存放的，所以 Kubernetes 定义了数据卷（Volume）以解决容器中的文件因容器重启而丢失的问题。数据卷（Volume）是 Pod 与外部存储设备进行数据传递的通道，也是 Pod 内部容器间、Pod 与 Pod 间、Pod 与外部环境进行数据共享的方式。

数据卷（Volume）定义了外置存储的细节，并内嵌到 Pod 中作为 Pod 的一部分。其实质是外置存储在 Kubernetes 系统的一个资源映射，当负载需要使用外置存储的时候，可以从数据卷（Volume）中查到相关信息并进行存储挂载操作。

数据卷（Volume）分类	描述

本地存储	适用于本地存储的数据卷，例如 HostPath、emptyDir 等。本地存储卷的特点是数据保存在集群的特定节点上，并且不能随着应用漂移，节点停机时数据即不再可用。
网络存储	适用于网络存储的数据卷，例如 Ceph、GlusterFS、NFS、iSCSI 等。网络存储卷的特点是数据不在集群的某个节点上，而是在远端的存储服务上，使用存储卷时需要将存储服务挂载到本地使用。
Secret 和 ConfigMap	Secret 和 ConfigMap 是特殊的数据卷，其数据是集群的一些对象信息，该对象数据以卷的形式被挂载到节点上供应用使用。
PVC	一种数据卷定义方式，将数据卷抽象成一个独立于 Pod 的对象，这个对象定义（关联）的存储信息即存储卷对应的真正存储信息，供 Kubernetes 负载挂载使用。

数据卷（Volume）使用原则：

- 一个 Pod 可以挂载多个数据卷（Volume）。
- 一个 Pod 可以挂载多种类型的数据卷（Volume）。
- 每个被 Pod 挂载的 Volume 卷，可以在不同的容器间共享。
- Kubernetes 环境推荐使用 PVC 和 PV 方式挂载数据卷（Volume）。
- 虽然单 Pod 可以挂载多个数据卷（Volume），但是并不建议给一个 Pod 挂载过多数据卷。

6.5.1.2 PV 和 PVC

并非所有的 Kubernetes 数据卷（Volume）具有持久化特征，为了实现持久化的实现，容器存储需依赖于一个远程存储服务。为此 Kubernetes 引入了 PV 和 PVC 两个资源对象，将存储实现的细节从其如何被使用中抽象出来，并解耦存储使用者和系统管理员的职责。PV 和 PVC 的概念如下：

PV，PV 是 PersistentVolume 的缩写，译为持久化存储卷。PV 在 Kubernetes 中代表一个具体存储类型的卷，其对象中定义了具体存储类型和卷参数。即目标存储服务所有相关的信息都保存在 PV 中，Kubernetes 引用 PV 中的存储信息执行挂载操作。

PVC，PVC 是 PersistentVolumeClaim 的缩写，译为存储声明。PVC 是在 Kubernetes 中一种抽象的存储卷类型，代表了某个具体类型存储的数据卷表达。其设计意图是分离存储与应用编排，将存储细节抽象出来并实现存储的编排。这样 Kubernetes 中存储卷对象独立于应用编排而单独存在，在编排层面使应用和存储解耦。

PV 和 PVC 使用说明

- PVC 和 PV 的绑定

  PVC 与 PV 是一一对应关系，不能一个 PVC 挂载多个 PV，也不能一个 PV 挂载多个 PVC。为应用配置存储时，需要声明一个存储需求声明（PVC），而 Kubernetes 会通过最佳匹配的方式选择一个满足 PVC 需求的 PV，并与之绑定。所以从职责上 PVC 是应用所需要的存储对象，属于应用作用域。PV 是存储平面的存储对象，属于整个存储域。

  PVC 只有绑定了 PV 之后才能被 Pod 使用，而 PVC 绑定 PV 的过程即是消费 PV 的过程，这个过程是有一定规则的，以下规则都满足的 PV 才能被 PVC 绑定。

- VolumeMode：被消费 PV 的 VolumeMode 需要和 PVC 一致。
- AccessMode：被消费 PV 的 AccessMode 需要和 PVC 一致。
- StorageClassName：如果 PVC 定义了此参数，PV 必须有相关的参数定义才能进行绑定。
- LabelSelector：通过标签（labels）匹配的方式从 PV 列表中选择合适的 PV 绑定。
- Size：被消费 PV 的 capacity 必须大于或者等于 PVC 的存储容量需求才能被绑定。

