目录

角色授权

当用户开通金山云容器服务时，需要授权**KsyunKCEDefaultRolePolicy**系统默认的角色给金山云容器服务。当该角色被正确授予后，容器服务才能正常的调用相关的服务（SLB、EIP、EBS等），保证容器服务的正常运行。

授权流程

1. 登录容器服务控制台。
2. 若之前没有正确给容器服务账号授予默认的角色，系统会提示如下：

3. 点击**前往IAM控制台授权**，进行授权。
4. 授权完成，进入容器服务控制台页面，进行相应操作。

备注：

- 如您需要查看默认角色详细的策略信息，可以登录IAM控制台进行查看。
- 如需修改角色权限，请前往角色管理设置，需要注意的是，错误的配置可能导致金山云容器服务无法获取必要的权限，造成容器服务的不可用。

KsyunKCEDefaultRolePolicy角色的权限

- 负载均衡（SLB）权限
- 云硬盘（EBS）权限
- 弹性IP（EIP）权限
- 裸金属服务器权限
- 虚拟私有网络（VPC）权限
- 云监控（Monitor）相关权限

权限名称（action）	权限说明
Create*	注册监控相关信息
Delete*	删除监控相关信息
GetMetricStatistics	获取监控数据

集群概述

集群是指容器运行的所需要的云资源的组合，包含了若干的云服务器、负载均衡、专有网络等云资源。

集群配置

集群类型：容器服务根据Master平面部署方式，包括托管集群、独立部署集群两种类型。

- 托管集群：用户只需创建Worker节点，Master节点由容器服务创建并托管。用户无需管理Master节点，仅需承担Worker节点以及其他基础资源的费用。
- 独立部署集群：用户需自行创建Master节点及Worker节点。用户可以对集群基础设施进行更细粒度的控制，以及自行规划、维护服务器集群，需要承担Master节点、Worker节点以及其他基础资源的费用。

集群配置：用户可在创建集群时自定义设置集群的规模和配置，包含集群的节点个数，节点的机型，操作系统，数据盘大小等信息。

集群管理

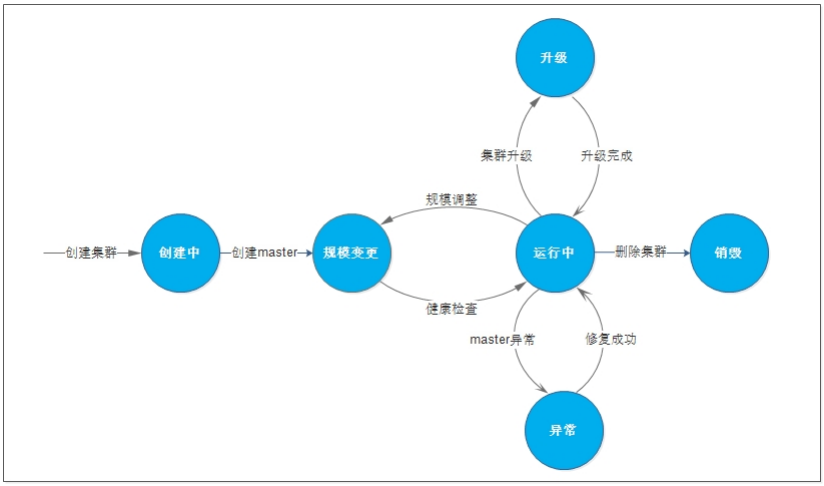
用户可通过容器服务控制台或者Open Api进行创建集群，扩缩节点，删除集群等操作，也支持使用Open Api直接操作集群。

集群生命周期

集群状态定义

状态	说明
创建中	集群正在申请相应的云资源
规模变更中	变更集群的节点数量
运行中	集群正常运行
升级中	集群升级服务端版本
异常	集群中出现异常情况，如master节点异常
删除中	集群正在删除

状态流转图



创建集群

集群作为容器运行的基础云资源，创建集群是使用容器服务的第一步，在创建集群过程中，用户可以自定义集群中云服务器的数量、操作系统、数据盘等配置信息。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 单击**新建集群**，进入创建集群流程。

4. 根据以下提示进行**集群配置**，配置完成后点击**下一步**。
   - 集群名称：用户自定义集群的名称，2-64个字符，支持中文，英文，数字，以及特殊字符-,.!$^*()%#&+/:;<=>[]_`{|}~
   - Kubernetes版本：目前金山云kubernetes版本为1.15.5、1.17.6和1.19.3，自定义选择。
   - Master管理模式：支持独立部署/托管模式，两种管理模式差别请参考集群类型。
   - 集群网络：选择集群所在的VPC网络。
   - 普通子网：当所选Master管理模式为托管模式时，需为托管的控制面节点配置所在子网。控制面节点会至少占用所选子网的三个IP。当选择不同可用区的多个子网时，各控制节点会优先选择不同可用区子网进行高可用部署。
   - 终端子网：终端子网用于创建私网负载均衡，用于集群内master节点和node节点通信。
   - Pod CIDR：为集群内的Pod分配此网络地址段的IP，客户自定义三个私有网络作为Pod网段。
   - Service CIDR：为集群内的Service分配此网络地址段的IP，客户自定义三个私有网络作为Service网段。
   - 网络模型：选择集群的网络模型，目前支持**Flannel**和**Canal**。
   - 集群描述：集群的描述，用户自定义填写。
5. 根据以下提示进行**节点配置**，配置完成后点击**下一步**。
   - 计费方式：支持包年包月、按量付费（按日月结）和按量付费的计费方式。
   - 节点类型：目前支持选择普通云主机和专属云主机创建集群。
   - Master&Etcd节点选择：
     - 可用区：选择节点部署的可用区。
     - 集群网络-子网：选择节点部署的VPC内的子网信息。
     - 机型：用户根据需求选择云服务器机型。
     - 镜像：目前金山云容器服务支持CentOS-7.3 64位、CentOS-7.6 64位、Debian-8.2 64位和Ubuntu-18.04 64位节点镜像。
     - 系统盘：用户根据需求选择系统盘大小。
     - 数据盘：自定义数据盘大小以及选择是否格式化并挂载至指定目录下。
     - 购买数量：目前master节点数量必须设置为3或5个，支持挎可用区部署。
   - Worker节点选择：配置信息同Master&Etcd。
   - 容器运行时：提供Docker和Containerd两种运行时，用户根据实际需求选择运行时，详情请见如何选择Containerd和Docker
   - 容器存储目录：按需选择是否自定义容器和镜像存储目录，建议存储到数据盘，如不设置，默认为/data/docker。
   - 安全组：定义集群中节点所属的安全组，关于容器集群安全组推荐设置，请参考容器集群安全组推荐设置。
   - 高级配置（可选）：按需配置以下信息：
     - 部署前执行脚本：指定自定义数据来配置Node，即当节点部署前执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/pre_userscript路径查看。
     - 部署后执行脚本：指定自定义数据来配置Node，即当节点部署后执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/userscript路径查看。
     - 封锁：根据需求勾选是否封锁worker节点，封锁worker节点后，将不接受新的Pod调度到该节点，若需重新调度，则要手动取消封锁的节点。更多操作请见详情。
     - Label：根据需求进行Label自定义设置。
6. 根据以下提示进行**设置基本信息**，配置完成后点击**下一步**。
   - 所属项目：自定义购买的云资源的所属项目。
   - 服务器名称：自定义服务器名称，2-64个字符，支持中文，英文，数字，以及特殊字符-,.!$^*()%#&+/:;<=>[]_`{|}~
   - 登录方式：用户自定义选择登录方式，支持设置密码和密钥登录。
7. 点击**创建**，执行集群创建的操作，在集群列表页中可查询创建的集群。

集群基本信息查询

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 点击集群名称，进入该集群操作页面。
4. 点击**基本信息**，可查看该集群的基本信息：
   - 基本信息：包括集群名称，集群ID，集群运行区域，节点数量，集群网络，Pod CIDR，Service CIDR，创建时间，最后更新时间，集群描述等信息，其中，集群名称和集群描述支持编辑，点击对应的按钮，即可进行编辑。
   - 组件：可选择是否开启NFSClient。

删除集群

若客户暂时不需要使用容器服务，可以直接将集群删除。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要删除的集群，点击**更多>删除集群**。
4. 在出现的弹窗中，可按需选择是否删除集群内非包年包月云服务器实例，点击**确认**即可删除集群。

备注： 1.包年包月的云服务器实例和裸金属服务器仅作移出集群操作，不会被删除 2.删除集群会同步删除Apiserver相关网络等资源，数据不可恢复，请谨慎操作！

通过kubectl连接Kubernetes集群

如果用户需要从本地个人计算机连接到金山云的 Kubernetes 集群，请使用 Kubernetes 命令行客户端kubectl。

安装kubectl工具

详细安装过程参考安装并配置 kubectl。

获取集群凭证

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 点击集群ID，进入该集群操作页面。
4. 点击**基本信息** 进入集群基本信息页后，点击**获取集群config**，如下图所示：
5. 在弹窗中，即可获取集群凭证信息。

配置 Kubeconfig

下载集群的配置文件，复制到本地计算机的 $HOME/.kube/config（kubectl的默认路径）。

注：若您之前配置过KUBECONFIG环境变量，kubectl会优先加载KUBECONFIG环境变量，而不是$HOME/.kube/config，使用时请注意。

访问 Kubernetes 集群

配置完成后，您可以使用本地计算机通过kubectl访问Kubernetes集群，如：

```
kubectl get node
```

容器服务推荐安全组设置

金山云容器服务选择金山云私有网络VPC作为容器的底层网络，本文主要介绍使用金山云容器服务安全组的使用原则，帮助大家选择对应的安全组策略。

安全组

安全组是一种有状态的服务器虚拟防火墙，它用于设置单台或多台云服务器的网络访问控制，是金山云提供的重要的网络安全隔离手段。更多关于安全组的信息，请参考安全组。

使用容器服务选择安全组的建议

1. 容器集群中，不同的服务实例分布部署在集群中的不同节点，根据资源的使用情况不同服务的pod还会出现实例的迁移，建议同一集群下的云主机绑定同一个安全组规则，集群的安全组不添加其他的云主机。
2. 若您需要基于Service的NodePort的模式进行服务转发（即访问Node IP：NodePort），其中Node Port创建Service时集群自动分配或者手动指定的，Node Port的范围是30000-32768，请根据自己的需要放行节点 30000 ~ 32768 端口。
3. 需要 SSH 登录节点的放通 22 端口。

推荐安全组设置

入站规则

协议	行为	起始端口	结束端口	源IP	备注
TCP	允许	30000	32768	0.0.0.0/0	放通所有IP对30000-32768端口的TCP访问
UDP	允许	30000	32768	0.0.0.0/0	放通所有IP对30000-32768端口的UDP访问
TCP	允许	22	22	0.0.0.0/0	放通所有IP对22端口的访问

出站规则

协议	行为	起始端口	结束端口	源IP	备注
IP	允许	-	-	0.0.0.0/0	放行所有出VPC流量

集群网络地址段划分

在金山云创建kubernetes集群时，需要用户自主规划集群VPC网络、pod网络和service网络。本文将详细介绍金山云VPC环境下kubernetes里各个网络的作用，以及地址段将如何划分。

Kubernetes集群网段解释

集群网络：选择集群所在VPC的网络。为集群内主机分配在节点网络地址范围内的 IP 地址，您可以选择私有网络中的子网用于集群的节点网络，更多私有网络的介绍请参考虚拟网络和子网。

Pod网络：将为kubernetes集群的pod分配此网络地址段内的IP，您可以自定义三个私有网段作为pod的网段，金山云容器服务将自动为集群内的每台云主机分配一个24位的网段用于该主机分配Pod的IP地址。

Service网络：将为Service分配此网络地址段的IP。Service地址仅能在集群内使用，不能在集群外使用。

集群网络、Pod网络、Service网络的关系

1. 在一个集群中，集群网络、Pod网络和Service网络地址段不能冲突。
2. 同一个VPC内，不同集群的Pod网络地址段不能冲突，Service地址段可以重叠。

如何划分地址段

下面将列举几个典型场景，帮助大家更深刻的理解Kubernetes集群网络地址段的划分：

单VPC-单Kubernetes集群场景

集群网络在创建VPC的时候就已经确定，在创建Kubernetes集群时，只要设定Pod和Service网络地址段与VPC不冲突即可。

单VPC-多Kubernetes集群场景

在一个VPC下可以创建多个Kubernetes集群。基于金山云容器服务网络模型（容器服务会自动在VPC路由上配置每个Pod地址的路由，Pod与Pod，Pod与Node之间的通信利用VPC的路由转发），故一个VPC内的多个集群，Pod地址段不能冲突，Service地址段允许重复。

跨VPC互联

两个VPC互联的情况下，需要使用对等连接。举例如下：VPC1（10.0.0.0/16）和VPC2（172.16.0.0/16）通过对等连接建立了跨VPC互联，在VPC1内创建Kubernetes集群，首先该集群的Pod网络地址段不能和VPC1的网段冲突，同时其地址段也不能和VPC2的地址段冲突。同理，在VPC2创建Kubernetes集群也类似。

容器集群支持GPU调度

若您的业务有运行机器学习、图像处理的等高运算密度的场景，您可以通过金山云容器集群+GPU快速开始使用GPU容器，无需手动安装nvidia driver和CUDA。

容器服务控制台操作说明

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 点击**新建集群**，进入创建集群页面。
4. 在**节点配置**流程中，在选择云服务器机型时选择对应GPU机型，点击确认。

kubectl命令操作说明

不同于CPU和内存，您需要在yaml文件中显式申明你打算使用的GPU的数量，通过在container的resources.limits中设置nvidia.com/gpu，申明你想要使用的GPU数量。

示例：

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vector-add
spec:
  restartPolicy: OnFailure
  containers:
    - name: cuda-vector-add
      image: hub.kce.ksyun.com/ksyun/cuda-vector-add:0.1
      resources:
        limits:
          nvidia.com/gpu: 1   # 指定调用nvida gpu的数量
```

使用限制

1. 仅在创建时间为2018年12年27日之后的集群支持GPU调度，若您的集群创建时间早于上述时间且需要使用GPU容器，请提交工单申请。
2. GPU资源申请仅需要在容器资源的 limits 中配置。
3. 容器之间不支持共享GPU，每个容器可以申请一个或者多个GPU。
4. GPU必须以整数为单位被申请使用。

如何选择K8s集群的网络模型：Flannel和Canal

金山云容器服务提供了两种不同的集群网络模型：Flannel和Canal，本文详细描述了这两种网络模型，为您在创建集群选择网络模型提供参考。

Flannel：使用简单稳定的社区的Flannel CNI插件，搭配金山云VPC网络主机路由能力，提供高性能和稳定的集群网络体验，但是不支持设置集群内的隔离策略，例如：不支持基于Kubernetes标准的网络策略（Network Policy）。

Canal：flannel 和calico 中的 felix 组件结合而成，完全兼容flannel的功能。同时支持基于Kubernetes标准的NetworkPolicy来定义容器间的访问策略。

对于不需要使用Network Policy的用户，可以选择Flannel，其他情况建议选择Canal。

集群资源预留

节点作为集群的一部分需要运行必需的Kubernetes和KCE的系统组件。因此您的节点资源总量（capacity）与节点所在的KCE中的可分配资源数量（allocatable）之间存在差异。

为了保证节点的稳定性，KCE集群节点上会根据节点的规格预留一部分资源给Kubernetes的相关组件（kubelet,kube-proxy以及docker等）。

用户集群节点资源预留的计算规则为：

Allocatable = Capacity - Reserved - Eviction Threshold

注： Allocatable ：节点上可分配给pod的资源。 Capacity ：Node的硬件资源总量。 Reserved ：节点上预留给系统组件的资源。 Eviction-threshold：节点的驱逐阈值。

KCE对节点内存的预留规则

1. total_mem <= 4GB，reserved_value= total_mem*25%
2. 4GB < total_mem <= 8GB，reserved_value= 4GB*25% + (total_mem - 4GB)*20%
3. 8GB < total_mem < =16GB，reserved_value= 4GB*25% + 4GB*20% + (total_mem - 8GB)*10%
4. 16GB < total_mem < =128GB，reserved_value= 4GB*25% + 4GB*20% + 8GB*10% + (total_mem - 16GB)*6%
5. total_mem > 128GB，eserved_value= 4GB*25% + 4GB*20% + 8GB*10% + 112GB*6% +(total_mem - 128GB)*2%

注： “total_mem”为内存总量量，“reserved_value”为预留值。

KCE对每个节点预留了额外的100Mi给kubelet驱逐所用

常见配置的资源预留列表如下：

CPU

节点总量/单位：核	2	4	8	16	32
节点预留量/单位：核	0.07	0.08	0.09	0.11	0.15

Memory

节点总量/单位：Gib	4	8	16	32	64
节点预留量/单位：Mib	1024	1843	2662	3645	5611

集群弹性伸缩

Cluster AutoScaler （简称CA）是一个自动扩展和收缩 Kubernetes 集群 Node 的扩展。当集群容量不足时，它会自动去 Cloud Provider 创建新的 Node，而在 Node 长时间（超过 10 分钟）资源利用率很低时（低于 50%）自动将其删除以节省开支。

扩容条件

Cluster AutoScaler 定期（默认间隔 10s）检测是否有充足的资源来调度新创建的 Pod，当资源不足时会调用 Cloud Provider 创建新的 Node。 每当kubernetes调度程序找不到一个运行pod的地方时，它会将pod的PodCondition设置为false，并将原因设置为“unschedulable ”。集群自动扩缩容程序正是每隔一段时间扫描一次是否有不可调度的pod来进行扩容的，如果有就尝试扩容节点来运行这些pod。

扩容策略

当集群中有多个节点池时，可以通过选项配置选择节点池的策略，支持如下三种方式：

- random：随机选择节点池进行扩容。
- most-pods：选择容量最大（能调度更多pod）的节点池进行扩容。
- least-waste：以最小浪费原则选择，选择调度pod后可用资源剩余更少的节点池进行扩容。

缩容条件

Cluster AutoScaler 也会定期（默认间隔 10s）自动监测 Node 的资源使用情况，当一个 Node 长时间（超过 10 分钟其期间没有执行任何扩展操作）资源利用率都很低时（默认值低于 50%）并且这个node上的pod可以被调度到别的节点上，那么CA自动将其从集群中删除。此时，原来的 Pod 会自动调度到其他 Node 上面。

当 Node 上面的 Pods 满足下面的条件之一时，Node 不会缩容：

- Pod 配置了 PodDisruptionBudget (PDB)，当Pod不满足PDB时，不会缩容。
- Node 上运行了 kube-system命名空间下非 DaemonSet 管理的 的pod。
- Pod 不是通过 deployment, replicaset, job, statefulset 等控制器创建的。
- Pod 使用了本地存储。
- 其他原因导致的 Pod 无法重新调度，如亲和、反亲和等，无法调度到其他节点。

注意事项

- Cluster AutoScaler 与基于监控指标的弹性伸缩的节点扩缩容（Auto Scaling，简称AS）相冲突，请不要为容器集群的节点池设置基于监控指标的自动扩缩。
- 请指定 Pod 的 request 值：自动扩容的触发条件是集群中存在由于资源不足而无法调度的 Pod，而判断资源是否充足正是基于 Pod 的 request 来进行的。
- 不要直接修改属于弹性伸缩节点池内的节点，确保弹性伸缩节点池中的节点具有相同的配置。
- 在集群进行缩容的过程中，服务可能会遇到一些中断。例如，如果服务包含具有单个副本的控制器，则在删除该副本的 Pod 的当前节点时，此 Pod 可能会在其他节点上重启。在启用自动缩容之前，请确保服务可以容忍潜在的中断。建议您使用PodDisruptionBudgets 阻止 Pod 在缩容时被删除。

集群节点配置推荐

集群规划

目前在创建Kubernetes集群时，存在着使用很多小规格节点的现象，这样有以下弊端：

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

使用大规格节点的优势：

- 网络带宽大，对于大带宽类的应用，资源利用率高。
- 容器在一台节点内建立通信的比例增大，减少网络传输。
- 拉取镜像的效率更高。因为镜像只需要拉取一次就可以被多个容器使用。而对于小规格的节点拉取镜像的次数就会增多，影响容器的启动速度。

Master配置推荐

Master规格跟集群规模有关，集群规模越大，所需要的Master规格也越高，不同集群规模的，Master节点配置推荐如下：

集群规模（节点数量）	master规格
0-100	>=4C8G,SSD数据盘>=50G
100-300	>=8C16G,SSD数据盘>=50G
300-500	>=16C32G,SSD数据盘>=100G
500-1000	>=32C64G,SSD数据盘>=100G
1000节点以上	请联系我们

Worker规格选型

- 为了保证节点的稳定性，金山云容器集群节点上会根据节点的规格预留一部分资源给Kubernetes的相关组件（kubelet,kube-proxy以及docker等），详见集群资源预留，建议结合资源预留和业务的资源申请，选择合适的节点配置。

- 根据业务类型确认CPU：Memory比例 对于使用内存比较多的应用例如java类应用，建议考虑使用1:8的机型；对于CPU密集型的业务，可以申请1：2的机器；如果是不同业务的混合部署，建议给不同机型或者配置的节点打上标签，配合nodeAffinity调度pod。

集群master节点管理模式说明

目前金山云提供两种模式的Kubernetes集群

- Master托管模式
- Master独立部署模式

Master托管模式

该模式下，集群中的Master和Etcd由金山云容器团队集中管理和运维。用户只需创建 Worker 节点，无需运维集群的Master节点，具备简单、低成本、高可用的特点

注意事项：

1. Master托管模式下，用户暂时无需为集群的Master、Etcd资源付费，但是需要为集群中的Worker节点、持久化存储以及网络资源付费
2. Master托管模式下，用户无法自主修改Master、Etcd的规格和服务参数。如果用户有自定义修改的需求，请使用Master独立部署的模式

Master独立部署模式

金山云容器服务为用户提供了Master独立部署模式。

用户的Master、Etcd会部署在用户购买的云服务器或者裸金属服务器上，用户拥有Kubernetes集群的全部权限。

使用自定义镜像

本文主要介绍如何在金山云容器服务使用自定义镜像创建Kubernetes集群。

注意事项

- 使用自定义的镜像，请在金山云容器团队提供的标准镜像上制作自定义镜像。
- 目前仅支持同类型的操作系统镜像的制作。例如，使用 CentOS 基础镜像制作 CentOS 类的自定义镜像。

操作步骤

获取金山云容器服务提供的标准镜像

为了提升镜像的稳定性，并提供更多特性，金山云容器团队为您提供容器服务的标准镜像 您可以通过调用容器服务的获取容器服务支持的节点操作系统接口，获取容器服务提供的标准镜像。

这里列举了华北1（北京）标准镜像的相关信息

镜像ID	操作系统名称	OS类型	是否支持GPU
8c8456b6-2c79-4ac4-86b2-21fd2fbbc9de	Ubuntu-18.04 64位	Ubuntu	否
4b293997-9940-42bd-a5f0-65ecb788f394	CentOS-7.6 64位	CentOS	否
6f124026-02ca-403a-8f4c-be8e60726268	Ubuntu-18.04 CUDA10.2	Ubuntu	是
3c04b700-7e3d-4e06-95c1-d789ef223c4a	CentOS-7.5 CUDA10.2	CentOS	是

备注

- 我们将持续对标准镜像进行优化或者 bugfix，并生成新版本标准镜像。建议您通过调用获取容器服务支持的节点操作系统接口获取最新版本的标准镜像。
- 新版本标准镜像不会对您之前使用旧版本标准镜像制作的自定义镜像产生任何影响。为了达到更好的使用效果，建议您使用新版本标准镜像。

使用容器团队提供的标准镜像创建云服务器

调用云主机的 RunInstances 创建云服务器实例,其中ImageId请替换为容器服务的标准镜像，获取方式详见获取容器服务支持的节点操作系统。

制作自定义镜像

用户根据业务的实际制作自定义镜像，参考制作镜像完成创建。

使用自定义镜像

自定义镜像制作完成后，请联系您的商务同学，将镜像导入到容器服务。导入成功后，即可在容器服务控制台使用用户自定义的镜像。

注意事项

- 请基于容器服务提供的标准镜像制作自定义镜像。
- 请勿删除镜像中/usr/libexec/appctl和/lib/systemd/system/appctl.service 文件。

概述

多集群管理旨在帮助用户实现容器服务在多云、混合云场景下的应用，具体场景包括业务备份和容灾，高可用部署，开发与生产环境隔离，单一厂商绑定规避，以及跨集群计算资源调度，跨区域服务就近访问等。集群联邦（Kubernetes Federation V2, KubeFed）支持通过Host集群中定义的一套API统一编排多个Member集群中的配置，以联邦形态实现对多个Kubernetes集群的统一管理。

金山云容器服务多集群管理支持丰富的集群类型与来源，在将第三方集群（包含来源于用户自建或其他公有云厂商的Kubernetes集群）成功纳管之后，即可支持选取多个托管集群、独立部署集群、第三方集群建立集群联邦。同时在北京、上海、广州三个地域支持联邦管理，满足用户多地域部署的业务需求。

基本概念

集群联邦（Federation）：多个集群组成的集合，包含一个Host集群和多个Member集群。Host集群和Member集群间将通过公网通道实现通信，Member集群的网络可以完全隔离。

Host集群：提供KubeFed API与控制平面的集群，同时也可以作为Member集群部署联邦化资源。

Member集群：通过KubeFed API注册加入联邦，接受Host集群的管理，联邦化资源部署后会自动在Member集群中同步。

纳管集群

您可以通过容器服务控制台纳管第三方的Kubernetes集群，并通过容器控制台对第三方集群的Kubernetes资源进行管理。

前提条件

- 被纳管的集群需要具备公网访问的能力。
- 对Kubernetes集群属性的管理仍然必须在原有的集群中进行，包括添加与删除节点、升级Kubernetes集群版本以及更改Kubernetes组件参数等。

创建纳管集群

新建纳管集群

1. 登录容器服务控制台。

2. 点击**新建纳管集群**，进入纳管集群新建页。
3. 设置基本信息
   - 集群名称：用户自定义纳管集群的名称，2-64个字符，支持中文，英文，数字，以及特殊字符-,.!$^*()%#&+/:;<=>[]_`{|}~。
   - 集群描述：最大支持256字符。
   - 数据中心：当前一期仅开放北京地区的集群纳管。
4. 基本信息设置完成后点击**创建**，进入集群导入页面。

集群导入

1. 按照页面提示，在目标集群内执行相应命令导入并执行配置文件，相应资源在目标集群中成功创建后，即可完成目标集群的导入。
2. 导入成功后，您可以在容器服务控制台的集群列表页面，看到目标纳管集群的运行状态为“已接入”。

若未能成功导入，目标集群状态为“等待接入”。您可以通过点击集群名称进入集群基本信息页，点击**获取集群导入配置文件**，再次在目标集群中进行配置文件的导入与执行。

管理纳管集群

查看集群状态与基本信息

集群成功纳管至金山云控制台后，您可以在控制台进行对金山云集群和第三方集群的统一管理，支持查看集群状态和基本信息，同时支持k8s原生资源相关操作（具体操作可参照相关文档）。

- 注意：目前暂不支持第三方集群直接调用金山云底层资源（如负载均衡器、云硬盘），如需实现相关功能需自行搭建相关组件。

删除纳管集群

进入容器服务控制台集群列表页，点击**删除集群**，即可取消金山云控制台对目标集群的纳管，仅通过集群导入配置文件创建的资源会被删除，集群内其他资源不受影响。

联邦管理

创建集群联邦

前提条件

创建集群联邦需满足以下条件：

- 目标Host集群创建/纳管于金山云控制台。
- 目标Host集群Kubernetes版本1.13+。
- 目标Host集群内预留空间大于2核CPU，4G内存。
- 若Member集群与Host集群不在同一地域，Member集群加入联邦前需开启跨地域访问入口。

新建集群联邦

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**多集群 > 联邦管理**，进入联邦管理页面。
3. 选择目标Host集群所在地域，点击**新建集群联邦**。
4. 选择地域与目标Host集群，点击**创建**，即开始联邦控制面的部署。您可以通过部署状态页查看控制面组件状态，出现异常状态时点击**重新部署**，将会对异常组件进行覆盖重装。

5. 部署成功后，返回联邦列表页，可查看Host集群联邦状态。

备注：“未加入”表示控制面状态正常，Host集群仅作为控制面部署集群但不具备Member角色，因此不会同步部署联邦化资源；若需要Host集群同时具备Member角色，可选择**更多 - 加入联邦**。

新增Member集群

1. 完成Host集群部署后，点击**新增Member集群**进入添加Member集群的流程，可选择不同区域下的多个集群加入联邦（一次最多添加五个Member集群）。

2. 若所选集群中存在与Host集群不同地域的集群，需勾选相应集群，手动开启跨地域访问通道。
3. 点击**确定**，将为与Host集群同地域以及符合跨地域访问要求的集群执行加入联邦操作，您可返回联邦列表查看Member集群联邦状态。

管理集群联邦

退出联邦

若想要退出Member集群，可执行**退出联邦**操作。注意退出联邦将删除集群中联邦相关资源，若此集群同时属于其他联邦，相同命名空间下资源会受影响，因此不建议将同一集群同时纳入多个联邦。

若退出状态异常，可勾选强制退出选项，将跳过异常资源执行强制退出操作。

解除联邦

执行**更多 > 解除联邦**操作，将删除Host与Member集群中所有联邦相关资源。 

多集群联邦负载均衡实践

概述

Istio 实现联邦集群负载均衡

Istio 支持多集群部署，并能够实现跨集群服务发现和跨集群负载均衡。Istio 多集群的部署方案一般有两种，分别是多副本控制平面部署和共享控制平面部署。而共享控制平面部署又分为单一网络和多网络部署。在多集群联邦环境中集群常分布在不同可用区下，因此集群不会处于同一网络，则共享控制平面的多网络部署模式符合多集群联邦场景下的负载均衡场景。

Istio 多网络共享控制平面方案

使用共享控制平面方案部署，需要有一个主集群用来部署 control plane，其他远程集群要加入 istio 网格的集群则需要连接主集群的 control plane。在Istio 1.7的部署方案中，控制平面将会以istiod形态部署，远程集群中部署的istiod用于CA和集群中工作负载的webhook注入，服务发现则会导向主集群中的控制面实现。集群间通信通过gateway实现，不同集群的工作负载之间既不要求 VPN 连接也不要求直接网络访问。

目标集群配置

前提条件

- Kubernetes 集群版本要求1.16及以上。
- 创建联邦集群要求至少两个 Kubernetes 集群，多个集群可以属于不同VPC。
- 若Member/remote集群与Host集群不属于同一VPC，需为Member/remote集群开启Api Server公网访问。
- Host集群worker节点预留资源≥4核8G。

以下以两个处于不同VPC环境的集群为例，基于集群联邦实现Istio多集群网格部署以及跨集群负载均衡。

准备工作

创建两个集群

登录容器服务控制台，分别在华北1（北京）和华东1（上海）创建两个Kubernetes 集群。

创建联邦集群

在容器控制台进入多集群-联邦管理，联邦地域为目标Host集群所在地域，点击**新建集群联邦**。

选择地域与目标Host集群，点击**创建**，即开始联邦控制面的部署。

查看部署状态，当全部组件部署成功后，此时Host集群处于"未加入"状态，点击**更多 > 加入联邦**将集群加入到联邦集群中。

接着添加另一个Kubernetes 集群，点击**新增Member集群**。

为了演示我们选择与Host集群不同的区域下的集群，这也是使用联邦集群实现跨地域容灾常用部署方式。

添加完成后，联邦集群页面中已经可以看到有两个Kubernetes集群。

查看联邦集群状态

进入联邦Host集群节点，获取当前联邦的集群信息。

```
$ kubectl get kubefedclusters -n kube-federation-system
NAME                                 AGE     READY
d161c69e-286b-4541-901f-b121c9517f4e   5m      True
fcb0f8d3-9907-4a4a-9653-4584b367ee29   12m     True
```

准备联邦证书及kubefedctl

在安装好联邦集群后，联邦证书会默认保存在Host集群kube-system namespaces下的kubefedconfig的configMap中。将集群证书保存到本地~/.kube/config中，便于kubectl及kubefedctl读取使用。

```
$ kubectl get cm -n kube-system kubefedconfig -o yaml
```

验证证书是否正确配置，并将当前context指向Host集群。

```
$ kubectl config get-contexts
CURRENT   NAME                                          CLUSTER                                       AUTHINFO                                      NAMESPACE
          context-d161c69e-286b-4541-901f-b121c9517f4e   cluster-d161c69e-286b-4541-901f-b121c9517f4e   admin-d161c69e-286b-4541-901f-b121c9517f4e
          context-fcb0f8d3-9907-4a4a-9653-4584b367ee29   cluster-fcb0f8d3-9907-4a4a-9653-4584b367ee29   admin-fcb0f8d3-9907-4a4a-9653-4584b367ee29

$ kubectl config use-context context-fcb0f8d3-9907-4a4a-9653-4584b367ee29
Switched to context "context-fcb0f8d3-9907-4a4a-9653-4584b367ee29".
```

下载kubefedctl命令行工具。

```
$ wget https://github.com/kubernetes-sigs/kubefed/releases/download/v0.4.1/kubefedctl-0.4.1-linux-amd64.tgz
$ tar zxvf kubefedctl-0.4.1-linux-amd64.tgz
$ rm kubefedctl /usr/local/bin/
```

Istio多集群网格部署

部署示例用到了istioctl命令和istio提供的配置文件，下载并安装。

```
$ curl -L https://istio.io/downloadIstio | sh -
$ cd istio-1.7.2 && export PATH=$PWD/bin:$PATH
```

在本配置中，安装 Istio 时同时开启控制平面和应用pods的双向TLS。对于共享的根CA使用Istio示例目录下相同的Istio证书，在host和member集群中都创建相同的namespace和secret保存根证书。

```
# 创建 Namespace/istio-system 和 FederatedNamespace/istio-system
$ kubectl create namespace istio-system
$ kubefedctl federate ns istio-system

# 创建secret 和 FederatedSecret
$ kubectl create secret generic cacerts -n istio-system \
    --from-file=samples/certs/ca-cert.pem \
    --from-file=samples/certs/ca-key.pem \
    --from-file=samples/certs/root-cert.pem \
    --from-file=samples/certs/cert-chain.pem

$ kubefedctl federate secret cacerts -n istio-system
```

安装主集群

在联邦Host集群部署istio控制面板服务，在部署istio控制面服务时会使用以下环境变量用于替换其中配置模板中定义的变量并生成配置文件。

```
# 设置环境变量
$ export MAIN_CLUSTER_CTX=context-fcb0f8d3-9907-4a4a-9653-4584b367ee29
$ export REMOTE_CLUSTER_CTX=context-d161c69e-286b-4541-901f-b121c9517f4e

$ export MAIN_CLUSTER_NAME=cluster-fcb0f8d3-9907-4a4a-9653-4584b367ee29
$ export REMOTE_CLUSTER_NAME=cluster-d161c69e-286b-4541-901f-b121c9517f4e

$ export MAIN_CLUSTER_NETWORK=network1
$ export REMOTE_CLUSTER_NETWORK=network2

# 创建主集群配置文件
cat <<EOF> istio-main-cluster.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
```

```
spec:
  values:
    global:
      multiCluster:
        clusterName: ${MAIN_CLUSTER_NAME}
      network: ${MAIN_CLUSTER_NETWORK}

      # Mesh network configuration. This is optional and may be omitted if
      # all clusters are on the same network.
      meshNetworks:
        ${MAIN_CLUSTER_NETWORK}:
          endpoints:
          - fromRegistry:  ${MAIN_CLUSTER_NAME}
          gateways:
          - registry_service_name: istio-ingressgateway.istio-system.svc.cluster.local
            port: 443

        ${REMOTE_CLUSTER_NETWORK}:
          endpoints:
          - fromRegistry: ${REMOTE_CLUSTER_NAME}
          gateways:
          - registry_service_name: istio-ingressgateway.istio-system.svc.cluster.local
            port: 443

      # Use the existing istio-ingressgateway.
      meshExpansion:
        enabled: true
EOF

# 部署控制面板
$ istioctl install -f istio-main-cluster.yaml --context=${MAIN_CLUSTER_CTX}
Detected that your cluster does not support third party JWT authentication. Falling back to less secure first party JWT. See https://istio.io/docs/ops/best-practices/security/#configure-third-party-service-accou
nt-tokens for details.
   Istio core installed
   Istiod installed
   Ingress gateways installed
   Installation complete

$ kubectl get pod -n istio-system --context=${MAIN_CLUSTER_CTX}
NAME                                    READY   STATUS    RESTARTS   AGE
istio-ingressgateway-6bdbbc5566-c9kxk   1/1     Running   0          26s
istiod-689b5cbd7d-2dsml                 1/1     Running   0          37s

# 设置环境变量`ISTIOD_REMOTE_EP`
$ export ISTIOD_REMOTE_EP=$(kubectl get svc -n istio-system --context=${MAIN_CLUSTER_CTX} istio-ingressgateway -o jsonpath='{.status.loadBalancer.ingress[0].ip}')
$ echo "ISTIOD_REMOTE_EP is ${ISTIOD_REMOTE_EP}"
```

远程集群安装

```
# 创建远程集群配置文件
cat <<EOF> istio-remote0-cluster.yaml
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
spec:
  values:
    global:
      # The remote cluster's name and network name must match the values specified in the
      # mesh network configuration of the primary cluster.
      multiCluster:
        clusterName: ${REMOTE_CLUSTER_NAME}
      network: ${REMOTE_CLUSTER_NETWORK}

      # Replace ISTIOD_REMOTE_EP with the the value of ISTIOD_REMOTE_EP set earlier.
      remotePilotAddress: ${ISTIOD_REMOTE_EP}

  ## The istio-ingressgateway is not required in the remote cluster if both clusters are on
  ## the same network. To disable the istio-ingressgateway component, uncomment the lines below.

  components:
   ingressGateways:
   - name: istio-ingressgateway
     enabled: true
EOF

$ istioctl install -f istio-remote0-cluster.yaml --context=${REMOTE_CLUSTER_CTX}
$ kubectl get pod -n istio-system
```

跨集群负载均衡设置

配置 ingress 网关

在联邦集群多网络共享控制平面场景下，使用Istio ingress gateway作为流量入口，实现跨网络间通信。为了提高网络通信安全性，需要配置ingress gateway使用443端口和SNI header。

```
cat <<EOF> cluster-aware-gateway.yaml
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: cluster-aware-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 443
      name: tls
      protocol: TLS
    tls:
      mode: AUTO_PASSTHROUGH
    hosts:
    - "*.local"
EOF
```

在Host和Member集群中创建gateway，部署gateway用于实现跨集群服务路由。

```
$ kubectl apply -f cluster-aware-gateway.yaml --context=${MAIN_CLUSTER_CTX}
$ kubectl apply -f cluster-aware-gateway.yaml --context=${REMOTE_CLUSTER_CTX}
```

配置跨集群服务注册

为了实现联邦集群跨集群负载均衡，部署联邦的Host集群上的控制面必须要能够访问到全部集群的kube-apiserver服务，以实现服务发现，获取endpoints和pod属性等。 要访问member集群需要配置member集群的kube-apiserver公网访问证书。获取金山云kubernetes集群公网，容器服务控制台中选择需要获取证书的集群，进入集群基本信息页面，点击**获取集群config**，选择**公网访问config**。

将获取到的remote集群的公网访问证书添加到.kube/config文件中，并执行以下操作。Istio将使用该公网证书在remote集群创建serviceaccount `io-reader-service-account` 和相关role、rolebinding，并在host集群中创建secret保存io-reader-service-account的证书信息。

```
$ istioctl x create-remote-secret --name ${REMOTE_CLUSTER_NAME} --context=${REMOTE_CLUSTER_CTX} | \
    kubectl apply -f - --context=${MAIN_CLUSTER_CTX}
```

跨集群负载均衡功能测试

在Host集群和Member集群都部署一个 helloworld 服务，在主集群部署 v1 版本服务，在从集群部署 v2版本服务。部署完成后，在任何一个集群访问该服务，流量会在多集群服务间路由。

部署测试服务

创建 sample namespace 并设置 istio-injection=enabled 标签。

```
$ kubectl create namespace sample --context=${MAIN_CLUSTER_CTX}
$ kubectl label namespace sample istio-injection=enabled --context=${MAIN_CLUSTER_CTX}
```

创建 federatenamespace。

```
$ kubefedctl federate ns sample
```

创建 federatedeployment 部署在两个集群中的helloworld服务并使用不同的镜像版本，分别为v1和v2。helloworld-deploy.yaml：

```
apiVersion: types.kubefed.io/v1beta1
kind: FederatedDeployment
metadata:
  name: helloworld
  namespace: sample
spec:
  template:
    metadata:
      labels:
        app: helloworld
        version: v1
    spec:
      replicas: 1
      selector:
        matchLabels:
          app: helloworld
          version: v1
      template:
        metadata:
          labels:
            app: helloworld
            version: v1
        spec:
          containers:
            - image: docker.io/istio/examples-helloworld-v1
              name: helloworld
  placement:
    clusters:
      - name: fcb0f8d3-9907-4a4a-9653-4584b367ee29
      - name: d161c69e-286b-4541-901f-b121c9517f4e
  overrides:
    - clusterName: d161c69e-286b-4541-901f-b121c9517f4e
      clusterOverrides:
        - path: "/spec/template/spec/containers/0/image"
          value: "docker.io/istio/examples-helloworld-v2"
        - path: "/spec/template/metadata/labels/version"
          value: "v2"
        - path: "/spec/selector/matchLabels/version"
          value: "v2"
        - path: "/metadata/labels/version"
          value: "v2"
```

helloworld-svc.yaml：

```
apiVersion: v1
kind: Service
metadata:
  name: helloworld
  labels:
    app: helloworld
spec:
  ports:
  - port: 5000
    name: http
  selector:
    app: helloworld
```

在Host和Member中部署helloworld服务：

```
kubectl apply -f helloworld-svc.yaml -n sample --context=${MAIN_CLUSTER_CTX}
kubectl apply -f helloworld-svc.yaml -n sample --context=${REMOTE_CLUSTER_CTX}
```

验证

为了演示访问 helloworld 服务的流量是分布到多集群的，需要部署一个示例提供的 sleep 服务来调用 helloworld 服务。

```
# 在Host和member集群分别部署sleep服务
$ kubectl apply -f samples/sleep/sleep.yaml -n sample --context=${MAIN_CLUSTER_CTX}
$ kubectl apply -f samples/sleep/sleep.yaml -n sample --context=${REMOTE_CLUSTER_CTX}

# 在Host集群中多次调用helloworld.sample服务
$ kubectl exec -it -n sample -c sleep --context=${MAIN_CLUSTER_CTX} $(kubectl get pod -n sample -l app=sleep --context=${MAIN_CLUSTER_CTX} -o jsonpath='{.items[0].metadata.name}') -- curl helloworld.sample:50
00/hello

# 在Member集群中多次调用helloworld.sample服务
$ kubectl exec -it -n sample -c sleep --context=${REMOTE_CLUSTER_CTX} $(kubectl get pod -n sample -l app=sleep --context=${REMOTE_CLUSTER_CTX} -o jsonpath='{.items[0].metadata.name}') -- curl helloworld.sampl
e:5000/hello
```

NOTE: 如果部署正确，服务helloworld.sample的流量会分发到cluster-d161c69e-286b-4541-901f-b121c9517f4e和cluster-fcb0f8d3-9907-4a4a-9653-4584b367ee29集群，得到的响应会是v1和v2两种。

```
Hello version: v2, instance: helloworld-v2-758dd55874-cxjnw
Hello version: v1, instance: helloworld-v1-7c5df4c84d-vzjtj
```

在Host集群中通过如下命令查看istio为helloworld 服务注册的路由信息，如下，对于 helloworld 服务，istio 发现并注册了两个路由，一个指向本地的 helloworld 服务，另一个通过成员集群的 ingress gateway 的EIP来路由到从集群部署的helloworld 服务。

```
$ kubectl get pod -n sample -l app=sleep --context=${MAIN_CLUSTER_CTX} -o name | cut -f2 -d'/' |     xargs -I{} istioctl -n sample --context=${MAIN_CLUSTER_CTX} proxy-config endpoints {} --cluster "outbound|5
000||helloworld.sample.svc.cluster.local"
ENDPOINT                STATUS      OUTLIER CHECK     CLUSTER
10.2.2.14:5000          HEALTHY     OK                outbound|5000||helloworld.sample.svc.cluster.local
120.92.145.63:443       HEALTHY     OK                outbound|5000||helloworld.sample.svc.cluster.local

# 本地Host集群helloworld endpoint 信息
$ kubectl get ep helloworld -n sample --context=${MAIN_CLUSTER_CTX}
NAME         ENDPOINTS         AGE
helloworld   10.2.2.14:5000   20m