- PV 和 PVC 定义中的 size 字段

  PVC 和 PV 里面的 size 字段作用如下：

  - PVC、PV 绑定时，会根据各自的 size 进行筛选。
  - 通过 PVC、StorageClass 动态创建 PV 时，有些存储类型会参考 PVC 的 size 创建相应大小的 PV 和后端存储。
  - 对于支持 Resize 操作的存储类型，PVC 的 size 作为扩容后 PV、后端存储的容量值。

  一个 PVC、PV 的 size 值只是在执行一些 PVC 和 PV 管控操作的时候，作为配置参数来使用。

  真正的存储卷数据流写数据的时候，不会参考 PVC 和 PV 的 size 字段，而是依赖底层存储介质的实际容量。

6.5.1.3 数据卷（Volume）使用方式

常见的容器存储数据卷使用方式如下：

- 静态存储卷

静态存储卷一般是指由管理员创建的 PV。所有的数据卷（Volume）都支持创建静态存储卷。

静态存储卷先由集群管理员分析集群中存储需求，并预先分配一些存储介质（例如云盘、NAS 盘等），同时创建对应的 PV 对象。创建好的 PV 对象等待 PVC 来消费。如果负载中定义了 PVC 需求，Kubernetes 会通过相关规则实现 PVC 和匹配的 PV 进行绑定，这样就实现了应用对存储服务的访问能力。

- 动态存储卷

  动态存储卷一般是指通过存储插件自动创建的 PV。

  动态卷一般由集群管理员配置好后端的存储池，并创建相应的模板（StorageClass），当有 PVC 需要消费 PV 的时候，根据 PVC 定义的需求，并参考 StorageClass 的存储细节，由存储插件动态创建一个 PV。StorageClass 的定义如下：

  - StorageClass

    当您声明一个 PVC 时，如果在 PVC 中添加了 StorageClassName 字段，意味着当 PVC 在集群中找不到匹配的 PV 时，会根据 StorageClassName 的定义触发相应的 Provisioner 插件创建合适的 PV 供绑定，即创建动态数据卷。

    动态数据卷由 Provisioner 插件创建，并通过 StorageClassName 与 PVC 进行关联。

    StorageClass 可译为存储类，表示为一个创建 PV 存储卷的模板。在 PVC 触发自动创建 PV 的过程中，即使用 StorageClass 对象中的内容进行创建。其内容如下：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: alicloud-disk-topology
parameters:
  type: cloud_ssd
provisioner: diskplugin.csi.ctyun.com
reclaimPolicy: Delete
      allowVolumeExpansion: true