# 远程Member集群helloworld endpoint 信息
$ kubectl get svc -n istio-system istio-ingressgateway --context=${REMOTE_CLUSTER_CTX}
NAME                   TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)                                                                      AGE
istio-ingressgateway   LoadBalancer   10.254.60.1    120.92.145.63    15021:31509/TCP,80:32473/TCP,443:30816/TCP,15443:31135/TCP   20m
```

支持裸金属服务器

容器服务支持金山云裸金属服务器作为集群中的节点。容器直接运行于高性能裸金属服务器上，无虚拟化性能损失，将为您带来更强劲的性能体验。

使用前须知

- 金山云容器服务暂不负责裸金属服务器的创建，裸金属服务器仅支持在容器集群创建完成后移入到集群，您需要在裸金属服务器控制台提前创建相应的裸金属服务器实例。
- 支持任意裸金属服务器机型。

  需要注意的是，裸金属服务器必须同时满足以下条件才能移入集群：

  - 裸金属服务器与集群在同一个VPC内。
  - 裸金属服务器的状态为运行中。

- 裸金属服务器安装了容器Agent且没有被加入到k8s集群。

关于裸金属服务器安装容器Agent的问题

- 您可在创建裸金属服务器时，选择安装容器Agent，该裸金属服务器可直接添加至集群，无需重装系统；若您在创建裸金属服务器时未安装容器Agent，该裸金属服务器需重装系统，在此过程中我们会为您默认安装容器Agent。

  注意事项：

  - 重装系统后裸金属服务器系统盘、数据盘所有数据将被清除；
  - 您在操作前请做好重要数据的备份工作，以免数据丢失给您造成损失；

- 由于Kubernetes以及Docker软件对于镜像内核版本的要求，我们仅支持使用以下镜像开机的裸金属服务器安装容器的Agent：

  - **标准镜像**：CentOS-7.3及以上、Ubuntu-18.04。
  - **用户自定义镜像**：对于自定义镜像，目前我们默认不支持安装容器的Agent，如您有使用自定义镜像需求，可以提工单进行申请，提供自定义镜像的相关信息，容器业务线会对此自定义镜像进行校验。若满足Kubernetes等相关的要求，则会为您配置支持安装容器Agent的选项。

使用流程

创建裸金属服务器主机

1. 登录裸金属服务器控制台。
2. 点击**新建裸金属服务器**，进入创建裸金属服务器的流程。
3. 在**设置基本信息**流程中，勾选安装容器的Agent，如下图：

添加裸金属服务器至集群

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要添加裸金属服务器的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 单击**添加裸金属服务器节点**，进入添加裸金属服务器节点页面。
6. 按需选择添加至集群的裸金属服务器，点击**下一步**。
7. 根据以下提示进行配置，配置完成后点击**添加至集群**，即可完成节点的添加。
   - 若您选择的裸金属服务器未安装容器Agent，则需重装系统，在此过程中我们会为您默认安装容器Agent。
     - 镜像：按需选择重装时的镜像。
     - 登录方式：按需选择登录方式，支持设置仅密钥、密钥加密码登录。

       注：以上配置只对未安装容器Agent的裸金属服务器生效。

   - 容器存储目录：按需选择是否自定义容器和镜像存储目录，建议存储到数据盘，如不设置，默认为/data/docker。
   - 容器运行时：提供Docker和Containerd两种运行时，用户根据实际需求选择运行时，详情请见如何选择Containerd和Docker。
   - 容器存储目录：按需选择是否自定义容器和镜像存储目录，建议存储到数据盘，如不设置，默认为/data/docker。
   - 高级配置（可选）：按需配置以下信息
     - 部署前执行脚本：指定自定义数据来配置Node，即当节点部署前执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/pre_userscript路径查看。
     - 部署后执行脚本：指定自定义数据来配置Node，即当节点部署后执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/userscript路径查看。
     - 封锁：根据需求勾选是否封锁worker节点，封锁worker节点后，将不接受新的Pod调度到该节点，若需重新调度，则要手动取消封锁的节点。更多操作请见详情。
     - Label：根据需求进行Label自定义设置。

从集群中移出裸金属服务器节点

我们支持从集群中移出裸金属服务器。 进入节点列表页面，选择需要移出的裸金属服务器节点，点击【移出节点】即可节点的移出操作。

说明

- 在移出节点的过程中，会卸载该节点上k8s相关的组件，约1min内生效，请不要重复进行移出操作。
- 裸金属服务器节点仅会被移出集群，不会进行销毁。如需要销毁该裸金属服务器，请在裸金属服务器控制台操作。
- 移出集群的裸金属服务器支持再次移入集群。

GPU裸金属服务器支持

本期我们支持NVIDIA GPU和寒武纪

NVIDIA GPU

启动Kubernetes对NVIDIA GPU的支持需要GPU预装英伟达的驱动和CUDA，我们推荐您在移入GPU裸金属服务器到k8s集群前，提前预装以上软件。

说明

- 创建NVIDIA GPU时，您需要安装容器Agent。
- 若您的GPU裸金属服务器没有预装以上软件，我们会在移入集群的过程中为您默认安装英伟达驱动和CUDA（版本10.1），安装过程中会自动重启GPU裸金属服务器。

寒武纪

Cambricon Kubernetes Device Plugin 是基于 Kubernetes 在 1.8 版本后推出的 device plugin 架构，它可以在不改变 Kubernetes 源代码的情况下，通过在裸机中部署设备插 件或作为 deamonset 运行来管理 MLU 资源，实现了自动向集群上报 MLU100 设备数目、 追踪 MLU100 设备健康状况和 运行 MLU 使能的容器等功能。

说明

- 仅支持Kubernetes 1.8以上版本的集群。
- 用户创建寒武纪主机&选装容器标准组件后，执行移入k8s集群操作，由容器业务线安装寒武纪驱动等相关软件（此过程不需要重启实例），支持Kubernetes调度。

新增节点

若目前集群内节点的规模暂不能满足业务运行的要求，金山云容器服务支持向已有的集群中新增节点。新增节点的流程与创建集群流程基本一致。

新增节点流程如下：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新增节点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 在节点列表中，点击**新增节点**，进入新增节点的页面。
6. 可查看该**集群配置**基本信息，包括集群名称，数据中心，集群网络，Pod CIDR、Service CIDR和集群描述等，点击**下一步**进行节点配置。
7. 根据以下提示进行**节点配置**，配置完成后点击**下一步**。
   - 计费方式：支持包年包月、按量付费（按日月结）和按量付费等计费方式。
   - 节点类型：目前支持选择普通云主机和专属云主机创建集群。
   - Worker机型：
     - 可用区：选择节点部署的可用区。
     - 节点网络：选择节点部署的VPC内的子网信息。
     - 机型：用户根据需求选择云服务器机型。
     - 镜像：目前金山云容器服务支持CentOS-7.3 64位、CentOS-7.6 64位、Debian-8.2 64位和Ubuntu-18.04 64位节点镜像。
     - 系统盘：用户根据需求选择系统盘大小。
     - 数据盘：自定义数据盘大小以及选择是否格式化并挂载至指定目录下。

        - 购买数量：用户根据需求选择购买数量。
    - 容器运行时：提供Docker和Containerd两种运行时，用户根据实际需求选择运行时，详情请见如何选择Containerd和Docker
    - 容器存储目录：按需选择是否自定义容器和镜像存储目录，建议存储到数据盘，如不设置，默认为/data/docker。
    - 安全组：定义集群中节点所属的安全组，关于容器集群安全组推荐设置，请参考容器集群安全组推荐设置。
    - 高级配置（可选）：按需配置以下信息，
        - 部署前执行脚本：指定自定义数据来配置Node，即当节点部署前执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/pre_userscript路径查看。
        - 部署后执行脚本：指定自定义数据来配置Node，即当节点部署后执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/userscript路径查看。
        - 封锁：根据需求勾选是否封锁worker节点，封锁worker节点后，将不接受新的Pod调度到该节点，若需重新调度，则要手动取消封锁的节点。更多操作请见详情。
        - Label：根据需求进行Label自定义设置。

            注：容器服务支持将主机创建在同一地域下不同可用区下的不同子网中，以实现跨可用区容灾。

8. 根据以下提示进行**设置基本信息**，配置完成后点击**下一步**。
    - 所属项目：自定义购买的云资源的所属项目。
    - 服务器名称：自定义服务器名称，2-64个字符，支持中文，英文，数字，以及特殊字符-,.!$^*()%#&+/:;<=>[]_`{|}~
    - 登录方式：用户自定义选择登录方式，支持设置密码和密钥登录。
9. 点击**创建**，则新增加的节点会出现该在集群的节点列表中。

移出节点

操作步骤

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要移出节点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 选择对应的节点，点击**移出节点** 。
6. 在移出节点弹窗中，按需选择是否删除非包年包月云服务器实例，系统盘和数据盘上的数据会被删除，无法恢复，请谨慎操作，点击**确认**完成移出节点操作。

注意事项

- 包年包月的云服务器实例和裸金属服务器仅作移出集群操作，不会被删除。
- 当节点状态变为**移除中**时，可点击**强制移出**，进行强制移出操作。
- 按量计费节点移出节点若不选择销毁，将继续扣费
- 不建议在云服务器控制台直接销毁容器集群中的服务器。

添加已有节点

容器服务支持向集群中添加已有节点。

前提条件

- 该云服务器没有加入到其他k8s集群中。
- 该云服务器与当前k8s集群在同一个VPC下。
- 该云服务器处于关闭状态。

操作步骤

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要添加已有节点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 在节点列表页中，点击**添加已有节点**，进入添加已有节点的页面。
6. 根据以下提示进行**节点配置**，配置完成后点击**下一步**。
    - Worker机型：您可选择添加当前集群所在VPC下可用节点，以及重装时的镜像。 注：所选云服务器需重装系统，重装后云服务器系统盘所有数据将被清除；请在重装系统前做好数据备份并关闭实例，以免数据丢失给你造成损失。
    - 数据盘挂载：按需设置数据盘挂载路径，仅针对第一块数据盘生效。 注：已格式化的ext3，ext4，xfs文件系统的数据盘将直接挂载，其他文件系统或未格式化的数据盘将自动格式化为ext4并挂载。
    - 容器运行时：提供Docker和Containerd两种运行时，用户根据实际需求选择运行时，详情请见如何选择Containerd和Docker
    - 容器存储目录：按需选择是否自定义容器和镜像存储目录，建议存储到数据盘，如不设置，默认为/data/docker。
    - 高级配置（可选）：按需配置以下信息：
        - 部署前执行脚本：指定自定义数据来配置Node，即当节点部署前执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/pre_userscript路径查看。
        - 部署后执行脚本：指定自定义数据来配置Node，即当节点部署后执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/userscript路径查看。
        - 封锁：根据需求勾选是否封锁worker节点，封锁worker节点后，将不接受新的Pod调度到该节点，若需重新调度，则要手动取消封锁的节点。更多操作请见详情。
        - Label：根据需求进行Label自定义设置。
7. 根据以下提示进行**设置基本信息**，配置完成后点击**下一步**。
    - 所属项目：自定义购买的云资源的所属项目。
    - 服务器名称：自定义服务器名称，2-64个字符，支持中文，英文，数字，以及特殊字符-,.!$^*()%#&+/:;<=>[]_`{|}~
    - 登录方式：用户自定义选择登录方式，支持设置密码和密钥登录。
8. 点击**创建**，新添加的已有节点会出现该在集群的节点列表中。

驱逐与封锁节点

封锁节点

进行封锁节点操作后，后续新的Pod不会被调度到此节点，该节点上存在的Pod不会受到影响，可以正常对外服务。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要封锁节点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 选择对应的节点，点击**更多**> **封锁** 。
6. 在封锁节点弹窗中，点击**确认**完成封锁节点操作。

取消封锁节点

进行取消封锁节点操作后，将允许新的Pod调度到该节点。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要取消封锁节点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 选择对应的节点，点击**更多**> **取消封锁** 。
6. 在取消封锁节点弹窗中，点击**确认**完成取消封锁节点操作。

驱逐节点

进行驱逐节点操作后，将把节点内的所有Pod（不包含DaemonSet管理的Pod）驱逐到集群内其他节点上，并将该节点设置为封锁状态。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要驱逐节点的集群ID，进入该集群操作页面。

4. 选择**节点管理** > **节点**，进入节点列表页。
5. 选择对应的节点，点击**更多**> **驱逐** 。
6. 在驱逐节点弹窗中，点击**确认**完成驱逐节点操作。

管理节点标签

容器服务支持对集群节点进行标签管理。

添加标签

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要添加节点标签的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 点击节点列表上方**标签和污点**。
6. 选择**标签**页签，点击**添加标签**。
7. 在新建标签弹窗中，用户根据需求输入标签的名称和值，然后单击**确定**，即可在标签页面看到此次添加的标签。

编辑标签

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要编辑节点标签的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 点击节点列表上方**标签和污点**。
6. 选择**标签**页签，选择某个节点后点击标签的修改图标。
7. 在编辑标签弹窗中，用户根据需求编辑标签的值，然后单击**确定**，即可在标签页面看到此次编辑。

删除标签

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要删除节点标签的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 点击节点列表上方**标签和污点**。
6. 选择**标签**页签，选择某个节点后点击标签的删除图标。
7. 在删除标签弹窗中，用户根据需求删除标签，然后单击**确定**，即可看到节点标签被删除。

   注意：请确认此标签没有被业务使用，否则会影响Pod重启后的调度。

管理节点污点

容器服务支持对集群节点进行污点管理。您可按需为节点设置污点，使得节点可以排斥某些 Pod。

添加污点

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要添加节点污点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 点击节点列表上方**标签和污点**。
6. 选择**污点**页签，点击**添加污点**。
7. 在新建污点弹窗中，用户根据需求输入标签的名称、值和Effect，然后单击**确定**，即可在污点页面看到此次添加的污点。

   Effect有以下三中类型可选：

   - NoSchedule：禁止新的Pod调度至节点。
   - PreferNoSchedule：尽量避免将没有匹配容忍的 Pod 调度到该节点上。
   - NoExecute：不能在该节点上运行（如果已经运行，将被驱逐）。

编辑污点

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要编辑节点污点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 点击节点列表上方**标签和污点**。
6. 选择**污点**页签，选择某个节点后点击污点的修改图标。
7. 在编辑污点弹窗中，用户根据需求编辑污点的值和Effect，然后单击**确定**，即可在标签页面看到此次编辑。

删除污点

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要删除节点污点的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点**，进入节点列表页。
5. 点击节点列表上方**标签和污点**。
6. 选择**污点**页签，选择某个节点后点击污点的删除图标。
7. 在删除污点弹窗中，用户根据需求删除污点，然后单击**确定**，即可看到节点污点被删除。

节点池概述

通过节点池，您可以更加高效便捷地管理Kubernetes集群内的节点，例如快速创建、分组管理和销毁节点，开启/关闭节点自动弹性伸缩，修改节点池配置等。以及实现节点的动态扩缩容：当集群中出现因资源不足而无法调度的实例（Pod）时，自动触发扩容；当满足节点空闲等缩容条件时，自动触发缩容。

注：节点池功能已上线，已覆盖伸缩组的全部功能。若您的集群中已创建伸缩组，现在仍可继续使用，但无法进行任何操作，建议您点击确认对原伸缩组进行转换节点池操作。

节点池架构

节点池可以包含一个或多个节点，您可以根据实际使用情况在集群中创建不同大小和类型的节点池，来分组管理这些节点。每个节点池对应独有的配置，您可根据实际使用情况灵活配置节点池信息、节点模板配置。节点池整体架构如下图所示：

- 节点池信息包括：
  - 是否开启弹性伸缩
  - 节点数量
  - 节点数量范围
  - Label、Taints
- 节点模板包括：
  - 计费方式
  - 节点配置（节点类型、操作系统、系统盘、数据盘、数据盘挂载、容器存储目录）
  - 关联VPC
  - 关联子网、多子网扩展策略
  - 安全组
  - 部署前/后执行脚本
  - 封锁

功能点与相关注意事项

功能	功能说明	注意事项
创建节点池	新增节点池	支持计费方式为按量付费（按日月结）、按量付费的节点
开启/关闭弹性伸缩	• 开启弹性伸缩后，节点池内节点数量将随集群负载情况自动调整； • 未开启弹性伸缩，节点池内节点数量需用户手动进行调整	当节点池弹性伸缩已开启，允许关闭弹性伸缩（该操作不可逆）
调整节点池节点数量	调整节点池内节点数量	• 若节点池已开启弹性伸缩，节点池内节点数量将随集群负载情况自动调整，不支持手动更改； • 若手动减小节点数量，先缩容节点池扩出来的节点，后缩容添加已有节点至节点池的节点（对于手动加入的节点，缩容后只会从集群中移出，请去云服务器控制台作删除处理）
调整节点池信息	可对节点池信息进行修改，例如：节点池名称、关闭弹性伸缩、节点数量、节点数量范围、Label、Taints	可以按需选择Label、Taints更新是否对节点池内所有已有节点生效： • 勾选后，本次对Label、Taints的更新会对节点池内已有节点以及扩容出的节点生效。可能会引起已有节点上的Pod重新调度，请谨慎变更； • 若未勾选，本次对Label、Taints的更新只会对扩容出的节点生效
修改节点模板配置	可对节点配置配置、节点类型、操作系统、系统盘、数据盘、数据盘挂载、容器存储目录、关联子网、多子网扩展策略、安全组、部署前执行脚本、部署后执行脚本、封锁等信息进行修改	修改节点模板配置，只会对扩容出来的节点生效
管理节点池	可对节点池信息、节点模板配置进行统一更改管理	--
删除节点池	删除节点池时可按需选择是否同步删除节点池内的节点	节点删除后，系统盘和数据盘上的数据会被删除，无法恢复，请谨慎操作
添加已有节点至节点池	可添加集群内不属于任何节点池的节点，要求如下： • 该节点与节点池属于相同子网； • 只可添加集群内开启状态的节点至节点池； • 仅支持集群内按量付费（按日月结）、按量付费的节点加入到节点池	已开启弹性伸缩节点池：完全自动化扩缩容操作，不可进行添加已有节点操作
移出节点池内节点	支持将节点池内的节点移出集群，移出时可选择是否同步删除节点	已开启弹性伸缩节点池：完全自动化扩缩容操作，不可进行移出节点操作

说明： 1、只能在创建节点池的时候开启弹性伸缩功能，开启弹性伸缩后，节点池内节点数量将随集群负载情况自动调整，完全自动化扩缩容。不支持手动更改节点数量、添加已有节点、移出节点等操作；允许后续关闭弹性伸缩配置，但反向操作不允许。关于弹性伸缩更多详情，请见集群弹性伸缩。 2、请通过容器控制台进行节点池相关配置，请勿通过弹性伸缩控制台进行操作，以免影响节点池相关功能正常运作。

后续相关操作

您可以登录容器服务控制台并参考以下文档，进行对应节点池操作：

- 创建节点池
- 查看节点池
- 管理节点池
- 删除节点池
- 存量伸缩组转换节点池

创建节点池

本章将介绍如何通过容器服务控制台创建节点池，具体步骤如下：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要创建节点池的集群ID，进入该集群操作页面。
4. 选择**节点管理** > **节点池**，进入节点池列表页。
5. 在节点池列表上方，点击**创建节点池**，进入创建节点池的页面。
6. 在创建节点池页面中，可按需对以下信息进行配置：
   - 节点池名称：用户自定义节点池的名称，2-64个字符，支持中文，英文，数字，以及特殊字符-,.!$^*()%#&+/:;<=>[]_`{|}~
   - 弹性伸缩：节点池是否开启弹性伸缩，默认为关闭状态。
   - 计费方式：支持按量付费（按日月结）、按量付费两种计费模式，可根据实际需求进行选择。 **注：开启弹性伸缩的节点池只支持计费方式为按量付费的节点**
   - 机型配置：用户根据实际情况配置数据中心、CPU、内存、是否支持IPV6、架构、分类、镜像、系统盘等机型信息。
   - 数据盘：自定义数据盘大小以及选择是否格式化并挂载至指定目录下。
   - 数据盘标签：按需输入或选择已有标签，最多可添加10个数据盘标签。
   - 容器运行时：提供Docker和Containerd两种运行时，用户根据实际需求选择运行时，详情请见如何选择Containerd和Docker
   - 容器存储目录：按需选择是否自定义容器和镜像存储目录，建议存储到数据盘，如不设置，默认为/data/docker。
   - 登录方式：节点池内云服务器的登录方式，支持密钥登陆和密码登陆。
   - 所属项目：选择节点池中的机器所属的项目。
   - 关联VPC：默认集群所在VPC，不可更换。
   - 关联子网：选择节点池内机器所在子网。
   - 多子网扩展策略：有均衡分布以及优先选择两种策略，请根据实际需求进行选择：
     - 均衡分布：弹性服务器扩容时优先保证选择的子网列表中各子网下弹性服务器数量均衡，当无法在目标可用区下完成弹性服务器扩容时，按照优先选择原则选择其他子网。
     - 优先选择：弹性服务器扩容时目标子网的选择按照选择的子网列表的顺序进行优先级排序。
   - 安全组：选择节点池内机器的安全组信息。
   - 节点数量：节点池刚创建时的节点数量，请根据实际需求进行配置。
   - 节点数量范围：节点数量将在设定的节点范围内自动调节，不会超出该设定范围。
   - 高级配置（可选）：按需配置以下信息：
     - 部署前执行脚本：指定自定义数据来配置Node，即当节点部署前执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/pre_userscript路径查看。
     - 部署后执行脚本：指定自定义数据来配置Node，即当节点部署后执行的命令脚本，需要自行保证脚本的可重入及重试逻辑，脚本及其生成的日志文件可在节点的/usr/local/ksyun/kce/userscript路径查看。
     - 封锁：根据需求勾选是否封锁worker节点，封锁worker节点后，将不接受新的Pod调度到该节点，若需重新调度，则要手动取消封锁的节点。更多操作请见详情请见详情。
     - Label：为节点池设置 Label，会自动在节点池中的节点上设置 Label，从而实现服务的灵活调度策略，可根据需求进行Label自定义设置。
     - Taints：为节点池设置Taints，会自动在节点池中的节点上设置Taints，具体的参数设置请见详情。
7. 点击**创建**，执行创建节点池操作，在节点池列表页中可查看创建的节点池。

查看节点池

本章将介绍如何通过容器服务控制台查看已创建的节点池以及指定节点池的详细信息，详情如下：

查看节点池列表

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要查看节点池的集群ID，进入该集群操作页面。
4. 选择**节点池管理** > **节点池**，进入节点池列表页，即可看到已创建的节点池，若您创建了开启弹性伸缩的节点池，也可看到弹性伸缩配置信息。如下图所示：

查看节点池详情

1. 点击需要查看节点池详情的节点池名称。
2. 进入所选节点池详情页，可查看节点池信息以及节点模板配置，如下图所示：

查看节点池内节点信息

1. 点击需要查看节点池内节点信息的节点池名称。
2. 点击**节点管理**，进入节点管理页，可查看节点池内节点的相关信息，如下图所示：

管理节点池

本章将介绍如何通过容器服务控制台调管理节点池，包括调整节点池信息、调整节点模板配置、整体管理节点池、调整弹性伸缩配置、对节点池内节点进行相关操作等。

调整节点池信息

通过调整节点池信息，可针对节点池名称、弹性伸缩、节点数量、节点数量范围、Label、Taints等信息进行修改。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要调整节点池信息的集群ID，进入该集群操作页面。
4. 选择**节点池管理** > **节点池**，进入节点池列表页。
5. 选择需要调整节点池信息的节点池，点击**调整节点池信息**，如下图所示：
6. 在跳出的弹窗中，可按需对节点池信息进行修改，如下图所示：
   - 节点池名称：用户自定义节点池的名称，2-64个字符，支持中文，英文，数字，以及特殊字符-,.!$^*()%#&+/:;<=>[]_`{|}~
   - 弹性伸缩：可按需关闭弹性伸缩。 **注：您可按需关闭弹性伸缩，反向操作不允许。**
   - 节点数量：按需调整节点池内节点数量，该数量必须在节点数量范围之内。
   - 节点数量范围：节点数量将在设定的节点数量范围之内调节。
   - Label：为节点池设置 Label，会自动在节点池中的节点上设置 Label，从而实现服务的灵活调度策略，可根据需求进行Label自定义设置。
   - Taints：为节点池设置Taints，会自动在节点池中的节点上设置Taints。
   - Label、Taints更新对节点池内所有已有节点生效：可按需是否勾选。

     注： 1、勾选后，本次对Label、Taints的更新会对节点池内已有节点以及扩容出的节点生效；可能会引起已有节点上的Pod重新调度，请谨慎变更。 2、若未勾选，本次对Label、Taints的更新只会对扩容出的节点生效。
7. 点击**确定**，即可调整节点池相关信息。

调整节点模板配置

通过调整节点模板配置，可针对节点配置、节点类型、操作系统、系统盘、数据盘、数据盘标签、数据盘挂载、容器存储目录、关联子网、多子网扩展策略、安全组、部署前执行脚本、部署后执行脚本、封锁等信息进行修改。

1. 点击需要调整节点模板配置的节点池名称。
2. 进入所选节点池详情页，点击**编辑**，即可按需对节点模板进行修改，如下图所示：
3. 按需修改完毕后，点击**更新**，即可完成调整节点模板配置操作。

   注：调整节点模板配置，只会对扩容出来的节点生效，不影响节点池中正在运行的节点。

管理节点池

您可根据实际需求分别进行调整节点池信息、调整节点模板配置等操作，同时您也可选择整体管理节点池，整体进行设置规划。

1. 在节点池列表页中，选择需要整体管理的节点池，点击**管理**，如下图所示：
2. 在管理节点池页面中，按需修改相关信息，点击**更新**即可。

调整弹性伸缩配置

若您的节点池中存在开启弹性伸缩的节点池，可按需对弹性伸缩配置中的集群扩容策略、集群自动缩容进行修改。

1. 在节点池列表页中，点击集群扩容策略或集群自动缩容的修改图标，如下图所示：
2. 在弹出的弹窗中，可按需对以下信息进行修改：
   - 扩容策略：当集群中有多个开启弹性伸缩的节点池时，可以通过选项配置选择节点池的策略，支持如下三种方式：
     - random：随机选择节点池进行扩容。
     - most-pods：选择容量最大（能调度更多pod）的节点池进行扩容。
     - least-waste：以最小浪费原则选择，选择调度pod后可用资源剩余更少的节点池进行扩容。
   - 自动缩容：默认不勾选。开启自动缩容时，集群中节点空闲资源较多时将触发缩容。详情请参见集群弹性伸缩。
   - 缩容配置：该配置项仅在开启自动缩容时显示，请根据实际需求进行设置。
     - 最大缩容并发数：可以同时缩容的最大空节点数，如果存在pod，每次缩容最多一个节点。默认值为10，可按需自定义设置。
     - Node已分配的资源/可分配资源小于：可设置Node已分配的资源/可分配资源在占比小于设定值时开始判断缩容条件。占比值范围需确保在0~80之间。默认值为50，可按需自定义设置。
     - 节点满足缩容条件：可自定义设置节点满足缩容条件时间超过几分钟之后会被缩容。默认值为10，可按需自定义设置。
     - 集群扩容：可自定义设置节点超过几分钟其期间没有执行任何扩展操作后判断缩容条件。默认值为10，可按需自定义设置。
3. 点击**确定**，即可设置成功。

   注： 1、调整弹性伸缩配置只对开启弹性伸缩的节点池生效。 2、若您不调整弹性伸缩配置，默认设置为随机扩容模式、关闭自动缩容。

节点池内节点相关操作

添加已有节点至节点池

您可添加集群内不属于任何节点池的节点至节点池，相关步骤如下：

1. 点击需要添加已有节点至节点池的节点池名称。
2. 点击**节点管理**，进入节点管理页。
3. 点击**添加已有节点**，进入添加已有节点的页面。
4. 您可按需选择添加当前集群所在VPC、节点池相同子网下可用的节点至节点池。
5. 点击**确定**，即可添加已有节点至节点池。

   注： 1、仅支持集群内按量付费（按日月结）、按量付费的节点加入到节点池。 2、只可添加集群内开启状态的实例至节点池。

开启/取消节点缩容保护

您可按需选择是否开启节点缩容保护，开启缩容保护后，该节点不会受弹性伸缩缩容、调整节点数量影响。相关步骤如下：

1. 点击需要进行相关操作的节点池名称。
2. 点击**节点管理**，进入节点管理页。
3. 选择需要缩容保护的节点，点击**缩容保护**，如下图所示：
4. 在跳出的弹窗中点击**确认**，即可对该节点进行缩容保护。

   您也可以按照上述步骤进行取消缩容保护操作，关闭缩容保护后，该节点会受弹性伸缩缩容、调整节点数量影响。

移出节点池内节点

1. 点击需要进行相关操作的节点池名称。
2. 点击**节点管理**，进入节点管理页。
3. 选择需要移出节点池的节点，点击**移出节点**，如下图所示：
4. 在移出节点的弹窗中，按需选择是否删除节点，系统盘和数据盘上的数据会被删除，无法恢复，请谨慎操作，点击**确认**完成移出节点操作。

其他操作

您也可按需对节点池中的节点进行驱逐与封锁操作，相关操作详情请见驱逐与封锁节点。

删除节点池

本章将介绍如何通过容器服务控制台删除已创建的节点池，操作步骤如下：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要删除节点池的集群ID，进入该集群操作页面。
4. 选择**节点池管理** > **节点池**，进入节点池列表页。
5. 选择需要删除的节点池，点击**删除**，如下图所示：
6. 在跳出的弹窗中，可按需选择是否同步删除节点池内节点，如下图所示：

注：节点删除后，系统盘和数据盘上的数据会被删除，无法恢复，请谨慎操作。

7. 点击**确定**，即可删除节点池。

存量伸缩组转换为节点池

节点池功能已上线，除了覆盖伸缩组的全部功能以外，您也可通过节点池方便地对节点进行分组管理：例如节点运维、节点配置、开启/关闭节点弹性伸缩、批量管理等。建议您点击确认对原伸缩组进行转换节点池操作。 本章将介绍如何通过容器服务控制台将集群中原伸缩组转换为节点池。步骤如下：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要相关操作的集群ID，进入该集群操作页面。
4. 选择**节点池管理** > **伸缩组**。
5. 在跳出的弹窗中，点击确定即可完成转换，如下图所示：

注：伸缩组转换节点池操作，会将您集群中原伸缩组转换为开启弹性伸缩的节点池，节点池沿用原伸缩组信息，不会对您已有的业务造成影响。

概述

在 Kubernetes 中，应用管理是需求最多、挑战最大的领域。Helm 项目提供了一个统一软件打包方式，支持版本控制，可以大大简化 Kubernetes 应用分发与部署中的复杂性。

Helm是Kubernetes下的包管理工具（类似于apt/pip/brew），将应用部署包含的多种Kubernetes 资源对象整合成单一对象（Chart），模板提供默认部署配置，部署时可以进行变量替换修改，同时提供了Helm应用生命周期管理能力，支持升级、回滚、版本追踪等。更多详情请查看 Helm官方文档。

金山云容器服务集成了Helm相关功能，支持一键化开通Helm服务，支持Helm应用可视化管理。

Helm应用管理

金山云容器服务集成了Helm相关功能，您可以通过控制台可视化管理Helm应用。

Helm应用版本升级

目前金山云容器服务已全面支持Helm v3版本，基于v3相比于v2版本在易用性和安全性上的提升，我们建议您使用Helm 3进行应用部署。

注：容器服务自2021年7月15日起不再维护Helm 2服务，不再支持Helm 2服务开通，已部署应用不受影响。

对于已经开通Helm 2服务并部署过应用的用户，您仍能通过控制台Helm 2页面进行应用的部署与管理。若需要进行v2版本到v3版本的迁移，请参考Helm v2 迁移到 v3。

开通Helm 2服务中部署的组件支持通过**移除Helm2**按钮一键卸载。

Helm应用基本操作

目前Helm应用基于Helm 3提供完整的Helm应用管理能力。

注：为保证良好兼容性，推荐集群Kubernetes版本范围为1.17-1.20。

创建Helm应用

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**Helm应用**>**Helm 3**，进入Helm 3应用列表页。
3. 点击**新建**，配置相关信息，即可新建Helm应用。

- **应用名称**：填写Helm应用名称，此字段全集群唯一。
- **Chart来源**：选择需要部署的Helm Chart的类型，目前支持用户在金山云私有的Helm Chart，Ksyun 官方Helm Chart和第三方Helm Chart。
- **values.yaml**：用户可以自定义Chart中的默认变量值。对于来源金山云私有Helm Chart，Ksyun Helm Chart的Chart，支持在原始value.yaml中进行修改。对于第三方仓库来源Chart，用户自定义变量将替换chart中默认变量值。

4. 点击**部署**，创建成功返回Helm应用列表页面。

查看Helm应用详情

点击Helm应用名称，进入Helm应用详情页面，展示信息如下：

- **应用基本信息**：Helm应用名称，所在集群/命名空间，Helm描述，创建/最后部署时间。
- **应用资源列表**：支持查询应用中部署的工作负载、服务、其他资源，以及跳转到相应资源列表。
- **参数**：支持展示来源金山云私有Helm Chart，Ksyun Helm Chart部署的values.yaml详情。
- **版本管理**：Helm应用发布版本的管理，提供一键回滚功能。

更新Helm应用

Helm应用列表，选择对应的Helm应用，点击**更新**，进入更新Helm应用的页面。支持更新chart来源、名称、版本，以及修改自定义变量。

删除Helm应用

点击Helm应用列表页面的**删除**按钮，执行删除Helm应用的操作。

注意：删除Helm应用会同步删除该应用下的所有资源，请谨慎操作。

使用Helm本地客户端连接集群

这里我们介绍如何使用Helm本地客户端连接集群。

在本地计算机上安装和配置 kubectl

详见通过kubectl连接Kubernetes集群。

下载Helm客户端

下载对应的客户端版本（金山云容器服务Helm 3 默认安装版本为3.5.4，Helm 2默认安装版本为v2.13.0）。

```
wget https://get.helm.sh/helm-v3.5.4-linux-amd64.tar.gz
tar -zxvf helm-v3.5.4-linux-amd64.tar.gz
mv linux-amd64/helm /usr/local/bin/helm
```

通过客户端管理Helm应用

执行以下命令，验证Helm客户端。

```
[root@vm10-0-11-193 ~]# helm ls
NAME    NAMESPACE   REVISION    UPDATED                                 STATUS      CHART               APP VERSION
test    default     1           2021-07-15 21:33:39.78202073 +0800 CST  deployed    postgresql-10.5.2   11.12.0
```

Deployment管理

Deployment概述

本模块提供了基于Kubernetes原生的Deployment的创建、配置、删除等生命周期的管理指南。

创建Deployment

您可以通过以下步骤在容器服务控制台通过镜像创建一个Deployment：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建Deployment的集群，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。
5. 单击页面左上角**新建**，根据实际需求完成配置，单击**创建**，即可新建Deployment。

以下为创建过程中的配置详情：

基本信息配置

- 名称：用户自定义Deployment的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 地域：选择您部署Deployment所在的地域。
- 部署集群：选择Deployment运行的集群。
- 命名空间：选择Deployment所在集群的命名空间。
- 标签：为Deployment创建一个标签，用于标示该资源。
- 描述：创建Deployment的相关信息，用户自定义填写。

部署配置

存储卷

目前支持使用主机路径、本地路径、金山云云硬盘、文件存储、已有PVC等存储类型。

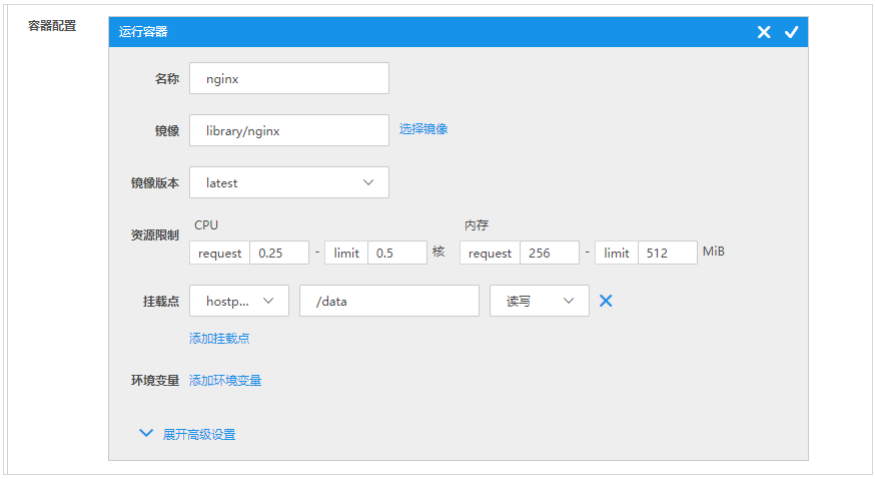
- 类型：包含主机路径、本地路径、金山云云硬盘、文件存储、已有PVC、使用ConfigMap/Secret。
- 存储卷名称：存储卷的名称。
- 资源名称：选择相应存储资源的名称。
- 其他信息：指定容器要挂载的路径。

备注：

使用本地硬盘时，若不指定源路径，则默认分配临时路径（对应k8s存储中EmpytDir）。

当使用云硬盘作为存储卷时，存储卷的名称为云硬盘的id，不可修改。

容器配置



设置运行的容器信息：

- 名称：容器的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 镜像：点击**选择镜像**，从镜像仓库中选择；或输入镜像仓库地址。
- 镜像版本：镜像的tag。
- 资源限制：设定容器所使用的CPU和内存资源的限制。
- 环境变量：在容器中添加环境变量，一般用于通过环境变量设置参数，目前仅支持手动添加。
- 实例数量：一个实例由相关的一个或多个容器构成，用户自定义实例数量。

镜像访问凭证

目前支持从私有镜像仓库拉取镜像时，为容器配置镜像访问凭证（对应yaml中的imagePullSecret）。

- 初始状态下提供默认选项ksyunregistrykey，可以匹配金山云镜像仓库中的私有镜像。
- 若您使用的镜像来自第三方私有镜像仓库，可以通过**添加镜像访问凭证** > **使用新的访问凭证** > **设置访问凭证信息**，设置新的访问凭证的名称，以及第三方仓库的域名、用户名、密码，即完成添加第三方私有镜像仓库访问凭证。

访问设置

- 取消开启关联Service：不提供任何从前端服务访问到容器的入口。
- 公网访问：通过金山云公网负载均衡暴露服务，可以直接被公网访问。
- VPC内网访问：通过金山云私网负载均衡暴露服务，可以被同一VPC下其他集群或者云服务器访问。
- 集群内访问：通过集群的内部IP暴露服务，服务只能在集群内部访问。
- 主机端口访问：通过任意一个节点的ip:nodeport，可以从集群外访问。

配置完成后，点击**创建**，返回部署列表页面，查询部署的状态。

Deployment基本操作

更新部署

在Deployment列表页面，点击**更新**，进入更新部署页面，用户可以根据业务需要更新Deployment的配置。

更新策略：

- 滚动更新：对实例进行逐个更新，这种方式可以让您不中断业务实现对服务的更新，您可自定义滚动更新的参数。
- 删除重建：直接销毁所有实例，启动相同数量的新实例。

调节实例数量

在Deployment列表页面，点击**调节实例数量**，执行调节实例数量的操作。

选择**手动调节**，点击+、-或者直接输入实例数量调节，设置完成后，点击**确定**完成修改。

或选择**自动调节**，设置触发策略和实例范围，实现弹性调整实例的数量。

重新部署

重新部署是指Deployment中的容器重新部署，并重新拉取镜像。 在Deployment列表页面，点击**重新部署**，执行重新部署的操作。

删除部署

在Deployment列表页面，点击**删除**，执行删除Deployment的操作。

用户可以根据需要选择是否在删除Deployment的同时删除其关联的Service。

StatefulSet管理

概述

StatefulSet主要用于部署有状态服务的工作负载，相应pod具有稳定的标识符，并支持稳定持久的存储，有序的部署、删除和伸缩。

本模块提供了基于Kubernetes原生的StatefulSet的创建、配置、删除等生命周期的管理指南。

创建StatefulSet

您可以通过以下步骤在容器服务控制台通过镜像创建一个StatefulSet：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建StatefulSet的集群，进入该集群操作页面。
4. 选择**工作负载** > **StatefulSet**，进入StatefulSet列表页。
5. 单击页面左上角**新建**，根据实际需求完成配置，单击**创建**，即可新建StatefulSet。

以下为创建过程中的配置详情：

基本信息配置

- 名称：用户自定义StatefulSet的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 地域：选择您部署StatefulSet所在的地域。
- 部署集群：选择StatefulSet运行的集群。
- 命名空间：选择StatefulSet所在集群的命名空间。
- 描述：创建StatefulSet的相关信息，用户自定义填写。

部署配置

存储卷

目前支持使用主机路径、本地路径、金山云云硬盘、文件存储、已有PVC等存储类型。

- 类型：包含主机路径、本地路径、金山云云硬盘、文件存储、已有PVC、使用ConfigMap/Secret。
- 存储卷名称：存储卷的名称。
- 资源名称：选择相应存储资源的名称。
- 其他信息：指定容器要挂载的路径。

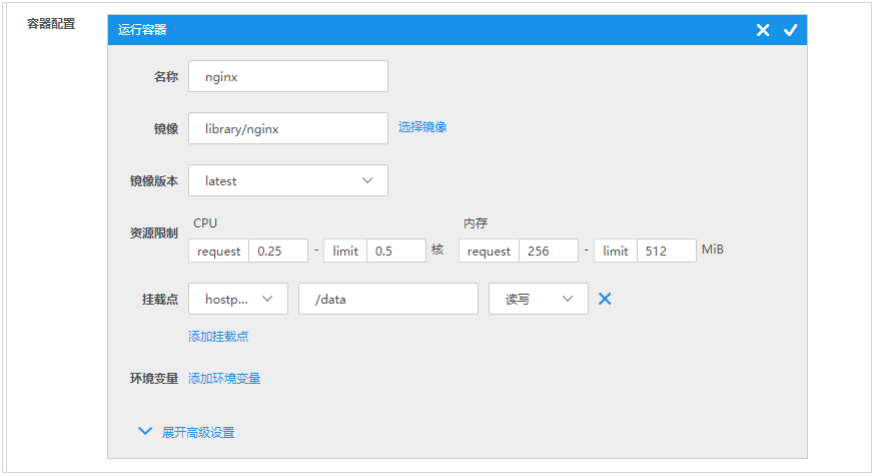
备注：

使用本地硬盘时，若不指定源路径，则默认分配临时路径（对应k8s存储中EmpytDir）。

当使用云硬盘作为存储卷时，存储卷的名称为云硬盘的id，不可修改。

容器配置

设置运行的容器信息：



- 名称：容器的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 镜像：点击**选择镜像**，从镜像仓库中选择；或输入镜像仓库地址。
- 镜像版本：镜像的tag。
- 资源限制：设定容器所使用的CPU和内存资源的限制。
- 环境变量：在容器中添加环境变量，一般用于通过环境变量设置参数，目前仅支持手动添加。
- 实例数量：一个实例由相关的一个或多个容器构成，用户自定义实例数量。

镜像访问凭证

目前支持从私有镜像仓库拉取镜像时，为容器配置镜像访问凭证（对应yaml中的imagePullSecret）。

- 初始状态下提供默认选项ksyunregistrykey，可以匹配金山云镜像仓库中的私有镜像。
- 若您使用的镜像来自第三方私有镜像仓库，可以通过**添加镜像访问凭证** > **使用新的访问凭证** > **设置访问凭证信息**，设置新的访问凭证的名称，以及第三方仓库的域名、用户名、密码，即完成添加第三方私有镜像仓库访问凭证。

访问设置

- 取消开启关联Service：不提供任何从前端服务访问到容器的入口。
- 公网访问：通过金山云公网负载均衡暴露服务，可以直接被公网访问。
- VPC内网访问：通过金山云私网负载均衡暴露服务，可以被同一VPC下其他集群或者云服务器访问。
- 集群内访问：通过集群的内部IP暴露服务，服务只能在集群内部访问。
- 主机端口访问：通过任意一个节点的ip:nodeport，可以从集群外访问。

配置完成后，点击**创建**，返回StatefulSet列表页面，查询StatefulSet的状态。

StatefulSet基本操作

更新StatefulSet

点击容器控制台左侧导航栏，进入**StatefulSet**列表页面，点击**更新**，进入更新StatefulSet页面，用户可以根据业务需要更新StatefulSet的配置。

更新策略：

- 滚动更新：对实例进行逐个更新，这种方式可以让您不中断业务实现对服务的更新，您可自定义滚动更新的参数。
- Ondelete：默认的更新策略，用户手动删除旧pod后触发新pod的创建。

Pod管理策略：

- OrderedReady：StatefulSet的默认管理策略，即按照顺序启动或终止所有的Pod，启动或者终止其他Pod前，需要等待Pod进入Running/Ready/完全停止状态。
- Parallel：并行的启动或终止所有的Pod，启动或终止其他Pod前，无需等待Pod进入Running/Ready/完全停止状态。

调节实例数量

进入StatefulSet列表页面，点击**调节实例数量**，执行调节实例数量的操作。当前版本仅支持对StatefulSet数量的手动调节，点击+、-或者直接输入实例数量调节。

删除StatefulSet

进入StatefulSet列表页面，点击**删除**，执行删除StatefulSet的操作。

DaemonSet管理

DaemonSet概述

DaemonSet可以保证全部（某些）节点上运行一个pod，适用于在每个节点上部署集群存储、日志收集等守护进程。本模块提供了基于Kubernetes原生的DaemonSet的创建、配置、删除等生命周期的管理指南。

创建DaemonSet

您可以通过以下步骤在容器服务控制台通过镜像创建一个DaemonSet：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建DaemonSet的集群，进入该集群操作页面。
4. 选择**工作负载** > **DaemonSet**，进入DaemonSet列表页。
5. 单击页面左上角**新建**，根据实际需求完成配置，单击**创建**，即可新建DaemonSet。

以下为创建过程中的配置详情：

基本信息配置

- 名称：用户自定义DaemonSet的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 地域：选择您部署DaemonSet所在的地域。
- 部署集群：选择DaemonSet运行的集群。
- 命名空间：选择DaemonSet所在集群的命名空间。
- 描述：创建DaemonSet的相关信息，用户自定义填写。

部署配置

存储卷

目前支持使用主机路径、本地路径、金山云云硬盘、文件存储、已有PVC等存储类型。

- 类型：包含主机路径、本地路径、金山云云硬盘、文件存储、已有PVC、使用ConfigMap/Secret。
- 存储卷名称：存储卷的名称。
- 资源名称：选择相应存储资源的名称。
- 其他信息：指定容器要挂载的路径。

备注：

使用本地硬盘时，若不指定源路径，则默认分配临时路径（对应k8s存储中EmpytDir）。

当使用云硬盘作为存储卷时，存储卷的名称为云硬盘的id，不可修改。

容器配置

设置运行的容器信息：

- 名称：容器的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 镜像：点击**选择镜像**，从镜像仓库中选择；或输入镜像仓库地址。
- 镜像版本：镜像的tag。
- 资源限制：设定容器所使用的CPU和内存资源的限制。
- 环境变量：在容器中添加环境变量，一般用于通过环境变量设置参数，目前仅支持手动添加。
- 实例数量：一个实例由相关的一个或多个容器构成，用户自定义实例数量。

镜像访问凭证

目前支持从私有镜像仓库拉取镜像时，为容器配置镜像访问凭证（对应yaml中的imagePullSecret）。

- 初始状态下提供默认选项ksyunregistrykey，可以匹配金山云镜像仓库中的私有镜像。
- 若您使用的镜像来自第三方私有镜像仓库，可以通过**添加镜像访问凭证** > **使用新的访问凭证** > **设置访问凭证信息**，设置新的访问凭证的名称，以及第三方仓库的域名、用户名、密码，即完成添加第三方私有镜像仓库访问凭证。

访问设置

- 取消开启关联Service：不提供任何从前端服务访问到容器的入口。
- 公网访问：通过金山云公网负载均衡暴露服务，可以直接被公网访问。
- VPC内网访问：通过金山云私网负载均衡暴露服务，可以被同一VPC下其他集群或者云服务器访问。
- 集群内访问：通过集群的内部IP暴露服务，服务只能在集群内部访问。
- 主机端口访问：通过任意一个节点的ip:nodeport，可以从集群外访问。

配置完成后，点击**创建**，返回DaemonSet列表页面，查询DaemonSet的状态。

DaemonSet基本操作

更新DaemonSet

进入DaemonSet列表页面，点击**更新**，进入更新DaemonSet页面，用户可以根据业务需要更新DaemonSet的配置。

更新策略：

- 滚动更新：对实例进行逐个更新，这种方式可以让您不中断业务实现对服务的更新，您可自定义滚动更新的参数。
- Ondelete：更新模板后，只有手动删除了旧的pod才会创建新的pod。

删除DaemonSet

进入DaemonSet列表页面，点击**删除**，执行删除DaemonSet的操作。

Job管理

Job概述

Job负责批量处理短暂的一次性任务，即仅执行一次的任务，它保证批处理任务的一个或多个Pod成功结束。相较于Deployment，Statefulset这类正常状态下保持永远运行的任务，Job的主要区别在于成功完成用户设置的任务后会自动退出。因此Job更适合于数据处理、迁移等一次性任务处理场景。 本模块提供了基于Kubernetes原生的Job的创建、配置、删除等生命周期的管理指南。

创建Job

您可以通过以下步骤在容器服务控制台通过镜像创建一个Job：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建Job的集群，进入该集群操作页面。
4. 选择**工作负载** > **Job**，进入Job列表页。
5. 单击页面左上角**新建**，根据实际需求完成配置，单击**创建**，即可新建Job。

以下为创建过程中的配置详情：

基本信息配置

- 名称：用户自定义Job的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 部署集群：选择Job运行的集群。
- 命名空间：选择Job所在集群的命名空间。
- 描述：创建Job的相关信息，用户自定义填写。

部署配置

Job设置

根据实际需求，您可以设置Job的关键参数：

- 重复次数：设置Job结束所需pod成功运行的次数，默认为1。
- 并行度：设置Job下pod并行执行的数量，默认为1。
- 失败重启策略：设置pod的容器异常退出时是否重启。
  - Never：失败容器不会重启，直至所有容器全部退出，新的pod会被启动。
  - OnFailure：失败容器将会重新启动，pod继续运行。

存储卷

目前支持使用主机路径、本地路径、金山云云硬盘、文件存储、已有PVC等存储类型。

- 类型：包含主机路径、本地路径、金山云云硬盘、文件存储、已有PVC、使用ConfigMap/Secret。
- 存储卷名称：存储卷的名称。
- 资源名称：选择相应存储资源的名称。
- 其他信息：指定容器要挂载的路径。

备注：

使用本地硬盘时，若不指定源路径，则默认分配临时路径（对应k8s存储中EmpytDir）。

当使用云硬盘作为存储卷时，存储卷的名称为云硬盘的id，不可修改。

容器配置

设置运行的容器信息：

- 名称：容器的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 镜像：点击**选择镜像**，从镜像仓库中选择；或输入镜像仓库地址。
- 镜像版本：镜像的tag。
- 资源限制：设定容器所使用的CPU和内存资源的限制。
- 环境变量：在容器中添加环境变量，一般用于通过环境变量设置参数，目前仅支持手动添加。
- 实例数量：一个实例由相关的一个或多个容器构成，用户自定义实例数量。

镜像访问凭证

目前支持从私有镜像仓库拉取镜像时，为容器配置镜像访问凭证（对应yaml中的imagePullSecret）。

- 初始状态下提供默认选项ksyunregistrykey，可以匹配金山云镜像仓库中的私有镜像。
- 若您使用的镜像来自第三方私有镜像仓库，可以通过**添加镜像访问凭证** > **使用新的访问凭证** > **设置访问凭证信息**，设置新的访问凭证的名称，以及第三方仓库的域名、用户名、密码，即完成添加第三方私有镜像仓库访问凭证。

配置完成后，点击**创建**，返回Job列表页面，查询Job的状态。

Job基本操作

查看Job状态

在Job列表页面，即可查看当前命名空间下所有Job的运行状态。

- 实例个数（成功/全部）：这一指标反映该job当前运行进展，全部实例代表Job设置中的重复次数，成功个数代表当前已经成功退出的pod数量。

点击Job名称进入某个Job的Pod列表，在状态栏中可以看到各pod当前的运行状态。

点击Job名称进入某个Job的详情页，在详情中的基本信息部分，也可以看到当前Job下不同状态pod的数量统计，状态包含“成功”，“失败”，“运行中”三种。

删除

点击Job列表中的**删除**，执行删除Job的操作。 点击Job名称进入某个Job的详情页，在pod列表中点击**销毁实例**，即可删除某Job下的特定实例。

Kubectl操作示例

在Job列表页面，可以通过**YAML创建资源**新建Job，也可以通过Job列表中的**编辑YAML**更新已有Job的配置。 以下提供一个YAML示例，创建一个Job执行计算π到2000位并打印输出。

pi-job.yaml如下：