volumeBindingMode: WaitForFirstConsumer
```

参数	描述
reclaimPolicy	用来指定创建 PV 的 persistentVolumeReclaimPolicy 字段值，支持 Delete 和 Retain。 ● Delete 表示动态创建的 PV，在销毁的时候也会自动销毁。 ● Retain 表示动态创建的 PV，不会自动销毁，而是由管理员处理。
allowVolumeExpansion	定义由此存储类创建的 PV 是否运行动态扩容，默认为 false。是否能动态扩容是由底层存储插件来实现的，这里只是一个开关。
volumeBindingMode	表示动态创建 PV 的时间，支持 Immediate 和 WaitForFirstConsumer，分别表示立即创建和延迟创建。

- 延迟绑定

  针对某类存储（例如天翼云云盘）在挂载属性上有所限制，只能挂载相同可用区的数据卷和节点，不在同一个可用区不可以挂载。这种类型的存储卷通常遇到如下问题：

  - 创建了 A 可用区的数据卷，但是 A 可用区的节点资源已经耗光，导致 Pod 启动无法完成挂载。
  - 集群管理员在规划 PVC、PV 的时候不能确定在哪些可用区创建多个 PV 来备用。

    StorageClass 中的 volumeBindingMode 字段正是用来解决此问题，如果将 volumeBindingMode 配置为 WaitForFirstConsumer 值，则表示存储插件在收到 PVC Pending 的时候不会立即进行数据卷创建，而是等待这个 PVC 被 Pod 消费的时候才执行创建流程。

  动态存储卷的优势

  - 动态存储卷让 Kubernetes 实现了 PV 的自动化生命周期管理，PV 的创建、删除都通过 Provisioner 完成。
  - 自动化创建 PV 对象，减少了配置复杂度和系统管理员的工作量。
  - 动态卷可以实现 PVC 对存储的需求容量和 Provisioner 出来的 PV 容量一致，实现存储容量规划最优。

- 挂在 SubPath 卷

  Kubernetes 提供了 volumeMounts.subPath 属性配置，可用于指定所引用的卷内的子路径，而不是其根路径。

6.5.1.4 存储卷配额说明

- 部分存储类型在每个节点的挂载数量是有限制的。

6.6 可观测性

6.6.1 天翼云 ARMS 监控

您可以通过开通天翼云 ARMS，在 CCSE 控制台查看监控大盘和监控性能指标。本文为您介绍如何在 CCSE 中接入天翼云 ARMS 监控。

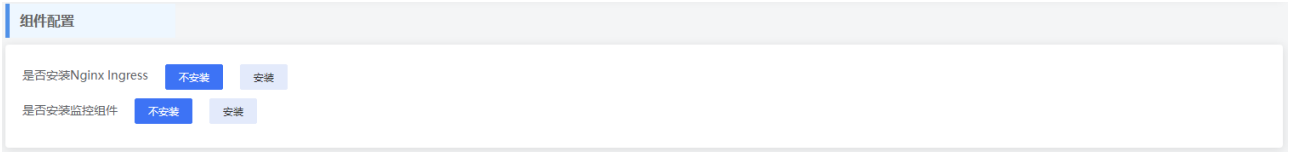
6.6.1.1 背景信息

天翼云 ARMS 监控全面对接开源 Prometheus 生态，支持类型丰富的组件监控，提供全面托管的 Prometheus 服务。借助天翼云 ARMS 的 Prometheus 监控，您无需自行搭建 Prometheus 监控系统，因而无需关心底层数据存储、数据展示、系统运维等问题。

6.6.1.2 开启天翼云 ARMS 监控

6.6.1.2.1 方式一：创建集群时开启

在创建集群的组件配置页面，选中安装监控插件。



6.6.1.2.2 方式二：在已有的集群中开启

- 登录容器服务管理控制台，在左侧导航栏中选择集群。
- 在集群列表页面中，单击目标集群名称，然后在左侧导航栏中，选择监控或者插件 > 插件市场。
- 在监控页面或者插件市场，单击开始安装 ccse-monitor。

  控制台会自动安装组件。安装完成后，单击各个页签查看相应监控数据。

6.6.2 基础资源监控

资源监控是 Kubernetes 中最常见的监控方式，通过资源监控可以快速查看负载的 CPU、内存、网络等指标的使用率。在天翼云云容器引擎中，资源监控已经与 ARMS 互通。本文介绍如何在控制台查看基础资源的监控。

6.6.2.1 前提条件

若需要使用基础资源监控，需开通天翼云 ARMS，并在 CCSE 控制台插件市场中安装 ccse-monitor 插件。

6.6.2.2 功能特性

- 提供全集群视角指标，洞悉集群概况。包括集群、节点、命名空间和工作负载。
- 更合适的容器场景指标。

  在宿主机基础设施层、容器 PaaS 层及 Kubernetes 调度层不同场景下使用最合适的指标。例如，容器中影响 Kubernetes 调度的内存指标，会使用容器工作内存的专用指标，与宿主机的内存 Usage 区分。

6.6.2.3 查看资源监控

- 登录 CCSE 控制台。