```
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  completions: 2
  parallelism: 2
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
  backoffLimit: 4
```

- spec.completions: Job结束所需pod成功运行的次数
- spec.parallelism: Job下pod并行执行的数量
- spec.template: Job管理的Pod的详细模板配置
- spec.backoffLimit: Job的容错次数，当达到这个值时不会再创建新的pod而会退出job，默认值为6

创建该job

```
# kubect apply -f pi-job.yaml
```

查看Job的状态

```
# kubectl get job
```

CronJob管理

CronJob概述

在Job仅执行一次任务的基础上，CronJob增加了时间调度，适合应用于在给定的时间点执行一个任务，或者周期性的在某时间点进行一个任务。

本模块提供了基于Kubernetes原生的CronJob的创建、配置、删除等生命周期的管理指南。

创建CronJob

您可以通过以下步骤在容器服务控制台通过镜像创建一个CronJob：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建CronJob的集群，进入该集群操作页面。
4. 选择**工作负载** > **CronJob**，进入CronJob列表页。
5. 单击页面左上角**新建**，根据实际需求完成配置，单击**创建**，即可新建CronJob。

以下为创建过程中的配置详情：

基本信息配置

- 名称：用户自定义CronJob的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 部署集群：选择CronJob运行的集群。
- 命名空间：选择CronJob所在集群的命名空间
- 描述：创建CronJob的相关信息，用户自定义填写。

部署配置

并发策略

在多个Job按照定时规则执行时，若新任务开始时旧任务还未完成，支持从以下三种模式中选择新旧任务的并发策略：

- Forbid: 旧任务未完成时禁止创建新任务。
- Allow: 允许新旧任务并发运行。
- Replace: 旧任务未完成时新任务替代正在运行的旧任务。

默认并发策略为“Allow”。

定时策略

指定新建定时任务在何时执行。根据5位Cron表达式输入定时策略，5位分别代表“分 时 日 月 周”，如“ /1 *”表示每小时执行一次。

```
# 5位cron表达式格式说明
#  ——分（0 - 59）
# |  ——时（0 - 23）
# | |  ——日（1 - 31）
# | | |  ——月（1 - 12）
# | | | |  ——周 (0 - 7) (Sunday=0 or 7)
# | | | | |
# * * * * *
```

Job设置

根据实际需求，您可以设置Cronjob中每次执行Job的关键参数：

- 重复次数：设置Job结束所需pod成功运行的次数，默认为1。
- 并行度：设置Job下pod并行执行的数量，默认为1。
- 失败重启策略：设置pod的容器异常退出时是否重启。
  - Never:失败容器不会重启，直至所有容器全部退出，新的pod会被启动。
  - OnFailure:失败容器将会重新启动，pod继续运行。

存储卷

目前支持使用主机路径、本地路径、金山云云硬盘、文件存储、已有PVC等存储类型。

- 类型：包含主机路径、本地路径、金山云云硬盘、文件存储、已有PVC、使用ConfigMap/Secret。
- 存储卷名称：存储卷的名称。
- 资源名称：选择相应存储资源的名称。
- 其他信息：指定容器要挂载的路径。

备注：

使用本地硬盘时，若不指定源路径，则默认分配临时路径（对应k8s存储中EmpytDir）。

当使用云硬盘作为存储卷时，存储卷的名称为云硬盘的id，不可修改。

容器配置

设置运行的容器信息：

- 名称：容器的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- 镜像：点击**选择镜像**，从镜像仓库中选择；或输入镜像仓库地址。
- 镜像版本：镜像的tag。
- 资源限制：设定容器所使用的CPU和内存资源的限制。
- 环境变量：在容器中添加环境变量，一般用于通过环境变量设置参数，目前仅支持手动添加。
- 实例数量：一个实例由相关的一个或多个容器构成，用户自定义实例数量。

镜像访问凭证

目前支持从私有镜像仓库拉取镜像时，为容器配置镜像访问凭证（对应yaml中的imagePullSecret）。

- 初始状态下提供默认选项ksyunregistrykey，可以匹配金山云镜像仓库中的私有镜像。
- 若您使用的镜像来自第三方私有镜像仓库，可以通过**添加镜像访问凭证** > **使用新的访问凭证** > **设置访问凭证信息**，设置新的访问凭证的名称，以及第三方仓库的域名、用户名、密码，即完成添加第三方私有镜像仓库访问凭证。

配置完成后，点击**创建**，返回CronJob列表页面，查询CronJob的状态。

CronJob基本操作

运行/停止Cronjob

创建Cronjob后，若您想要运行/停止某个Cronjob，可以在Cronjob列表中对该Cronjob执行**运行**或**停止**操作。

查看Cronjob状态

点击Cronjob列表中某Cronjob名称，可至Cronjob任务列表页查看Cronjob中已创建的Job的运行状态。点击任务列表中的任务名称可进入对应Job的详情页。

删除

点击CronJob列表中的**删除**，执行删除CronJob的操作。 若需要删除Cronjob中某单个Job，可进入该Cronjob任务列表点击**删除**，实现该Job相关资源的删除。

Kubectl操作示例

在CronJob列表页面，可以通过**YAML创建资源**新建CronJob，也可以通过CronJob列表中的**编辑YAML**更新已有CronJob的配置。

方法一：

以下提供一个YAML示例，创建一个每分钟输出问候语的定时任务。 hello-cronjob.yaml如下：

```
apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: hello
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox
            args:
            - /bin/sh
            - -c
            - date; echo Hello from the Kubernetes cluster
          restartPolicy: OnFailure
```

- spec.schedule: 定时策略
- spec.jobTemplate: 定义Cronjob执行的Job对象的模板

创建Cronjob

```
# kubectl apply -f hello-cronjob.yaml
```

方法二：

通过Kubectl run快速创建一个不需完整配置信息的Cronjob，命令如下：

```
kubectl run hello --schedule="*/1 * * * *" --restart=OnFailure --image=busybox -- /bin/sh -c "date; echo Hello"
```

查看CronJob的状态

```
kubectl get cronjob
```

设置容器运行命令

在默认的情况下，镜像会运行默认的命令，如果我们想运行一个特定的命令或重写镜像的默认值，这里需要使用到以下三个设置：

- 工作目录（workingDir）：指定运行命令的工作目录。
- 运行命令（command）：控制镜像运行的实际命令。
- 运行参数（args）：传递给运行命令的参数。

工作目录说明

指定运行命令的工作目录。若镜像中未指定工作目录，且在控制台未指定，则默认为"/"。

设置容器启动时运行命令和参数

Docker 的镜像拥有存储镜像信息的相关元数据，如果不提供运行命令和参数，容器运行会运行镜像制作时提供的默认的命令和参数，Docker 原生定义这两个字段为 “ Entrypoint ” 和 " CMD "。详情可查看 Docker的 Entrypoint 和 CMD说明。

如果要覆盖默认的Entrypoint 与 CMD，需要遵循如下规则：

- 如果在容器配置中没有设置command 或者 args，那么将使用Docker镜像自带的命令及参数。
- 如果在容器配置中只设置了command但是没有设置args,那么容器启动时只会执行该命令，Docker镜像中自带的命令及其参数会被忽略。
- 如果在容器配置中只设置了args,那么Docker镜像中自带的命令会使用该新入参作为其执行时的参数。
- 如果在容器配置中同时设置了command 与 args，那么Docker镜像中自带的命令及其入参会被忽略。容器启动时只会执行配置中设置的命令，并使用配置中设置的入参作为命令的参数。

下表涵盖了各类设置场景：

镜像Entrypoint	镜像CMD	容器command	容器args	最终执行
[touch]	[/usr/test]	未设置	未设置	[touch /usr/test]
[touch]	[/usr/test]	[mkdir]	未设置	[mkdir]
[touch]	[/usr/test]	未设置	[opt/test]	[touch /opt/test]
[touch]	[/usr/test]	[mkdir]	[opt/test]	[mkdir /opt/test]

容器健康检查

对于Pod的健康检查，可以通过两类探针来检查： LivenessProbe（存活探针）和ReadinessProbe（就绪探针）。

- LivenessProbe（存活探针）：用于判断容器是否存活。如果LivenessProbe探针探测到容器不健康，则kubelet将会杀掉该容器，并根据容器的重启策略做相应的处理；若探测到容器健康，则不做任何处理。
- ReadinessProbe（就绪探针）：用于判断容器是否已经就绪可以接收流量。一些程序的启动时间可能很长，比如要加载磁盘数据或者要依赖外部的某个模块启动完成才能提供服务。这时候程序进程在，但是并不能对外提供服务。如果ReadinessProbe检测通过，则允许服务将流量发送到该容器；如果ReadinessProbe检测到失败，则集群将会停止向该容器发送流量。

健康检查方式

- **TCP端口检查**

  对于提供 TCP 通信服务的容器，集群周期性地对该容器建立 TCP 连接，如果连接成功，则证明探测成功，否则探测失败。选择 TCP 端口探测方式，必须指定容器监听的端口。比如我们有一个 nginx 容器，它的服务端口是 80，我们对该容器配置了 TCP 端口探测，指定探测端口为 80，那么集群会周期性地对该容器的 80 端口发起 TCP 连接，如果连接成功则证明检查成功，否则检查失败。

- **HTTP请求检查**

  HTTP 请求探测针对的是提供 HTTP/HTTPS 服务的容器，通过容器的IP地址、端口号以及路径，集群周期性的对该容器发起HTTP/HTTPS Get的请求，如果相应的状态码大于200且小于400，则认为容器状态健康否则探测异常。 例如：提供 HTTP 服务的容器，服务端口为 80，HTTP 检查路径为 /health，那么集群会周期性地对容器发起如下请求： `GET http://containerIP:80/health`。

- **执行命令检查**

  执行命令检查是一种强大的检查方式，该方式要求用户指定一个容器内的可执行命令，集群会周期性地在容器内执行该命令，如果命令的返回结果是 0 则检查成功，否则检查失败。

  对于上面提到的 TCP 端口检查和 HTTP 请求检查，都可以通过执行命令检查的方式来替代：

  - 对于 TCP 端口探测，我们可以写一个程序来对容器的端口进行 connect，如果 connect 成功，脚本返回 0，否则返回 -1。
  - 对于 HTTP 请求探测，我们可以写一个脚本来对容器进行 wget。 `wget http://containerIP:80/health` 并检查 response 的返回码，如果返回码在 200~399 的范围，脚本返回 0，否则返回 -1。

  备注：

  - 必须把要执行的程序放在容器的镜像里面，否则会因找不到程序而执行失败。
  - 如果执行的命令是一个 shell 脚本，由于集群在执行容器里的程序时，不在终端环境下，因此不能直接指定脚本为执行命令，需要加上脚本解释器。比如脚本是 /data/scripts/health.sh，那么我们使用执行命令检查时，指定的程序应该是 sh /data/scripts/health.sh。究其原因是集群在执行容器里的程序时，不在终端环境下。

配置Probe

Probe 中有很多精确和详细的配置，通过它们你能准确的控制liveness和readiness检查：

- **启动延迟**：initialDelaySeconds，单位秒。该参数用于设置容器启动后多久开始启动探测。例如设置启动延迟为30s，那么健康检查将在容器启动后30s后开始。
- **间隔时间**：periodSeconds，单位秒。该参数用于设置健康检查的频率。例如设置间隔时间为10s，则每10s检查一次。
- **响应超时**：timeoutSeconds，单位秒。该参数用于设置检查探测的超时时间，对应到 TCP 端口探测、HTTP 请求探测、执行命令检查三种方式，分别表示 TCP 连接超时时间、HTTP 请求响应超时时间以及执行命令的超时时间。
- **健康阈值**：successThreshold，单位次。该参数用来设定健康检查连续成功多少次后，才判定容器是健康的。例如健康阈值设置成 3，则说明只有满足连续 3 次探测都成功才认为容器是健康的。
- **不健康阈值**：failureThreshold，单位次。该参数用来设定健康检查连续失败多少次后，才判定容器是不健康的。例如不健康阈值设置成 3，只有满足连续 3 次都探测失败了，才认为容器是不健康的。

备注： 对于容器的存活检查（Liveness），健康阈值（successThreshold）只能是1，用户设置成其它值将被视为无效。因为只要探测成功一次，我们就能确定容器是存活的。

设置访问方式

金山云容器服务提供了五种访问方式：公网访问，VPC内访问，集群内访问，主机端口访问和不设置。用户可以根据业务的实际情况选择对应的访问方式。

公网访问

通过金山云负载均衡将服务暴露到公网，可以直接被公网访问。 创建完成后，可以通过**公网LB的IP+服务端口**直接访问。

备注：

1、系统会自动创建公网LB和EIP将服务暴露到公网，并实现监听器的动态监听和挂载。

2、目前EIP的外网访问带宽默认是1M，用户可以根据业务的需要调整。

VPC内网访问

通过金山云负载均衡将服务暴露在集群所在VPC内，可以被VPC下的其他资源或者集群访问。 创建完成后，可以通过**内网LB的IP+服务端口**直接访问。

集群内访问

将服务暴露到集群内部，仅限集群内访问。 创建完成后，可以通过**服务IP或服务名称 + 服务端口**直接访问。

主机端口访问

将服务暴露到集群外部，可以通过节点上开放的特定端口访问集群内服务。 创建完成后，可以通过**主机IP+端口**直接访问。

不设置

不提供任何从前端服务访问到容器的入口，可用于使用自定义的服务发现或简单启用多个容器实例。

使用限制

1、请不要修改集群自动创建的负载均衡的名称，否则可能会出现容器集群无法识别已有的负载均衡而重新创建的情况。

2、对于集群自动创建的EIP、负载均衡等资源，请不要更改资源所属的项目。

调度策略概述

简介

通过配置工作负载中高级设置的调度规则，您可限定 Pod 只在特定的节点上运行，或者优先选择在特定的节点上运行，实现指定工作负载下的Pod在集群内灵活调度的能力。 在调度策略配置中，您可根据实际需求设置节点选择、节点亲和、污点容忍、Pod亲和、Pod反亲和，详细信息请见Assigning Pods to Nodes。

调度策略操作

- 节点选择
- 节点亲和
- 污点容忍
- Pod亲和
- Pod反亲和

> 注：调度策略依赖于标签和Pod标签，您可根据实际需求预先为节点、Pod配置相关的标签。

节点选择

基于节点的标签，您可以限定Pod可以被调度到哪些节点上。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要操作的集群，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。

> 注：您可根据实际需求选择Deployment、StatefulSet、DaemonSet。

5. 单击页面左上角**新建**，进入创建流程第二步**部署配置**。
6. 点击高级配置，选择**添加NodeSelector**。
7. 在节点选择设置中，依据节点的标签进行业务需求的设置，单击**确定**，即可创建该策略。

节点亲和

节点亲和

节点亲和的概念与节点选择相似，基于节点的标签来限定 Pod 可以被调度到哪些节点上。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要操作的集群，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。

> 注：您可根据实际需求选择Deployment、StatefulSet、DaemonSet。

5. 单击页面左上角**新建**，进入创建流程第二步**部署配置**。
6. 点击高级配置，选择**添加 NodeAffinity**。
7. 在节点亲和设置中，依据节点的标签进行业务需求的设置，单击**确定**，即可创建该策略。

以下为节点亲和设置详情：

支持必须满足和尽量满足（硬约束Required/软约束Preferred）

参数	描述
必须满足	即硬约束，目标节点必须满足此条件，对应requiredDuringSchedulingIgnoredDuringExecution，您可以点击**新增调度规则**添加多条必须满足的规则，多个调度规则是逻辑或的关系，即只需要满足一条规则即可被调度。
尽量满足	即软约束，目标节点最好能满足此条件，对应preferredDuringSchedulingIgnoredDuringExecution，调度会尽量调度Pod到具有对应标签的节点上。您可以为规则设定权重，具体调度时，若存在多个符合条件的节点，权重最大的节点会被优先调度。同时您可点击**新增调度规则**添加多条尽量满足的规则，无论是满足其中一条或者是都不满足都会进行调度。

新增选择器

对应matchExpressions，您可以单击**新增选择器**添加多条选择器，多个选择器是逻辑与的关系，即需要满足全部选择器才能依据此条规则进行调度。

操作符

参数	描述
In	表示标签值需要在values的列表中
NotIn	标签的值不在某个列表中
Gt	标签的值大于某个值
Lt	标签的值小于某个值
Exists	某个标签存在
DoesNotExist	某个标签不存在

污点容忍

污点容忍

节点选择、节点亲和调度策略操作用于将Pod指定到合适的节点，相对的，节点中存在属性污点，使得节点可以排斥某些Pod。您可以为Pod添加容忍，使得该Pod能否容忍节点上的污点，调度到该节点

上。污点和容忍相互作用，可以确保 Pod 不会被调度到不合适的节点上。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要操作的集群，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。

   注：您可根据实际需求选择Deployment、StatefulSet、DaemonSet。

5. 单击页面左上角**新建**，进入创建流程第二步**部署配置**。
6. 点击高级配置，选择**添加PodToleration**。
7. 在污点容忍设置中，根据实际需求进行设置，单击**确定**，即可创建该策略。

以下为污点容忍设置详情：

参数	描述
容忍名	对应污点的key，如果该值为空，则匹配所有的污点，此刻操作符必须为Exists
操作符	表示 key 与 value 的关系，可选值为 Exists 和 Equal
容忍值	对应污点的 value
效果	可选值为 NoSchedule、PreferNoSchedule、NoExecute
时间	容忍的持续时间。默认情况下，该字段为空，代表 Pod 可以一直容忍该污点（不会被驱逐），若填写零或负数默则认定为立即驱逐

新增调度规则

您可以点击**新增调度规则**添加多条规则，多个调度规则是逻辑或的关系，即只需要满足一条规则即可被调度。

Pod亲和

Pod亲和

Pod亲和策略可以决定工作负载的Pod和哪些Pod部署在同一个拓扑域中，同时也可以基于已经运行在节点上的 Pod 的标签来限定 Pod 可以被调度到哪个节点上。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要操作的集群，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。

   注：您可根据实际需求选择Deployment、StatefulSet、DaemonSet。

5. 单击页面左上角**新建**，进入创建流程第二步**部署配置**。
6. 点击高级配置，选择**添加 PodAffinity**。
7. 在Pod亲和设置中，依据Pod的标签进行业务需求的设置，单击**确定**，即可创建该策略。

以下为Pod亲和设置详情：

支持必须满足和尽量满足（硬约束Required/软约束Preferred）

参数	描述
必须满足	即硬约束，Pod的亲和性调度必须要满足后续定义的约束条件，对应requiredDuringSchedulingIgnoredDuringExecution，您可以点击**新增调度规则**添加多条必须满足的规则，多个调度规则是逻辑或的关系，即只需要满足一条规则即可被调度。
尽量满足	即软约束，对应preferredDuringSchedulingIgnoredDuringExecution，Pod的亲和性调度会尽量满足后续定义的约束条件。同时您可点击**新增调度规则**添加多条尽量满足的规则，无论是满足其中一条或者是都不满足都会进行调度。

新增选择器

对应matchExpressions，您可以单击**新增选择器**添加多条选择器，多个选择器是逻辑与的关系，即需要满足全部选择器才能依据此条规则进行调度。

可设置参数

参数	描述
权重	**尽量满足**时可为规则设置权重值，权重值越高会被优先调度。
命名空间	依据Pod的标签进行调度，所以会受到命名空间的约束
拓扑域	即topologyKey，用于指定调度时的作用域。拓扑域通过设置Node节点的标签，例如指定为 kubernetes.io/hostname，以Node节点作为区分范围
查看应用列表	根据实际需求查看各命名空间下的应用，将可用的标签导入到亲和性配置中
标签名	对应工作负载的Pod标签，您可以手工填写，也可从应用列表中选择
操作符	可选值为In，NotIn，Exists，DoesNotExist
标签值	当操作符选择In或NotIn时，可以添加单个或多个value值；选择Exists或DoesNotExist时不需设置value值，即判断某个label是否存在

Pod反亲和

Pod反亲和

Pod反亲和策略可以决定工作负载的Pod不和哪些Pod部署在同一个拓扑域中，同时也可以基于已经运行在节点上的 Pod 的标签来限定 Pod 可以不被调度到哪个节点上。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要操作的集群，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。

   注：您可根据实际需求选择Deployment、StatefulSet、DaemonSet。

5. 单击页面左上角**新建**，进入创建流程第二步**部署配置**。
6. 点击高级配置，选择**添加 PodAntiAffinity**。
7. 在Pod反亲和设置中，依据Pod的标签进行业务需求的设置，单击**确定**，即可创建该策略。

以下为Pod反亲和设置详情：

支持必须满足和尽量满足（硬约束Required/软约束Preferred）

参数	描述
必须满足	即硬约束，Pod的反亲和性调度必须要满足后续定义的约束条件，对应requiredDuringSchedulingIgnoredDuringExecution，您可以点击**新增调度规则**添加多条必须满足的规则，多个调度规则是逻辑或的关系，即只需要满足一条规则即可
尽量满足	即软约束，对应preferredDuringSchedulingIgnoredDuringExecution，Pod的反亲和性调度会尽量满足后续定义的约束条件。同时您可点击**新增调度规则**添加多条尽量满足的规则，无论是满足其中一条或者是都不满足都会进行调度

新增选择器

对应matchExpressions，您可以单击**新增选择器**添加多条选择器，多个选择器是逻辑与的关系，即需要满足全部选择器才能依据此条规则进行调度。

可设置参数

参数	描述
权重	**尽量满足**时可为规则设置权重值，权重值越高会被优先调度
命名空间	依据Pod的标签进行调度，所以会受到命名空间的约束
拓扑域	即topologyKey，用于指定调度时的作用域。拓扑域通过设置Node节点的标签，例如指定为kubernetes.io/hostname，以Node节点作为区分范围
查看应用列表	根据实际需求查看各命名空间下的应用，将可用的标签导入到亲和性配置中
标签名	对应工作负载的Pod标签，您可以手工填写，也可从应用列表中选择
操作符	可选值为In，NotIn，Exists，DoesNotExist
标签值	当操作符选择In或NotIn时，可以添加单个或多个value值；选择Exists或DoesNotExist时不需设置value值，即判断某个label是否存在

Pod弹性伸缩

Pod弹性伸缩（Horizontal Pod Autoscaling，简称HPA）是Kubernetes中实现POD水平自动伸缩的功能，可以根据CPU使用率或者其他自定义的指标自动扩展部署中的Pod数量。

自动伸缩算法

HPA组件会每隔30s从集群中的Metrics Server组件获取pod的监控指标（如果设置了目标利用率，则需要计算监控指标与每个pod中容器的resource request的百分比；如果设置了目标原始值，则直接使用该原始metrics值），根据工作负载当前副本数和该指标的目标值计算出期望副本数，作为工作负载的期望副本数，具体计算公式如下：

扩缩容算法：期望副本数量=采集的使用率/用户自定义使用率*当前pod数量

支持设置多个弹性伸缩的策略，HPA会根据每个指标的目标值，分别计算出目标副本数，然后取最大的一个作为最终目标副本数。

容器服务控制台操作说明

新建HPA

可以通过以下三种方式新建HPA。

方式一：通过单击新建HPA

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建HPA的集群ID，进入该集群操作页面。
4. 选择**自动伸缩** > **HPA**，进入HPA列表页。
5. 单击**新建**，在新建HPA页面，根据以下提示进行HPA配置：
   - 名称：输入要创建HPA的名称。
   - 命名空间：请根据实际需求进行选择。
   - 关联Deployment：请根据实际需求进行选择。
   - 触发策略：HPA功能依赖的策略指标。
   - 实例范围：请根据实际需求进行选择，实例数量会在设定的范围内自动调节，不会超出该设定范围。

   注 当前支持设置的触发策略指标：

   - CPU利用率：容器CPU使用量和CPU request值的比率
   - CPU使用量：容器CPU使用量（核）
   - 内存使用率：容器内存的使用量和内存 request值的比率
   - 内存使用量：容器内存使用量（MiB）
6. 单击**创建**，完成HPA创建。

方式二：通过创建部署时，设置pod的弹性伸缩

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建HPA的集群ID，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。
5. 单击**新建**，在创建Deployment页面，第二步**部署配置**流程中选择**自动调节**并根据以下提示进行设置：
   - 触发策略：HPA功能依赖的策略指标。
   - 实例范围：请根据实际需求进行选择，实例数量会在设定的范围内自动调节，不会超出该设定范围。

方式三：通过YAML创建

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建HPA的集群ID，进入该集群操作页面。
4. 选择**自动伸缩** > **HPA**，进入HPA列表页。
5. 单击页面右上角**YAML创建资源**，在YAML创建资源页面，根据实际需求编辑内容，单击**创建**，即可新建HPA。

调整HPA配置

可以通过以下三种方式调整HPA配置。

方式一：通过单击调整配置修改

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要调整HPA配置的集群ID，进入该集群操作页面。
4. 选择**自动伸缩** > **HPA**，进入HPA列表页。
5. 单击**调整配置**，在出现的弹窗中，根据实际需求进行调整，并单击**确定**，即可调整HPA配置。

方式二：通过编辑YAML更新

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要调整HPA配置的集群ID，进入该集群操作页面。
4. 选择**自动伸缩** > **HPA**，进入HPA列表页。
5. 单击**编辑YAML**，在更新YAML页面，根据实际需求进行调整，并单击**确定**，即可调整HPA配置。

方式三：通过调节实例数量时，调整pod的弹性伸缩

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要调整HPA配置的集群ID，进入该集群操作页面。
4. 选择**工作负载** > **Deployment**，进入Deployment列表页。
5. 单击**调节实例数量**，在出现的弹窗中，选择**自动调节**并根据实际需求进行调整，单击**确定**即可调整HPA设置。

查看HPA相关信息

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要查看HPA相关信息的集群ID，进入该集群操作页面。
4. 选择**自动伸缩** > **HPA**，进入HPA列表页。
5. 单击需要查看HPA信息的名称。
6. 进入HPA详情页，可以选择**详情**、**事件**、**YAML**进行相关信息查看。

删除HPA

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要删除HPA的集群ID，进入该集群操作页面。
4. 选择**自动伸缩** > **HPA**，进入HPA列表页。
5. 单击**删除**，在出现的弹窗中，点击**确认**即可删除HPA。

Kubectl 命令操作说明

可以通过 YAML 文件创建和编辑 HPA 。以下为配置文件的示例：

hpa-example.yaml示例：

```
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-example
  namespace: default
spec:
  minReplicas: 1                    #最小副本数
  maxReplicas: 3                    #最大副本数
  metrics:
  - type: Resource
    resource:
      name: cpu
      targetAverageUtilization: 50       #当CPU使用率为50时触发HPA，实例范围为1-3
  scaleTargetRef:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: nginx
```

创建hpa-example.yaml

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

注意事项

- 当使用cpu/内存使用率作为伸缩策略指标时，请务必设置容器request的值，否则不支持
- HPA在计算目标副本数时会有一个 10% 的波动因子，如果在波动范围内，HPA 并不会调整副本数目。
- HPA在每一次作出决策后的一段时间内，将不再进行扩展决策。对于扩容而言，这个时间段为3分钟，缩容为5分钟
- 保证用户请求的负载均衡

Pod定时伸缩

Pod定时伸缩，即CronHPA是Kubernetes中Pod水平扩展功能的一种，Kubernetes中的HPA是基于监控度量来定义并驱动Pod水平扩展缩容的，而CronHPA是根据crontab的方式，定期性的对Pod水平进行扩展缩容操作。
CronHPA支持任意规模类型资源的扩展缩容操作，其中包括Deployment和StatefulSet等。

使用场景

通过运维经验积累掌握负载规律后，可以使用CronHPA实现定期自动扩展缩容操作，以应对可预期的业务需求。

使用说明

前提条件

您已使用金山云容器服务服务创建一个正常运行的Kubernetes集群，关于如何创建集群，请参见创建集群。

CronHPA插件安装

- 进入容器集群内任意节点，创建cron-hpa.yaml

```
$ kubectl apply -f cron-hpa.yaml
```

- cron-hpa.yaml如下：

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  creationTimestamp: null
  labels:
    controller-tools.k8s.io: "1.0"
  name: cronhorizontalpodautoscalers.autoscaling.kce.ksyun.com
spec:
  group: autoscaling.kce.ksyun.com
  names:
    kind: CronHorizontalPodAutoscaler
    plural: cronhorizontalpodautoscalers
    shortNames:
      - cronhpa
  scope: Namespaced
  validation:
    openAPIV3Schema:
      properties:
        apiVersion:
          description: 'APIVersion defines the versioned schema of this representation
            of an object. Servers should convert recognized schemas to the latest
            internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/api-conventions.md#resources'
          type: string
        kind:
          description: 'Kind is a string value representing the REST resource this
            object represents. Servers may infer this from the endpoint the client
            submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/api-conventions.md#types-kinds'
          type: string
        metadata:
          type: object
        spec:
          properties:
            jobs:
              items:
                properties:
                  name:
                    type: string
                  schedule:
                    type: string
                  targetSize:
                    format: int32
                    type: integer
                required:
                - name
                - schedule
                - targetSize
                type: object
              type: array
            scaleTargetRef:
              description: 'INSERT ADDITIONAL SPEC FIELDS - desired state of cluster
                Important: Run "make" to regenerate code after modifying this file'
              properties:
                apiVersion:
                  type: string
                kind:
```

```
                type: string
              name:
                type: string
            required:
            - apiVersion
            - kind
            - name
            type: object
        required:
        - scaleTargetRef
        - jobs
        type: object
      status:
        properties:
          conditions:
            description: 'INSERT ADDITIONAL STATUS FIELD - define observed state
              of cluster Important: Run "make" to regenerate code after modifying
              this file'
            items:
              properties:
                jobId:
                  type: string
                lastProbeTime:
                  format: date-time
                  type: string
                message:
                  description: Human readable message indicating details about last
                    transition.
                  type: string
                name:
                  description: Type of job condition, Complete or Failed.
                  type: string
                schedule:
                  type: string
                state:
                  type: string
              required:
              - name
              - jobId
              - schedule
              - state
              - lastProbeTime
              - message
              type: object
            type: array
        required:
        - conditions
        type: object
  version: v1beta1
status:
  acceptedNames:
    kind: ""
    plural: ""
  conditions: []
  storedVersions: []
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kubernetes-cronhpa-controller-role
rules:
- apiGroups: [""]
  resources: ["*"]
  verbs:
  - get
  - list
  - watch
  - update
- apiGroups:
  - extensions
  resources: ["*"]
  verbs:
  - get
  - list
  - watch
  - update
- apiGroups:
  - apps
  resources: ["*"]
  verbs:
  - get
  - list
  - watch
  - update
- apiGroups: [""]
  resources: ["events"]
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
- apiGroups: [""]
  resources: ["configmaps"]
  verbs:
    - get
    - create
    - update
- apiGroups:
  - autoscaling.kce.ksyun.com
  resources:
  - cronhorizontalpodautoscalers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
  - delete
- apiGroups:
  - admissionregistration.k8s.io
  resources:
  - mutatingwebhookconfigurations
  - validatingwebhookconfigurations
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - patch
```

```
  - delete
- apiGroups:
  - apps.kruise.io
  resources:
  - statefulsets/scale
  - unitedeployments/scale
  - clonesets/scale
  verbs:
  - get
  - list
  - watch
  - update
  - patch
- apiGroups:
  - autoscaling.kce.ksyun.com
  resources:
  - elasticworkloads/scale
  verbs:
  - get
  - list
  - watch
  - update
  - patch
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: kubernetes-cronhpa-controller
  namespace: kube-system
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  creationTimestamp: null
  name: kubernetes-cronhpa-controller-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kubernetes-cronhpa-controller-role
subjects:
- kind: ServiceAccount
  name: kubernetes-cronhpa-controller
  namespace: kube-system
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubernetes-cronhpa-controller
  namespace: kube-system
  labels:
    app: kubernetes-cronhpa-controller
    controller-tools.k8s.io: "2.0"
spec:
  replicas: 1 # The default is primary and standby mode (currently cold standby)
  selector:
    matchLabels:
      app: kubernetes-cronhpa-controller
      controller-tools.k8s.io: "2.0"
  template:
    metadata:
      labels:
        app: kubernetes-cronhpa-controller
        controller-tools.k8s.io: "2.0"
    spec:
      containers:
      - image: hub.kce.ksyun.com/ksyun/kubernetes-cronhpa-controller:v1.0.0
        imagePullPolicy: Always
        name: kubernetes-cronhpa-controller
        env:
        - name: TZ
          value: "Asia/Shanghai"
        resources:
          limits:
            cpu: 100m
            memory: 100Mi
          requests:
            cpu: 100m
            memory: 100Mi
      serviceAccount: kubernetes-cronhpa-controller
```

- 查看插件是否安装成功

```
# 查看自定义资源是否安装
$ kubectl api-resources | grep cronhpa
cronhorizontalpodautoscalers      cronhpa       autoscaling.kce.ksyun.com      true         CronHorizontalPodAutoscaler

# 查看CronHPA controller是否安装
$ kubectl get deploy kubernetes-cronhpa-controller -n kube-system
NAME                            READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-cronhpa-controller   1/1     1            1           4m11s
```

功能测试

成功安装CronHPA插件后，以下举例如何部署具体的cronhpa并进行测试。

- 部署测试对象与cronhpa。

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

- cronhpa-example.yaml如下：

```
---
apiVersion: apps/v1 # for versions before 1.8.0 use apps/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment-basic
  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

---
apiVersion: autoscaling.kce.ksyun.com/v1beta1
kind: CronHorizontalPodAutoscaler
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: cronhpa-sample
spec:
```

```
  scaleTargetRef:
    apiVersion: apps/v1beta2
    kind: Deployment
    name: nginx-deployment-basic
  excludeDates:
  - "* * * 15 11 *"                       # exclude November 15th
  - "* * * * * 5"                         # exclude every Friday
  jobs:
  - name: "scale-down"
    schedule: "30 */1 * * * *"            # 每分钟的第30秒，将deployment/nginx-deployment-basic 副本数调整为 1
    targetSize: 1
  - name: "scale-up"
    schedule: "0 */1 * * * *"             # 每分钟的第0秒，将deployment/nginx-deployment-basic 副本数调整为 3
    targetSize: 3
```

- 各字段的解释说明如下：

字段	说明	备注
scaleTargetRef	scaleTargetRef指定去扩大缩容对象。如果对象支持scale子资源，cronhpa即可支持	除scale子资源外，还支持HPA，以稳定性优先原则，在扩缩容时，以cronHPA与HPA设定副本数中较大值作为扩缩容最终值
excludeDates	excludeDates 是日期数组。当遇到符合excludeDates 描述的日期时任务将会被跳过	最小单位为天
jobs	支持在一个规格中设置多个cronhpa任务。每个cronhpa任务可以配置以下分段： - name：它在一个cronhpa中应是唯一的，这样就可以通过name 来区分不同的任务 - schedule：它的策略和crontab类似 - targetSize：到计划时间时，您想扩缩容到的Pod的数量 - runOnce：如果runOnce 设置为true ，那么任务将只执行一次，第一次执行完则exit	cron表达式格式： Field name Mandatory Allowed values Allowed special characters Seconds Yes 0-59 * / , - Minutes Yes 0-59 * / , - Hours Yes 0-23 * / , - Day of month Yes 1-31 * / , - ? Month Yes 1-12 or JAN-DEC * / , - Day of week Yes 0-6 or SUN-SAT * / , - ?

- 查看 cronhpa 实例

```
$ kubectl get cronhpa
NAME             AGE
cronhpa-sample   15s
```

- 查看cronhpa/cronhpa-sample详情

```
$ kubectl describe cronhpa cronhpa-sample
Events:
  Type    Reason  Age             From                                Message
  ----    ------  ----            ----                                -------
  Normal  Succeed  8m31s             cron-horizontal-pod-autoscaler  cron hpa job scale-down executed successfully. current replicas:2, desired replicas:1.
  Normal  Succeed  31s (x8 over 7m31s)  cron-horizontal-pod-autoscaler  cron hpa job scale-down executed successfully. current replicas:3, desired replicas:1.
  Normal  Succeed  1s (x9 over 8m1s)    cron-horizontal-pod-autoscaler  cron hpa job scale-up executed successfully. current replicas:1, desired replicas:3.
```

- 查看 deploy/nginx-deployment-basic 副本数状态变化情况

```
$ kubectl get deploy nginx-deployment-basic  -w
NAME                     READY   UP-TO-DATE   AVAILABLE      AGE
nginx-deployment-basic   1/1     1            1              3m31s
nginx-deployment-basic   1/3     1            1              3m48s
nginx-deployment-basic   1/3     1            1              3m48s
nginx-deployment-basic   1/3     3            1              3m48s
nginx-deployment-basic   2/3     3            2              3m50s
nginx-deployment-basic   3/3     3            3              3m50s
nginx-deployment-basic   3/1     3            3              4m18s
nginx-deployment-basic   3/1     3            3              4m18s
nginx-deployment-basic   1/1     1            1              4m18s
nginx-deployment-basic   1/3     1            1              4m48s
nginx-deployment-basic   1/3     1            1              4m48s
nginx-deployment-basic   1/3     1            1              4m48s
nginx-deployment-basic   1/3     3            1              4m48s
nginx-deployment-basic   2/3     3            2              4m49s
nginx-deployment-basic   3/3     3            3              4m49s
```

Service管理

Service概述

Kubernetes Service资源抽象了访问一组Pod的策略，屏蔽了后端实例的动态变化和对多实例的负载均衡，用于管理集群中基于四层网络的服务访问。Service主要包含以下几种类型：

- ClusterIP： 默认方式。根据是否生成ClusterIP又可分为普通Service和Headless Service两类：
  - 普通Service：通过为Kubernetes的Service分配一个集群内部可访问的固定虚拟IP（Cluster IP），实现集群内的访问，为最常见的方式。
  - Headless Service：该服务不会分配Cluster IP，也不通过kube-proxy做反向代理和负载均衡。而是通过DNS提供稳定的网络ID来访问，DNS会将headless service的后端直接解析为podIP列表。主要供StatefulSet使用。
- NodePort：除了使用Cluster IP之外，还通过将service的port映射到集群内每个节点的相同一个端口，实现通过nodeIP:nodePort从集群外访问服务。
- LoadBalancer：和NodePort类似，不过除了使用一个Cluster IP和NodePort之外，还会使用金山云的负载均衡器(负载均衡器后端映射到各节点的NodePort)，实现从集群外通过LB访问服务。

创建Service

通过控制台创建Service有多种实现方式，可以在创建负载时直接设置访问方式，也可以在创建Service时关联工作负载。这里主要介绍新建Service的操作步骤：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要新建Service的集群ID，进入该集群操作页面。
4. 选择**服务管理** > **Service**，进入Service管理页面，点击**新建**进入Service创建流程。
5. 设置Service基本信息：名称、所在命名空间等。
6. 设置访问方式：金山云容器服务提供四种访问方式，详细配置方式将在后文进行说明。
7. 关联工作负载：可通过手动设置selector，或引用workload设置两种方式选择Service关联的工作负载。
8. 点击**创建**，即可完成Service创建。