- 选择指定的 CCSE 集群。
- 在菜单“集群信息”的“概览”tab 中可以查看相关的监控信息。
- 在菜单“监控”中可以查看“集群资源监控”、“命名空间监控”、“节点资源监控”、“APIServer 监控”、“调度器组件监控”和“ETCD 组件监控”。
- 在菜单“节点”的列表页中点击“监控”按钮可以查看“节点物理资源监控”和“容器组监控”。
- 在菜单“命名空间”的列表页中点击“监控”按钮可以查看“命名空间资源监控”。
- 在菜单“工作负载”的列表页中点击工作负载进入详情页，点击“监控”tab 页可以查看 Pod 和 Container 的资源信息。

6.6.3 K8S 容器日志采集

天翼云云容器引擎 CCSE Kubernetes 集群集成了日志服务 ALS，您可在创建集群时启用日志服务，快速采集 Kubernetes 集群的容器日志，包括容器的标准输出以及容器内的文本文件。

6.6.3.1 步骤一：启用日志服务组件 ALS

您可以为已有集群启用 ALS 组件。

为已有集群启用 ALS

- 登录云容器引擎 CCSE 管理控制台。
- 在控制台左侧导航栏中，单击集群。
- 在集群列表页面中，单击目标集群名称。
- 在集群管理页左侧导航栏中，选择插件 > 插件市场，并在日志与监控区域找到 ctg-log-operator。
- 在 log-operator 上单击安装。
- 在安装插件对话框中选择插件版本和超时时间，单击安装。

6.6.3.2 步骤二：创建应用时配置日志服务

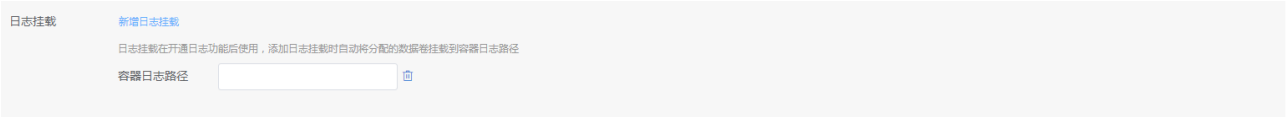
您可以在创建应用的同时配置日志服务 ALS，从而对容器的日志进行采集。目前支持控制台向导和 YAML 两种方式创建应用。

通过控制台向导创建应用并配置日志服务

- 登录容器服务管理控制台。
- 在控制台左侧导航栏中，单击集群。
- 在集群列表页面中，单击目标集群名称。
- 在集群管理页左侧导航栏中，选择工作负载 > 无状态。
- 在无状态页面上方的命名空间下拉框中设置命名空间，然后单击页面左上角的新增按钮。
- 在应用基本信息页签，设置应用名称、数据卷和实例数量等。

  以下仅介绍日志服务 ALS 相关的配置。

- 进行日志配置。单击“新增日志挂在”创建新的采集配置，每个采集配置由容器日志路径构成
  - 容器内日志路径：您可以用它来指定希望采集的日志所在的路径，如使用/usr/local/tomcat/logs/catalina.log 来采集 tomcat 的文本日志。



- 当完成所有配置后，可单击右上角的下一步进入后续流程。

通过 YAML 创建

- 登录容器服务管理控制台。
- 在控制台左侧导航栏中，单击集群。
- 在集群列表页面中，单击目标集群名称。
- 在集群管理页左侧导航栏中，选择工作负载 > 无状态。
- 在无状态页面上方的命名空间下拉框中设置命名空间，然后单击页面左上角的新增 YAML 按钮。
- YAML 模板的语法同 Kubernetes 语法，但是为了给容器指定采集配置，需要使用 env 来为容器增加采集配置和自定义 Tag，并根据采集配置，创建对应的 volumeMounts 和 volumes。以下是一个简单的 Pod 示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: my-demo
  annotations:
    ctyun.sls.logs: '[{"sls.capture.type":"stdout"},
{"sls.capture.type":"applog","sls.log.path":"/usr/local/tomcat/logs/catali
na.log"}]'
```

```
spec:
  containers:
  - name: my-demo-app
    image:              'registry-nm6b-crs.ctyun.com/log-service/docker-log-
test:latest'
    ######### 配置 volume mount ###########
    volumeMounts:
    - name: volumn-sls-mydemo
      mountPath: usr/local/tomcat/logs/catalina.log
  volumes:
  - name: volumn-sls-mydemo
    emptyDir: {}

  ################################
```