访问方式	Service类型	说明
公网访问	LoadBalancer	- 通过金山云负载均衡将服务暴露到公网，可以直接被公网访问。容器集群自动创建一个公网负载均衡和一个公网IP，执行监听器的动态挂载和同步。 - 创建完成的服务可以在集群外通过负载均衡IP+服务端口访问，详细配置参考通过金山云负载均衡访问服务。
VPC内访问	LoadBalancer	- 通过金山云负载均衡将服务暴露在集群所在VPC内，可以被VPC下的其他资源或者集群访问。容器集群会自动创建一个内网负载均衡，执行监听器的动态挂载和同步。 - 创建完成的服务可以在VPC内通过负载均衡IP+服务端口访问，详细配置参考通过金山云负载均衡访问服务。
集群内访问	ClusterIP	- 将服务暴露到集群内部，可以被集群内的其他服务或者容器访问。 - 支持创建时选择Headless Service。 - 创建完成的服务可以在集群内通过服务名+服务端口访问。
主机端口访问	NodePort	- 主机端口访问会在每个节点的开放一个静态端口，通过静态端口对外暴露服务。 - 创建完成后的服务可以在集群外通过节点IP+端口访问。

其他访问配置（包含不同负载均衡类型涉及的配置）说明如下：

- LB所在子网：对于VPC内访问类型的Service，需选择LB所在子网，选择范围仅限于集群所在VPC下的终端子网类型子网。
- 端口映射：通过指定协议和端口，配置能够准确被下发到监听器。Service配置中支持四层协议，默认支持协议为TCP。容器端口和服务端口分别指定了后端pod的targetPort，和监听器对外服务端口。

- ExternalTrafficPolicy：在服务类型为LoadBalancer或NodePort时，对外部流量路由到集群内部的转发方式进行设置。Local和Cluster（默认）选择将流量路由到节点本地或集群范围。Local模式支持获取客户端的源IP，转发效率更高，但可能存在流量转发不均衡的潜在风险。Cluster模式整体负载能力更强，同时无法获取客户端的源IP。
- Annotations参数配置：支持通过annotations对负载均衡进行个性化配置，支持指定负载均衡带宽与计费方式等。注释列表请参考通过金山云负载均衡访问服务。

Service基本操作

更新访问方式

在Service列表页面，点击**更新访问方式**，支持对Service的访问方式及其他访问配置进行更新。

删除Service

在Service列表页面，点击**删除**，将删除当前Service资源。

通过金山云负载均衡访问服务

您可以使用金山云负载均衡来访问服务。

前提条件

容器服务提供了金山云的cloud-controller-manager，基于此，用户可以通过金山云的负载均衡（LoadBalancer，以下简称LB）把服务暴露出去。在使用之前，请先确认：

- 集群中已经安装了cloud-controller-manager。

```
# kubectl get deployments -n kube-system
NAME                      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
cloud-controller-manager   1         1         1            1           3h
```

- 集群所属的VPC，安全组**入站规则**中已经放行对应的端口（这里放行了30000-32768端口，生产环境中可以根据业务实际情况放行）。

示例

以下通过YAML创建示例，展示一些常见场景下，如何配置和使用LB，来满足不同的需求。首先，创建一个deployment。

nginx-deployment.yaml如下：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

创建nginx deployment：

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

通过负载均衡向公网暴露服务

这里我们使用金山云负载均衡向公网暴露服务。simple-svc.yaml 如下：

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: nginx
  name: simple-svc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

创建服务，并获取服务的IP地址。

```
# kubectl apply -f simple-svc.yaml
# kubectl get svc
NAME         TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)        AGE
simple-svc   LoadBalancer   10.254.171.216   120.92.xx.xx    80:32733/TCP   11s
```

这里，我们创建了一个名为simple-svc的服务，并通过金山云的LB（指定type为LoadBalancer）将服务暴露出去。通过EXTERNAL-IP（120.92.xx.xx），我们可以访问这个服务。在控制台上，可以看到负载均衡列表里多了一个外网LB（IP地址为120.92.xx.xx，带宽为1m，计费方式为按日月结）。

金山云LB支持丰富的配置参数，为了使用这些配置，需要使用注释（annotations）。完整注释请参考后文附表。

创建HTTP类型的负载均衡

Kubernetes的服务配置中，协议Protocol字段只支持TCP和UDP。如果想使用7层负载均衡，可以通过注释 service.beta.kubernetes.io/ksc-loadbalancer-protocol-port，注释的格式是“PROTOCOL:PORT”（PORT必须和spec:ports中的port一致）。 simple-svc.yaml 如下：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ksc-loadbalancer-protocol-port: "HTTP:80"
  labels:
    app: nginx
  name: simple-http-svc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

创建HTTPS类型的负载均衡

创建HTTPS类型的负载均衡需要在金山云控制台申请一个证书，然后使用如下annotation创建一个HTTPS类型的LB。 https-svc.yaml 如下：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ksc-loadbalancer-protocol-port: "HTTPS:443"
    service.beta.kubernetes.io/ksc-loadbalancer-cert-id: "your-cert-id"
  labels:
    app: nginx
  name: https-lb
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    app: nginx
  type: LoadBalancer
```

使用已有的负载均衡

通过LB暴露服务时，默认会创建一个新的LB。如果不创建新的而是使用一个已经存在的LB，需要在注释中指定LB的ID（注意，如果指定的LB的PORT已经被占用，在创建服务的过程中会删除此监听器）。

支持多个Kubernetes Service复用同一个LB。限制如下：

- Kubernetes通过Service自动创建的LB不能复用（会导致LB被意外删除）。只能复用您手动在控制台（或调用OpenAPI）创建的LB。

svc-using-existing-lb.yaml 如下：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ksc-loadbalancer-id: "your-lb-id"
  labels:
    app: nginx
  name: svc-using-existing-lb
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

使用内网LB

如果是内部服务，不需要将服务暴露在公网，可以使用内网LB。一种方法，是在控制台先创建好内网LB，然后通过注释指定这个LB的ID（参照上述-使用已有的LB）。另一种方法，在注释中指定LB的类型为internal，同时指定一个终端子网的ID，会新建一个内网LB将服务暴露出去。 internal-svc.yaml 如下：

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ksc-loadbalancer-type: "internal"
    service.beta.kubernetes.io/ksc-loadbalancer-subnet-id: ""your-Reserve-id"
  labels:
    app: nginx
  name: internal-svc
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

使用指定Label的worker节点作为后端服务器

多个Label以逗号分隔。例如"k1=v1,k2=v2"。多个label之间是and的关系。 如下所示，lb 只挂载有 "failure-domain.beta.kubernetes.io/zone=cn-beijing-6a" 标签的worker节点。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ksc-loadbalancer-backend-label: "failure-domain.beta.kubernetes.io/zone=cn-beijing-6a"
  name: nginx
  namespace: default
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 443
  selector:
    run: nginx
  type: LoadBalancer
```

注释列表

注释	描述	默认值
service.beta.kubernetes.io/ksc-loadbalancer-id	负载均衡实例的 ID。通过 loadbalancer-id 指定您已有的 SLB，已有 listener 会被覆盖， 删除 service 时该 SLB 不会被删除	
service.beta.kubernetes.io/ksc-loadbalancer-type	指定LB的类型，参考创建负载均衡OpenAPI中的Type字段	public
service.beta.kubernetes.io/ksc-loadbalancer-subnet-id	创建内网LB时，需要指定endpoint子网。参考创建负载均衡OpenAPI中的SubnetId字段	-
service.beta.kubernetes.io/ksc-loadbalancer-bandwidth	外网IP的带宽，参考创建弹性IP OpenAPI中的BandWidth字段	1

service.beta.kubernetes.io/ksc-loadbalancer-charge-type	外网IP的计费方式，参考创建弹性IP OpenAPI中ChargeType字段	会根据用户已有的计费方式创建，优先选择后付费的方式PostPaidByDay
service.beta.kubernetes.io/ksc-loadbalancer-purchase-time	外网IP的购买时长，参考创建弹性IP OpenAPI中PurchaseTime字段	
service.beta.kubernetes.io/ksc-loadbalancer-protocol-port	指定HTTP、HTTPS等协议。多个值之间由逗号分隔，比如：HTTPS:443,HTTP:80	-
service.beta.kubernetes.io/ksc-loadbalancer-method	监听器的转发方式，参考创建监听器OpenAPI中的Method字段	RoundRobin
service.beta.kubernetes.io/ksc-loadbalancer-cert-id	协议为HTTPS时，指定证书ID。参考创建监听器OpenAPI中的CertificateId字段	-
service.beta.kubernetes.io/ksc-loadbalancer-session-state	是否开启会话保持，参考创建监听器OpenAPI中的SessionState字段	start
service.beta.kubernetes.io/ksc-loadbalancer-session-persistence-period	会话保持超时时间，参考创建监听器OpenAPI中的SessionPersistencePeriod字段	3600
service.beta.kubernetes.io/ksc-loadbalancer-cookie-type	协议为HTTP时，指定cookie类型。参考创建监听器OpenAPI中的CookieType字段	ImplantCookie
service.beta.kubernetes.io/ksc-loadbalancer-cookie-name	协议为HTTP时，指定cookie名字。参考创建监听器OpenAPI中的CookieName字段	-
service.beta.kubernetes.io/ksc-loadbalancer-healthcheck-state	是否开启健康检查，参考创建健康检查 OpenAPI中的HealthCheckState字段	stop
service.beta.kubernetes.io/ksc-loadbalancer-healthy-threshold	健康阈值，参考创建健康检查 OpenAPI中的HealthyThreshold字段	-
service.beta.kubernetes.io/ksc-loadbalancer-healthcheck-interval	健康检查时间间隔，参考创建健康检查 OpenAPI中的Interval字段	-
service.beta.kubernetes.io/ksc-loadbalancer-healthcheck-timeout	健康检查超时时间，参考创建健康检查 OpenAPI中的Timeout字段	-
service.beta.kubernetes.io/ksc-loadbalancer-healthcheck-urlpath	HTTP类型监听器健康检查的链接，参考创建健康检查 OpenAPI中的UrlPath字段	-
service.beta.kubernetes.io/ksc-loadbalancer-unhealthy-threshold	不健康阈值，参考创建健康检查 OpenAPI中的UnhealthyThreshold字段	-
service.beta.kubernetes.io/ksc-loadbalancer-healthcheck-hostname	HTTP类型健康检查的域名，参考创建健康检查 OpenAPI中的HostName字段	-
service.beta.kubernetes.io/ksc-loadbalancer-healthcheck-is-default-hostname	参考创建健康检查 OpenAPI中的IsDefaultHostName字段	-

备注:

- 请不要手动删除Kubernetes通过Service自动创建的LB
- 请不要手动更改Kubernetes通过Service自动创建LB的监听器

Ingress概述

Ingress简介

在Kubernetes集群中，Service和Pod的IP仅在集群内部访问，如果外部应用需要访问集群内的服务，集群外部请求将被转发到Service所在节点暴露的NodePort上，然后由Kube-proxy组件将其转发给相关的Pod。而Ingress是为集群外部请求提供的路由规则集合，简单来讲就是提供外部访问集群的入口，将外部的HTTP或HTTPS请求转发到集群内不同Service的后端Pod上。

Ingress资源中定义了外部流量的路由规则，而这些规则的解析与转发需要由Ingress controller来实现。Ingress controller负责解析Ingress规则，在Ingress规则有增删改的变动时同步更新controller对应的访问配置，当Ingress controller收到请求时根据Ingress中的访问规则将请求转发到对应的Service上。因此Ingress规则生效的前提之一，是Kubernetes集群中部署了可以解析对应规则的Ingress controller。Ingress工作原理示意图如下：

Nginx Ingress Controller

容器服务基于Nginx ingress controller和金山云负载均衡服务来实现外部请求在集群内的转发。您可根据业务负载情况对Ingress controller进行自定义容器化部署，也可兼容Nginx ingress开源方案，通过Annotations扩展Kubernetes原生Ingress能力，实现灰度发布、URL重定向等高级配置。

Nginx-ingress 名词解释

- **Nginx-ingress 组件**：容器集群中使用 Nginx-ingress依赖于Nginx-ingress组件，您可以在集群的组件管理页面一键安装部署 Nginx-ingress。
- **Nginx-ingress controller实例**：一个集群中可部署多个 Nginx-ingress controller实例（例如一个用于公网，一个用于内网）。在 Kubernetes 中对应一个 CRD，创建一个 Nginx-ingress 实例会在集群中自动创建 Nginx-ingress-controller、service、configmap 等 Kubernetes 资源。
- **Nginx-ingress-controller**：实际 Nginx 负载， 同时 controller 会 watch kubernetes ingress 对象的变化更新在集群中，Nginx 负载的转发配置即 `nginx.conf` 文件。

Nginx-ingress使用

本文档将介绍如何部署Nginx-ingress controller及配置Ingress规则，以实现集群外部请求的路由转发。

前提条件

在进行Nginx-ingress controller部署前，请确认所选集群满足以下前提条件：

1. 集群Kubernetes版本为1.18及以上；
2. 集群中未部署过其他Nginx-ingress controller。

   注：对于2022年1月13日前，通过控制台部署过老版本Nginx-ingress controller的用户，建议按照以下操作提示创建新版controller，并在业务低峰期时将老版ingress规则切换由新版controller管理。切换后新版controller支持由组件管理方式进行多实例管理，且对应社区版本（v1.1.0）支持更多优化特性。

   切换方式建议如下：

   1. 通过组件管理模块，指定IngressClass，创建Nginx-ingress controller实例。
   2. Nginx-ingress controller默认支持IngressClass为nginx的Ingress规则。由一期通过控制台部署的controller管理的ingress规则（默认指定ingressClass为nginx），可直接由新版controller接管，仅需切换域名解析规则，即可将访问流量切换到新版controller对应的service上。
   3. 选择业务低峰期，将原始Ingress规则中域名映射的IP，由旧版controller对应的service IP，切换为新版controller对应的service IP。

部署Nginx-ingress controller

1. 登录容器服务控制台，选择指定集群进入集群管理页面。
2. 选择**服务管理** > **Ingress**，进入Ingress管理页面。若此前未创建过Nginx-ingress controller，可通过新建按钮跳转至组件管理列表进行controller安装。

3. 选择**组件管理**，进入组件管理页面。完成第一个实例安装后，组件状态将切换为已安装。对于多controller实例的场景（例如一个用于公网，一个用于内网），可进入实例列表进行多controller实例部署。

4. 选择**ingress-nginx**，进入controller实例列表页。点击**新建**，创建nginx-ingress controller实例。

   - 配置IngressClass名称：指定 Nginx-ingress controller实例使用的 IngressClass 名称，构成对应实例Kubernetes资源的名称。
   - 命名空间：为Nginx-ingress controller实例部署指定命名空间。

- 访问范围：根据业务访问需求，为Nginx-ingress controller配置公网/私网访问，将分别为controller创建公网/私网LB作为集群外流量入口。
  - 修改LB配置：默认状态下，会创建普通LoadBalancer模式的Service，LB会绑定各节点的NodePort作为后端RS，流量会经由节点的NodePort。若您有获取客户端来源IP、提升转发性能等需求，可通过修改LB配置**开启LB直通Pod**，外部请求将不经过Nodeport转发直接到达Nginx-ingress实例pod。

    开启LB直通Pod：检查集群中kube-system命名空间下cloud-controller-manager版本，更新至v1.29-mp及更新版，方可支持LB直通Pod配置。
  - 选择公网：需为LB绑定的EIP指定带宽和计费方式。
  - 选择内网：需指定LB所在子网。
- 配置部署方式和调度策略：Nginx-ingress支持DaemonSet和Deployment两种部署方式，为保证Nginx-ingress服务性能，您可以通过指定部署方式和调度策略，将controller实例调度到特定节点上。调度策略配置可参考工作负载调度策略配置。
- 完成配置后，点击**确定**，即可完成Nginx-ingress实例部署。

5. 切换至**实例详情**，指定Nginx-ingress实例，即可查看当前实例下创建的相关Kubernetes资源。

创建Ingress规则

Nginx-ingress controller部署完成后，即可创建业务需要的ingress规则。Ingress规则配置方式如下：

1. 基本信息

- Ingress名称：用户自定义Ingress的名称，不超过63个字符，只能包含小写字母、数字、和“-”，并且必须以小写字母开头，小写字母或数字结尾。
- IngressClass：指定此ingress规则目标匹配的Nginx ingress实例的IngressClass。
- 命名空间：选择Ingress规则所在的命名空间。
- 描述：创建Ingress的相关信息，用户自定义填写。

2. 监听器配置

配置监听端口与协议，支持HTTP和HTTPS。

3. 转发规则配置

根据业务需求，配置服务的访问路径、名称、端口。

- 域名：指定服务对外暴露的域名。域名支持通配符，如*.example.com。
- URL路径：指定服务访问的URL路径。若使用根路径，则配置为“/”。
- 后端Service：每个访问路径都关联一个服务，从对应命名空间下已创建服务中进行选择。
- Service访问端口：选择服务暴露的端口。

4. TLS配置

当监听协议指定为HTTPS时，需通过TLS配置为域名指定SSL证书，其中证书将通过Secret资源来指定。

- 创建Secret：您可通过导入证书内容和私钥的方式一键创建Secret。
- 配置证书：为ingress规则中指定域名配置证书。

5. 配置注解

您可以通过配置注解来实现Nginx-ingress的高阶功能，详情可参见Annotations。

HTTPS安全访问

本文将介绍如何在金山云容器服务中配置HTTPS安全访问。

根据访问方式的不同，目前可以分为两种配置证书的方式：

- 在SLB上配置证书
- 在Ingress中配置证书

前提条件

- 您已经在金山云创建一个Kubernetes集群，且集群中的cloud-controller-manager正常运行。

```
[root@vm10-0-33-13 ~]# kubectl get deploy -n kube-system | grep cloud
cloud-controller-manager   1         1         1            1           35d
```

- 提前准备好证书，这里为了测试需要，我们使用自签名证书，使用如下命令快速创建。

```
[root@vm10-0-33-13 CAtest]# openssl req -newkey rsa:2048 -nodes -keyout tls.key -x509 -days 365 -out tls.crt
Generating a 2048 bit RSA private key
..............+++
.............................+++
writing new private key to 'tls.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:CN
State or Province Name (full name) []:Beijing
Locality Name (eg, city) [Default City]:Beijing
Organization Name (eg, company) [Default Company Ltd]:Kingsoft
Organizational Unit Name (eg, section) []:Ksyun
Common Name (eg, your name or your server's hostname) []:foo.bar.com
Email Address []:ksyun@kingsoft.com
```

在金山云SLB上配置证书

特点：证书配置在负载均衡上，为应用提供外部的访问入口，集群内部访问仍然使用HTTP的形式。

适用场景：应用不使用Ingress暴露访问方式，直接通过LoadBalancer类型的service进行应用访问的暴露。

1. 这里我们以nginx应用为例，首先创建一个nginx的应用，nginx-deploy.yaml如下：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
name: nginx
spec:
replicas: 2
template:
metadata:
  labels:
```

```
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

2. 将我们上一步创建的证书上传至金山云负载均衡-证书管理。

   ⅰ.进入金山云负载均衡控制台，选择**证书 > 负载均衡证书**，点击**创建证书**。

   ⅱ.在创建证书的弹窗中，填写证书名称，在**证书内容**和**私钥**中上传我们在前提条件中创建的证书和私钥，点击**创建**，完成证书的创建。

   ⅲ.在证书详情中获取证书的id。

3. 这里我们通过金山云的负载均衡暴露nginx应用到公网，采用HTTPS的访问形式，nginx-service.yaml如下：

   注意： service.beta.kubernetes.io/ksc-loadbalancer-protocol-port，注释的格式是“PROTOCOL：PORT”（PORT必须和spec:ports中的port一致）

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    service.beta.kubernetes.io/ksc-loadbalancer-protocol-port: "HTTPS:443"
    service.beta.kubernetes.io/ksc-loadbalancer-cert-id: "your-cert-id"  # 请填写您的证书id
  labels:
    app: nginx
  name: https-lb
spec:
  ports:
  - port: 443
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

4. 创建以上资源：

```
[root@vm10-0-33-13]# kubectl create -f nginx-deploy.yaml
deployment.extensions/nginx created
[root@vm10-0-33-13]# kubectl create -f nginx-svc.yaml
service/nginx created
```

5. 获取nginx服务对应的公网ip：

```
[root@vm10-0-33-13 CAtest]# kubectl get svc | grep nginx
nginx    LoadBalancer    10.254.101.229    120.92.86.xx    443:31937/TCP        76d
```

   备注： 获取nginx服务对应的公网ip，这里我们将测试域名foo.bar.com解析到nginx服务的公网IP，可以在hosts文件中添加一条记录。

```
120.92.86.xx foo.bar.com
```

   在浏览器中输入https://foo.bar.com 验证。

在Ingress中配置证书

特点：无需改动SLB的配置；每一个应用都可以通过Ingress管理自己的证书，互不干扰。

适用场景：每个应用都需要单独的证书进行访问；或者集群中存在需要证书才能访问的应用。

1. 根据前提条件中创建的证书和私钥创建secret资源。

```
[root@vm10-0-33-13]# kubectl create secret tls secret-https --key tls.key --cert tls.crt
secret/secret-https created
```

2. 将集群中内置的traefik服务暴露到公网，详见Ingress Controller暴露出集群。

3. 创建服务nginx的服务，这里我们仅需将nginx服务暴露到集群内部即可，服务端口80，nginx-service.yaml如下：

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

4. 创建对应的Ingress规则，ingress.yaml如下：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-https
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          serviceName: nginx
          servicePort: 80
  tls:
  - hosts:
    - foo.bar.com
    secretName: secret-https
```

5. 创建以上service、ingress资源。

```
[root@vm10-0-33-13]# kubectl create -f svc.yaml
service/nginx created
[root@vm10-0-33-13]# kubectl create -f ingress.yaml
ingress.extensions/nginx-https created
[root@vm10-0-33-13 CAtest]# kubectl get svc | grep nginx
nginx                    ClusterIP    10.254.99.223    <none>        80/TCP           3m35s
[root@vm10-0-33-13 CAtest]# kubectl get ing
NAME          HOSTS          ADDRESS    PORTS     AGE
nginx-https    foo.bar.com              80, 443    2m26s
```

6. 获取traefik服务的公网ip。

```
[root@vm10-0-33-13 CAtest]# kubectl get svc -n kube-system | grep traefik
traefik-ingress-service    LoadBalancer    10.254.101.229    120.92.86.xx    80:31833/TCP,443:31937/TCP,8080:30475/TCP        76d
```

   备注：

- 这里我们将测试域名foo.bar.com解析到traefik服务的公网IP，可以在hosts文件中添加一条记录。