6.6.3.3 步骤三：查看日志

本例中查看通过控制台向导创建的 tomcat 应用的日志。完成配置后，tomcat 应用的日志已被采集并存储到日志服务 ALS 中，您可以在 CCSE 控制台查看容器日志。操作步骤如下：

- 安装成功后，进入 CCSE 控制台。
- 登录云容器引擎 CCSE 管理控制台。
- 在控制台左侧导航栏中，单击集群。
- 在集群列表页面中，单击目标集群名称。
- 在集群管理页左侧导航栏中，选择运维管理 > 日志中心

- 本例中，在日志查询页面，您可查看 tomcat 应用的标准输出日志和容器内文本日志。

6.7 发布

6.7.1 灰度发布和蓝绿发布

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 CCSE 集群中如何通过 Nginx Ingress Controller 来实现应用服务的灰度发布。

6.7.1.1 背景信息

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

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

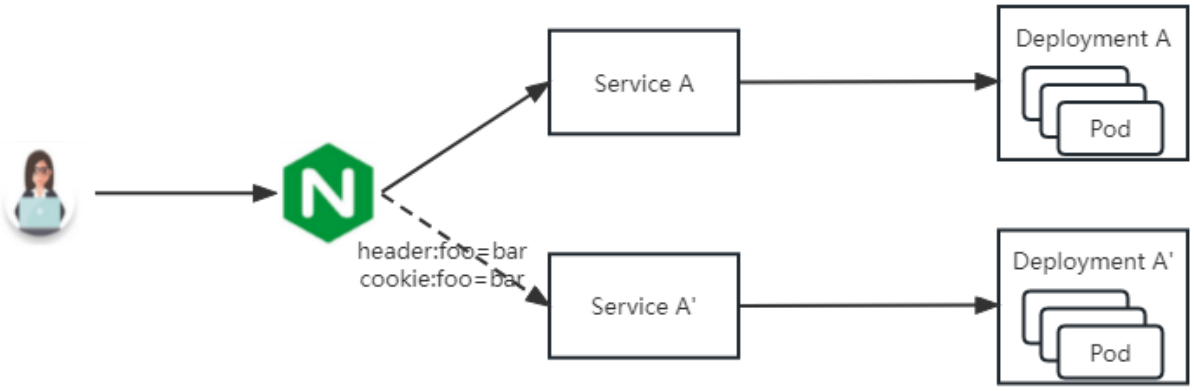
云容器引擎 CCSE 控制台灰度发布功能的用法如下。

canary-*注解方式：使用 canary-* Annotation 配置蓝绿发布与灰度发布，canary-* Annotation 是社区官方实现的灰度发布方式。

6.7.1.2 应用场景

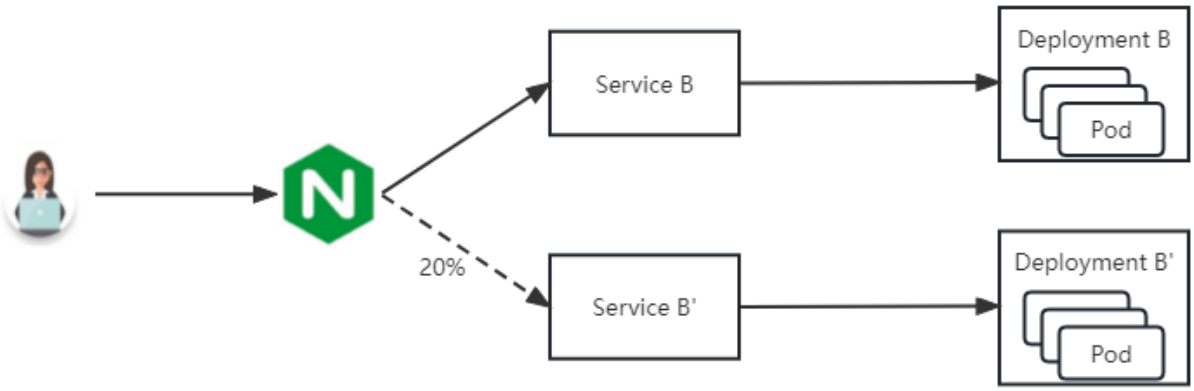
基于客户端请求的流量切分场景

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



基于服务权重的流量切分场景

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



针对以上多种不同的应用发布需求，天翼云云容器引擎 Ingress Controller 支持了多种流量切分方式：

基于 Request Header 的流量切分，适用于灰度发布以及 AB 测试场景。

基于 Cookie 的流量切分，适用于灰度发布以及 AB 测试场景。

基于 Query Param 的流量切分，适用于灰度发布以及 AB 测试场景。

基于服务权重的流量切分，适用于蓝绿发布场景。

6.7.1.3 canary-*注解方式

6.7.1.3.1 注解说明

Nginx Ingress Controller 通过下列 canary-* Annotation 来支持应用服务的灰度发布机制。

Annotation	说明	适用的 CCSE

		Nginx Ingress Controller 版本
nginx.ingress.kubernetes.io/canary	必须设置该 Annotation 值为 true，否则其它规则将不会生效。 取值： true：启用 canary 功能。 false：不启用 canary 功能。	≥v1.3.1
nginx.ingress.kubernetes.io/canary-by-header	表示基于请求头的名称进行灰度发布。 请求头名称的特殊取值： always：无论什么情况下，流量均会进入灰度服务。 never：无论什么情况下，流量均不会进入灰度服务。 若没有指定请求头名称的值，则只要该头存在，都会进行流量转发。	≥v1.3.1
nginx.ingress.kubernetes.io/canary-by-header-value	表示基于请求头的值进行灰度发布。 需要与 canary-by-header 头配合使用。	≥v1.3.1
nginx.ingress.kubernetes.io/canary-by-cookie	表示基于 Cookie 进行灰度发布。 Cookie 名称的特殊取值： always：无论什么情况下，流量均会进入灰度服务。 never：无论什么情况下，流量均不会进入灰度服	≥v1.3.1

	务。 只要存在该 Cookie 名称，都会进行流量转发。	
nginx.ingress.kubernetes.io/canary-weight	表示基于权重进行灰度发布。 取值范围：0~权重总值。 若未设定总值，默认总值为 100。	≥v1.3.1

6.7.1.3.2 操作方式

步骤一：部署服务

部署 Nginx 服务并通过 Nginx Ingress Controller 对外提供 7 层域名访问。

创建 Deployment 和 Service。

创建 nginx.yaml。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: old-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: old-nginx
  template:
    metadata:
      labels:
        run: old-nginx
    spec:
      containers:
      - image: registry- nm6b -crs.ctyun.com/ccse-sample/old-nginx
        imagePullPolicy: Always
        name: old-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
```

```
---
apiVersion: v1
kind: Service
metadata:
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: old-nginx
  sessionAffinity: None
  type: NodePort
```

执行以下命令，创建 Deployment 和 Service。

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

部署 Ingress

创建 ingress.yaml。

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

kind: Ingress

metadata:

   name: gray-release

spec:

   rules:

   - host: www.example.com
```

```
    http:
      paths:
      # 老版本服务。
      - path: /
        backend:
          service:
            name: old-nginx
            port:
              number: 80
        pathType: ImplementationSpecific
```

执行以下命令，部署 Ingress。

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

测试访问情况。

执行以下命令，获取外部 IP。

```
kubectl get ingress
```

执行以下命令，查看路由访问情况。

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

期望输出

```
old
```

步骤二：灰度发布新版本服务

发布一个新版本的 Nginx 服务并配置路由规则。

部署新版本的 Deployment 和 Service。

创建 nginx1.yaml。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: new-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: new-nginx
  template:
    metadata:
      labels:
        run: new-nginx
    spec:
      containers:
      - image: registry-nm6b-crs.ctyun.com/ccse-sample/new-nginx
        imagePullPolicy: Always
        name: new-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: new-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
```

```
    run: new-nginx
  sessionAffinity: None
  type: NodePort
```

执行以下命令，创建 Deployment 和 Service。

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

设置访问新版本服务的路由规则。

CCSE 支持设置以下三种路由规则，您可以根据实际情况选择路由规则。

设置满足特定规则的客户端才能访问新版本服务。以下示例仅请求头中满足 foo=bar 的客户端请求才能路由到新版本服务。

按照以上条件，创建新的 Ingress 资源 gray-release-canary。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gray-release-canary
  annotations:
    # 开启 Canary。
    nginx.ingress.kubernetes.io/canary: "true"
    # 请求头为 foo。
```

```
    nginx.ingress.kubernetes.io/canary-by-header: "foo"
    # 请求头 foo 的值为 bar 时，请求才会被路由到新版本服务 new-nginx
中。
    nginx.ingress.kubernetes.io/canary-by-header-value: "bar"
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # 新版本服务。
      - path: /
        backend:
          service:
            name: new-nginx
            port:
              number: 80
        pathType: ImplementationSpecific
```

查看路由访问情况。

执行以下命令，访问服务。

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

预期输出：

```
old
```

执行以下命令，请求头中满足 foo=bar 的客户端请求访问服务。

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

预期输出：

```
new
```

在满足特定规则的基础上设置一定比例的请求被路由到新版本服务中。以下示例要求请求头中满足 foo=bar 的客户端请求，且仅允许其中 50%的流量被路由到新版本服务中。

按照以下内容，修改步骤 2 中创建的 Ingress。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: gray-release-canary
  annotations:
    # 开启 Canary。
    nginx.ingress.kubernetes.io/canary: "true"
```

```
    # 请求头为 foo。
    nginx.ingress.kubernetes.io/canary-by-header: "foo"
    # 请求头 foo 的值为 bar 时，请求才会被路由到新版本服务 new-nginx 中。
    nginx.ingress.kubernetes.io/canary-by-header-value: "bar"
    # 在满足上述匹配规则的基础上仅允许 50%的流量会被路由到新版本服务 new-nginx 中。
    nginx.ingress.kubernetes.io/canary-weight: "50"
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # 新版本服务。
      - path: /
        backend:
          service:
            name: new-nginx
            port:
              number: 80
        pathType: ImplementationSpecific
```

查看路由访问情况。

执行以下命令，访问服务。

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

预期输出：

```
old
```

执行以下命令，请求头中满足 foo=bar 的客户端请求访问服务。

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

预期输出：

```
new
```

重复执行以上命令。可以看到，仅请求头中满足 foo=bar 的客户端请求，且只有 50%的流量才能路由到新版本服务。

重复执行以上命令。可以看到，仅请求头中满足 foo=bar 的客户端请求，且只有 50%的流量才能路由到新版本服务。

按照以下内容，修改步骤 2 创建的 Ingress。

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

kind: Ingress

metadata:
```

```
  name: gray-release-canary
  annotations:
    # 开启 Canary。
    nginx.ingress.kubernetes.io/canary: "true"
    # 仅允许 50%的流量会被路由到新版本服务 new-nginx 中。
    # 默认总值为 100。
    nginx.ingress.kubernetes.io/canary-weight: "50"
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # 新版本服务。
      - path: /
        backend:
          service:
            name: new-nginx
            port:
              number: 80
        pathType: ImplementationSpecific
```

执行以下命令，查看路由访问情况。

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

重复执行以上命令，可以看到仅 50%的流量路由到新版本服务。

步骤三：删除老版本服务基于 Helm 的发布管理

系统运行一段时间后，当新版本服务已经稳定并且符合预期后，需要下线老版本的服务 ，仅保留新版本服务在线上运行。为了达到该目标，需要将旧版本的 Service 指向新版本服务的 Deployment，并且删除旧版本的 Deployment 和新版本的 Service。

修改旧版本 Service，使其指向新版本服务。

```
apiVersion: v1
kind: Service
metadata:
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    # 指向新版本服务。
    run: new-nginx
  sessionAffinity: None
  type: NodePort
```

执行以下命令，请求头中满足 foo=bar 的客户端请求访问服务。

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

预期输出：

```
new
```

重复执行以上命令，可以看到请求全部被路由到了新版本的服务。

执行以下命令，删除 Canary Ingress 资源 gray-release-canary。

```
kubectl delete ingress gray-release-canary
```

删除旧版本的 Deployment 和新版本的 Service。

执行以下命令，删除旧版本的 Deployment。

```
kubectl delete deploy old-nginx
```

执行以下命令，删除新版本的 Service。

```
kubectl delete svc new-nginx
```

6.8 弹性伸缩

6.8.1 容器水平伸缩

云容器引擎 CCSE 支持在控制台界面上快速创建支持 HPA 的应用，实现容器资源的弹性伸缩。您也可通过定义 HPA（Horizontal Pod Autoscaling）的 YAML 来进行配置。

6.8.1.1 背景信息

从 v1.18 开始，K8s v2beta2 API 允许通过 HPA 的 behavior 字段配置扩缩行为。 在 behavior 字段中的 scaleUp 和 scaleDown 分别指定扩容和

缩容行为。当您在使用 HPA 时，希望只进行扩容或者只进行缩容的 Pod 伸缩，则可以通过开启指标伸缩，单击禁止缩容或者禁止扩容来实现。

默认值：均不禁止。

禁止扩容：selectPolicy 的值 Disabled 会关闭给定方向的扩容。因此使用以下策略，将会阻止扩容。

```
behavior:
  scaleUp:
    selectPolicy: Disabled
```

禁用缩容：selectPolicy 的值 Disabled 会关闭给定方向的缩容。因此使用以下策略，将会阻止缩容。

```
behavior:
  scaleDown:
    selectPolicy: Disabled
```

6.8.1.2 通过容器服务控制台创建 HPA 应用

天翼云云容器引擎已经集成了 HPA，您可以简便地通过容器服务控制台进行创建。您可以在创建应用的时候创建 HPA，也可以在已有应用的基础上开启 HPA。

方式一：在创建应用过程中，开启 HPA

- 登录云容器引擎 CCSE 管理控制台。
- 在控制台左侧导航栏中，单击集群。

- 在集群列表页面中，单击目标集群名称。
- 在集群管理页左侧导航栏中，选择工作负载 > 无状态。
- 点击左上角的新增按钮。
- 填写基本信息，在实例数量中选择自动伸缩，设置伸缩的条件和配置。
  - Resource 规则，支持 CPU 和内存，支持百分比和平均值等计值方式。
  - Pod 规则，支持 Pod 指标对象，支持平均值的计值方式，支持指定指标。
  - Object 规则，支持 Service 指标对象，支持阈值的计值方式，支持指定指标。
- 单击左下角的提交，一个支持 HPA 的 Deployment 就已经创建完毕。

结果验证

单击工作负载 > 无状态中单击应用名称，您可在部署的详情中查看伸缩组信息。

在实际使用环境中，应用会根据 CPU 负载进行伸缩。您也可在测试环境中验证弹性伸缩，通过给 Pod 进行 CPU 压测，可以发现 Pod 即可完成水平的扩展。

方式二：为已有应用开启 HPA

- 登录云容器引擎 CCSE 管理控制台。
- 在控制台左侧导航栏中，单击集群。
- 在集群列表页面中，单击目标集群名称。

- 在集群管理页左侧导航栏中，选择工作负载 > 无状态。
- 在应用列表页面中，选择目标应用，点击更多按钮。
- 设置伸缩的条件和配置。
  - Resource 规则，支持 CPU 和内存，支持百分比和平均值等计值方式。
  - Pod 规则，支持 Pod 指标对象，支持平均值的计值方式，支持指定指标。
  - Object 规则，支持 Service 指标对象，支持阈值的计值方式，支持指定指标。
- 单击确认，Deployment 的 HPA 设置完毕。

6.8.1.3 通过 kubectl 命令创建 HPA 应用

您也可通过 YAML 来手动创建 HPA，并将其绑定到要伸缩的 Deployment 对象上，通过 kubectl 命令实现容器自动伸缩配置。

下面针对一个 Nginx 应用进行举例。

- 创建并复制以下内容到 nginx.yml 中。

  Deployment 的编排模板如下。

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

```
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9 # replace it with your exactly <image_name:tags>
        ports:
        - containerPort: 80
        resources:
          requests:                    ##必须设置，不然 HPA 无法运行。
            cpu: 500m
```

- 执行以下命令，创建 Nginx 应用。

```
kubectl create -f nginx.yml
```

- 创建 HPA。

```
    apiVersion: autoscaling/v2beta2
    kind: HorizontalPodAutoscaler
    metadata:
      name: nginx-hpa
    spec:
      scaleTargetRef:
        apiVersion: apps/v1
        kind: Deployment
        name: nginx
      minReplicas: 1
      maxReplicas: 10
      metrics:
      - type: Resource
        resource:
          name: cpu
          target:
            type: Utilization
            averageUtilization: 50
```

- 执行以下命令，创建 Nginx HPA。

```
kubectl create -f nginx-pha.yml
```

- 创建好 HPA 后，再次执行 kubectl describe hpa <HPA 的名称>命令。

  可以看到以下信息，则表示 HPA 已经正常运行。

```
Normal SuccessfulRescale 30s horizontal-pod-autoscaler New size: 1; reason: All metrics below target
```

  此时当 Nginx 的 Pod 的利用率超过本例中设置的 50%利用率时，则会进行水平扩容，低于 50%的时候会进行缩容。