```
120.92.86.xx foo.bar.com
```

在浏览器中输入https://foo.bar.com 验证。

ConfigMap管理

ConfigMap 是一种 API 对象，用来将非机密性的数据保存到键值对中。通过ConfigMap 可以将配置文件从容器镜像中解耦，从而增强容器应用的可移植性。ConfigMap 是有 key-value 类型的键值对，可以通过控制台创建对应的 ConfigMap 对象，也可以通过挂载数据卷、环境变量或在容器的运行命令中使用 ConfigMap。

容器服务控制台操作说明

创建ConfigMap

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要创建ConfigMap的集群ID，进入该集群操作页面。
4. 选择**配置管理** > **ConfigMap**，进入ConfigMap列表页。
5. 单击**新建**，在创建ConfigMap页面，根据以下提示进行ConfigMap配置：
   - 名称：输入要创建ConfigMap的名称。
   - 命名空间：请根据实际需求进行选择。
   - 配置项：请根据实际需求定义变量名和变量值。
6. 单击**创建**，完成ConfigMap创建。

使用ConfigMap

方式一：通过挂载ConfigMap类型数据卷

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要使用ConfigMap的集群ID，进入该集群操作页面。
4. 单击**工作负载**，选择任意类型，进入相应列表页。例如选择**工作负载** > **Deployment**，进入Deployment列表页。
5. 单击**新建**，进入创建Deployment页面。
6. 在第二步**部署配置**中进行**存储卷**配置，选择**使用ConfigMap**类型，输入存储卷名称，如下图所示：
7. 点击**选择ConfigMap**，在出现的弹窗中，根据以下提示进行ConfigMap配置：
   - 选择ConfigMap：请根据实际需求进行选择。
   - 挂载选项：有**全部挂载**以及**指定key挂载**两种方式，请根据实际需求进行选择。

   注：当选择**指定key挂载**时，可以通过Items向特定路径挂载，如挂载路径是 /etc/config，子路径是dev，最终会存储在/etc/config/dev下。

8. 按需配置完所有信息后点击**创建**，完成创建。

方式二：通过定义容器环境变量

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要使用ConfigMap的集群ID，进入该集群操作页面。
4. 单击**工作负载**，选择任意类型，进入相应列表页。例如选择**工作负载** > **Deployment**，进入Deployment列表页。
5. 单击**新建**，进入创建Deployment页面，进行容器配置时，在**环境变量**项单击**添加环境变量**，如下图所示：

6. 在环境变量配置中，选择**引用ConfigMap**方式添加，输入变量名以及选择变量值/变量引用，如下图所示：

7. 按需配置完所有信息后点击**创建**，完成创建。

更新ConfigMap

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要更新ConfigMap的集群ID，进入该集群操作页面。
4. 选择**配置管理** > **ConfigMap**，进入ConfigMap列表页。
5. 选择要更新的ConfigMap，单击**编辑YAML**。
6. 在更新ConfigMap页面，根据实际需求进行配置，单击**更新**完成更新。

删除ConfigMap

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要删除ConfigMap的集群ID，进入该集群操作页面。
4. 选择**配置管理** > **ConfigMap**，进入ConfigMap列表页。
5. 选择要删除的ConfigMap，单击**删除**。
6. 在出现的弹窗中，点击**确认**即可删除ConfigMap。

Kubectl 命令操作说明

创建 ConfigMap

方式一：通过YAML文件创建ConfigMap

configmap-test.yaml示例：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config-test
  namespace: default
data:
  SPECIAL_LEVEL: very
  SPECIAL_TYPE: charm
```

- data：ConfigMap 的数据，以 key-value 形式呈现。

创建configmap-test.yaml

```
# kubectl apply -f configmap-test.yaml
```

方式二：通过kubectl create configmap命令直接创建ConfigMap

```
# kubectl create configmap <map-name> <data-source>

# <map-name>：ConfigMap对象的名称
# <data-source>：数据源，可以根据目录、文件或者直接创建ConfigMap对象。
```

使用 ConfigMap

方式一：通过挂载ConfigMap类型数据卷

configmap-volume.yaml示例：

```
apiVersion: v1
kind: Pod
metadata:
   name: pod1-test
spec:
   containers:
     - name: container-test
       image: ksyun/nginx:latest
       volumeMounts:                          #数据卷的挂载点
         - name: config-volume
           mountPath: /etc/config
   volumes:                                   #在pod级别设置卷，然后将其挂载到pod内的容器中
     - name: config-volume
       configMap:
         name: config-test                    #挂载的ConfigMap名称
   restartPolicy: Never
```

方式二：通过定义容器环境变量

configmap-env.yaml

```
apiVersion: v1
kind: Pod
metadata:
   name: pod2-test
spec:
   containers:
     - name: container-test
       image: ksyun/nginx:latest
       env:
         - name: SPECIAL_LEVEL_KEY
           valueFrom:
             configMapKeyRef:
               name: config-test                # 挂载的ConfigMap名称
               key: SPECIAL_LEVEL
   restartPolicy: Never
```

Secret管理

密码、令牌、密钥等敏感信息，可以使用Secret来管理和配置。Secret 是key-value 类型的键值对，可以通过控制台创建对应的Secret对象，也可以通过挂载数据卷、环境变量或在容器的运行命令中使用 Secret。

容器服务控制台操作说明

创建Secret

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要创建Secret的集群ID，进入该集群操作页面。
4. 选择**配置管理 > Secret**，进入Secret列表页。
5. 单击**新建**，在创建Secret页面，根据以下提示进行Secret配置：
   - 名称：输入要创建Secret的名称。
   - 命名空间：请根据实际需求进行选择。
   - 密钥类型：有**Opaque**和**kubernetes.io/dockerconfigjson**两种类型，请根据实际需求进行选择。
     - Opaque：base64编码格式的Secret，用来存储密码、密钥等。
     - kubernetes.io/dockerconfigjson：用于保存私有Docker Registry的认证信息。
   - 当密钥类型选择**Opaque**时：
     - 内容：请根据实际需求定义变量名和变量值。
   - 当密钥类型选择**kubernetes.io/dockerconfigjson**时：
     - 镜像仓库地址：请根据实际需求输入镜像仓库名或 ip。
     - 用户名：请根据实际需求输入镜像仓库登录用户名。
     - 密码：请根据实际需求输入镜像仓库登录密码。
6. 单击**创建**，完成Secret创建。

使用Secret

方式一：通过挂载Secret类型数据卷

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要使用Secret的集群ID，进入该集群操作页面。
4. 单击**工作负载**，选择任意类型，进入相应列表页。例如选择**工作负载 > Deployment**，进入Deployment列表页。
5. 单击**新建**，进入创建Deployment页面，在**存储卷**配置中，选择**使用Secret**类型，输入存储卷名称，如下图所示：

6. 点击**选择Secret**，在出现的弹窗中，根据以下提示进行Secret配置：
   - 选择Secret：请根据实际需求进行选择。
   - 挂载选项：有**全部挂载**以及**指定key挂载**两种方式，请根据实际需求进行选择。

   注：当选择**指定key挂载**时，可以通过Items向特定路径挂载，如挂载路径是 /etc/config，子路径是dev，最终会存储在/etc/config/dev下。

7. 点击**创建**，完成创建。

方式二：通过定义容器环境变量

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要使用Secret的集群ID，进入该集群操作页面。
4. 单击**工作负载**，选择任意类型，进入相应列表页。例如选择**工作负载 > Deployment**，进入Deployment列表页。
5. 单击**新建**，进入创建Deployment页面，进行容器配置时，在**环境变量**项单击**添加环境变量**，如下图所示：

6. 在环境变量配置中，选择**引用Secret**方式添加，输入变量名以及选择变量值/变量引用，如下图所示：

7. 点击**创建**，完成创建。

更新Secret

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要更新Secret的集群ID，进入该集群操作页面。
4. 选择**配置管理 > Secret**，进入Secret列表页。
5. 单击**编辑YAML**，在更新Secret页面，根据实际需求进行配置，单击**更新**完成更新。

删除Secret

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**集群**，进入集群管理页面。
3. 选择需要删除Secret的集群ID，进入该集群操作页面。
4. 选择**配置管理** > **Secret**，进入Secret列表页。
5. 单击**删除**，在出现的弹窗中，点击**确认**即可删除Secret。

Kubectl 命令操作说明

创建Secret

方式一：通过YAML文件创建Secret

首先将Secret的data进行Base64转码

```
# echo -n 'admin' | base64
YWRtaW4=
# echo -n '12345' | base64
MTIzNDU=
```

secret-test.yaml示例：

```
apiVersion: v1
kind: Secret
metadata:
   name: secret-test
type: Opaque
data:
   username: YWRtaW4=
   password: MTIzNDU=
```

创建secret-test.yaml

```
# kubectl apply -f secret-test.yaml
```

方式二：通过kubectl create secret命令直接创建Secret

将用户名和密码保存在本地的./username.txt和./password.txt 文件里。

```
# echo -n 'admin' > ./username.txt
# echo -n '12345' > ./password.txt
```

创建secret

```
# kubectl create secret generic secret-test --from-file=./username.txt --from-file=./password.txt
```

使用Secret

方式一：通过挂载Secret类型数据卷

secret-volume.yaml示例：

```
apiVersion: v1
kind: Pod
metadata:
   name: pod1-test
spec:
   containers:
     - name: container-test
       image: ksyun/nginx:latest
       volumeMounts:                          #数据卷的挂载点
         - name: secret-volume
           mountPath: /etc/config
   volumes:                                   #在pod级别设置卷，然后将其挂载到pod内的容器中
     - name: secret-volume
       secret:
         secretName: secret-test                    #挂载的Secret名称
   restartPolicy: Never
```

方式二：通过定义容器环境变量

secret-env.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: pod2-test
spec:
  containers:
    - name: container-test
      image: ksyun/nainx:latest
      env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: secret-test
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: secret-test
              key: password
  restartPolicy: Never
```

存储卷概述

存储卷概述

我们知道默认情况下容器的数据都是非持久化的，在容器消亡以后数据也跟着丢失，所以 Docker 提供了 Volume 机制以便将数据持久化存储。类似的，Kubernetes 提供了更强大的 Volume 机制和丰富的插件，解决了容器数据持久化和容器间共享数据的问题。金山云容器服务采用Kubernetes中存储卷的概念，支持多种类型的存储卷，同时Pod可以使用任意数量的存储卷。

存储卷类型

金山云容器服务基于原生的Kuberenetes开发和适配，你可以设置以下类型的存储卷：

1. **本地硬盘**：将容器所在宿主机的文件目录挂载到容器的指定路径中（对应Kubernetes的HostPath），也可以不填写源路径（对应Kubernetes中的EmptyDir），不填写时将分配主机的临时目录挂载到容器的挂载点，指定源路径的本地硬盘数据卷适用于将数据持久化存储到容器所在宿主机，EmptyDir适用于容器的临时存储。
2. **云硬盘**：金山云容器服务支持使用金山云云硬盘作为Kubernetes集群的存储卷，您可以指定金山云的EBS云硬盘挂载到容器的某一路径，容器迁移，云硬盘会跟随迁移，适用于数据的持久化存储。由于EBS仅支持ReadWriteOnce的访问模式，因此在设置云硬盘的服务时，实例的最大数量为1。
3. **KFS**：可为Kubernetes集群实例提供可扩展的共享文件存储服务，提供标准的文件访问协议，现有应用无需任何修改即可挂载使用，适用于内容管理、企业办公文件共享以及媒体处理等场景。

使用本地硬盘存储卷

使用本地硬盘有两种形式：

- 指定源路径（HostPath）,将容器所在宿主机的文件目录挂载到容器指定的挂载点中，如容器需要访问/etc/hosts则可以使用HostPath映射/etc/hosts等场景。

- 临时路径挂载（EmptyDir）：用于临时存储，生命周期与容器实例相同。容器实例消亡时，EmptyDir会被删除，数据会永久丢失。

指定源路径（HostPath）挂载

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: nginx
  name: nginx
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
        volumeMounts:
        - mountPath: /data
          name: hostpath
      volumes:
      - hostPath:
          path: /var
        name: hostpath
```

临时路径（EmptyDir）挂载

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: nginx
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:latest
        imagePullPolicy: Always
        name: nginx
        volumeMounts:
        - mountPath: /data
          name: emptydir
      volumes:
      - emptyDir: {}
        name: emptydir
```

注： 本地硬盘数据卷源路径为空时，系统分配临时目录在/data/kubelet/pods/pod_id/volumes/kubernetes.io~empty-dir. 使用临时的数据卷的生命周期与实例的生命周期保持一致。

使用云硬盘存储卷

您可以在金山云容器服务Kubernetes集群中使用云硬盘存储卷，支持以静态存储卷的形式挂载到容器的某一路径，云硬盘会随着容器的迁移而迁移。

目前，金山云提供两种kubernetes挂载方式：

- 静态存储卷

  可以通过以下两种方式使用云硬盘静态存储卷：

  - 直接通过volume使用
  - 通过PV/PVC使用

- 动态存储卷

静态存储卷

使用说明

1、云硬盘为非共享存储，只能被一个实例挂载，实例数量需要设置为1。

2、使用前需要先在控制台申请一块云硬盘，并获得磁盘 ID（volumeId）。

3、volumeName、PV Name要与之volumeId相同。

4、集群中只有与云盘在同一个可用区（Zone）的节点才可以挂载云盘。

5、文件系统类型（fsType）支持ext3、ext4、xfs。 6、目前金山云售卖云服务器中，通用性N1、通用型N2、IO优化型I2和IO优化型I3云主机支持挂载云硬盘（详见云硬盘使用限制）。如您选择云硬盘作为存储卷，建议在创建服务时，将pod调度到以上机型，否则可能存在云硬盘无法挂载的情况出现。具体调度方式参考将 Pod 分配给节点。

直接通过volume使用

下面的示例**nginx-disk-deploy.yaml**将yaml文件中声明的EBS云硬盘挂在到pod内的nginx-flexvolume-disk容器的**/data**路径下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-disk-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-flexvolume-disk
        image: nginx
```

```
        volumeMounts:
          - name: "b5c80953-1499-40ff-918a-4d6d4dfbfddd"
            mountPath: "/data"
      volumes:
        - name: "b5c80953-1499-40ff-918a-4d6d4dfbfddd"
          flexVolume:
            driver: "ksc/ebs"
            fsType: "ext4"
            options:
              volumeId: "b5c80953-1499-40ff-918a-4d6d4dfbfddd"
```

通过PV/PVC使用

定义PV：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "b5c80953-1499-40ff-918a-4d6d4dfbfddd"
spec:
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "ksc/ebs"
    fsType: "ext4"
    options:
      volumeId: "b5c80953-1499-40ff-918a-4d6d4dfbfddd"
```

定义PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
```

创建deployment：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-disk-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-flexvolume-disk
        image: nginx
        volumeMounts:
        - name: pvc-disk
          mountPath: "/data"
      volumes:
      - name: pvc-disk
        persistentVolumeClaim:
          claimName: pvc-disk
```

动态存储卷

动态存储卷需要手动创建 StorageClass，并在PVC中指定storageClassName。

创建StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ssd30
provisioner: ksc/ebs
parameters:
  type: SSD3.0
  zone: cn-beijing-6b # 选填 #
  chargetype: Daily
```

参数说明：

- provisioner：配置为 ksc/ebs，指定使用金山云Provsioner 插件创建。
- reclaimPolicy: 云盘的回收策略，默认为Delete，支持Retain。
- type：EBS类型，必填，可选参数：SSD2.0/SSD3.0/SATA2.0/SATA3.0（字母全部大写）。
- zone(选填)：创建云盘的可用区，注意不同可用区可创建的EBS类型不一样，具体对应关系参考云硬盘使用限制。当不指定zone参数时，则会在集群所拥有的全部节点所在可用区中随机选择可用区创建云盘。
- chargetype:云盘的计费方式，默认值为Daliy，详情参考创建云硬盘OpenAPI中的chargetype字段。
- purchasetime：若选择"包年包月"的计费方式，需要设置购买时长，单位为月。

创建Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - name: pvc-disk
          mountPath: "/data"
      volumes:
      - name: pvc-disk
        persistentVolumeClaim:
          claimName: nginx-pvc
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-pvc
```

```
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ssd30
  resources:
    requests:
      storage: 20Gi
```

使用KFS文件存储

您可以在金山云容器服务Kubernetes集群中使用金山云KFS存储卷。

目前，金山云提供两种kubernetes挂载方式：

- 静态存储卷

  可以通过以下两种方式使用KFS文件存储静态存储卷：

  - 直接通过volume使用
  - 通过PV/PVC使用

- 动态存储卷

前提

挂载文件系统（KFS）的前提是您有创建好的文件系统。如果您还未创建文件系统，您需要先创建文件系统。有关如何创建文件系统的详细信息，参见创建文件系统及挂载点

说明

1. 金山云KFS为共享存储，可以同时为多个 Pod 提供共享存储服务，即一个PVC可以同时被多个Pod 使用。
2. 在没有卸载文件系统前，务必不要删除文件系统的挂载点，否则会造成操作系统Hang。
3. KFS动态存储卷能力，集群创建时间在2021-2-24及之后的集群可以直接使用，创建时间在2021-2-24之前，需要更新集群中Kube-system命名空间下disk-provisioner组件，YAML见附录

查看文件系统

如上图所示：

server：10.0.1.x

挂载路径：/share-2c51940a-6261-4044-addc-xxxxxxxx

静态存储卷

直接通过Volume使用

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kfs
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - name: "kfs"
            mountPath: "/data"
      volumes:
        - name: "kfs"
          nfs:
            server: "10.0.1.x"
            path: "/share-2c51940a-6261-4044-addc-xxxxxxx"
```

通过PV/PVC使用

创建PV：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: kfs-pv
spec:
  storageClassName: "kfs"
  capacity:
    storage: 100Mi
  accessModes:
    - ReadWriteMany
  mountOptions:
    - nfsvers=3
  nfs:
    server: 10.0.1.x
    path: "/share-2c51940a-6261-4044-addc-xxxxxxx"
```

创建PVC：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: kfs-pvc
spec:
  storageClassName: "kfs"
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 100Mi
```

创建Deployment：

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: kfs-deploy
spec:
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - name: kfs
          mountPath: "/data"
```

```
      volumes:
      - name: kfs
        persistentVolumeClaim:
          claimName: kfs-pvc
```

动态存储卷

参数说明

参数	说明
server	指定kfs服务器地址（必填）
path	指定kfs挂载目录（必填，子目录不存在时可自动创建）
archiveOnDelete	表示删除PVC、PV时候，处理kfs子目录的方式：如果reclaimPolicy为Delete，且archiveOnDelete为false：会直接删除远端目录和数据，请谨慎使用。如果reclaimPolicy为Delete，且archiveOnDelete为true：会将远端的目录更新为其他名字备份。如果reclaimPolicy为Retain，远端的目录不作处理。（选填，默认为false）
storageType	ksc/kfs，指定使用kfs存储（必填）

创建Storageclass：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ksc-kfs
parameters:
  server: 10.0.1.xx
  path: /cfs-AyQaA3xxxx/test-path
  archiveOnDelete: "false"
  storageType: ksc/kfs
provisioner: ksc/storage
```

创建StatefulSet：

```
apiVersion:  apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  serviceName: test
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx
          volumeMounts:
            - name: www
              mountPath: "/data"
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes:
      - ReadWriteOnce
      storageClassName: ksc-kfs
      resources:
        requests:
          storage: 10Gi
```

附录

disk-provisioneryaml文件如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: disk-provisioner
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: disk-provisioner
  replicas: 1
  revisionHistoryLimit: 2
  template:
    metadata:
      labels:
        app: disk-provisioner
    spec:
      dnsPolicy: Default
      tolerations:
        # this taint is set by all kubelets running `--cloud-provider=external`
        - key: "node.cloudprovider.kubernetes.io/uninitialized"
          value: "true"
          effect: "NoSchedule"
      containers:
      - image: hub.kce.ksyun.com/ksyun/disk-provisioner:latest
        name: ebs-provisioner
        env:
        - name: OPENAPI_ENDPOINT
          value: "internal.api.ksyun.com"
        - name: OPENAPI_PREFIX
          value: "http"
        volumeMounts:
          - name: kubeconfig
            mountPath: /root/.kube/config
          - name: clusterinfo
            mountPath: /opt/app-agent/arrangement/clusterinfo
      - image: hub.kce.ksyun.com/ksyun/disk-provisioner:latest
        name: ksc-storage-provisioner
        securityContext:
          privileged: true   # do mount
        args:
        - --provisioner=ksc/storage
        env:
        - name: OPENAPI_ENDPOINT
          value: "internal.api.ksyun.com"
        - name: OPENAPI_PREFIX
          value: "http"
        volumeMounts:
          - name: kubeconfig
            mountPath: /root/.kube/config
          - name: clusterinfo
            mountPath: /opt/app-agent/arrangement/clusterinfo
      volumes:
        - name: kubeconfig
          hostPath:
            path: /root/.kube/config
        - name: clusterinfo
          hostPath:
            path: /opt/app-agent/arrangement/clusterinfo
```

使用自建NFS

用户可以根据自身需要，选择容器集群中的主机或者在容器集群所在VPC内新建云主机来搭建NFS，也支持本地自建的NFS。NFS文件存储适用于多读多写的持久化存储。本文将详细介绍搭建NFS的教程。

搭建NFS流程

在NFS服务器上，安装NFS服务：

```
yum install rpcbind nfs-utils -y
mkdir -p /nfs
```

配置共享目录：

```
cat >/etc/exports<<-EOF
/nfs  172.31.0.0/16(rw,sync,no_root_squash)
EOF
```

其中172.31.0.0/16(rw,sync,no_root_squash) 表示允许172.31.0.0/16网段地址允许以root权限读写NFS，我们建议来访地址配置为容器集群所在VPC网段。

启动服务并设置开机自启动：

```
systemctl enable rpcbind
systemctl enable nfs
systemctl start rpcbind
systemctl start nfs
```

检查配置：

```
# exportfs
/nfs            172.31.0.0/16
```

安装NFS客户端

在容器集群的所有节点上，安装NFS客户端：

```
yum install nfs-utils -y
```

Deployment使用NFS示例

以下示例，创建了一个含有2个副本的Deployment，并且使用NFS共享存储（NFS服务器地址为172.31.22.2）。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
        - mountPath: /usr/share/nginx/html
          readOnly: false
          name: nginx-data
      volumes:
      - name: nginx-data
        nfs:
          server: 172.31.22.2
          path: "/nfs"
```

镜像仓库概述

镜像仓库是用户存放Docker镜像的地方，金山云容器镜像服务提供用户提供安全可靠的企业级私有镜像仓库，用户可将私有的镜像托管至私有的镜像仓库中，方便管理。

镜像类型

目前金山云支持Docker Hub官方镜像，用户私有镜像以及Ksyun Hub镜像。

基本概念

镜像仓库存储的地方称为registry（注册服务器），其上往往存放着多个仓库。每个仓库集中存放某一类镜像，往往包括多个镜像文件，通过不同的标签（tag）来进行区分。例如存放Ubuntu操作系统镜像的仓库，称为Ubuntu仓库，其中可能包括14.04,12.04等不同版本的镜像。 根据存储的镜像公开与否，镜像仓库分为共有仓库（public）和私有仓库（private），目前最大的公有仓库是Docker Hub，用户可以从上面下载大量Docker官方的镜像。

当客户需要操作私有仓库的镜像时，需要先登录registry，才能够拉取镜像或者推送镜像到镜像仓库中。

每个镜像都有唯一的特定地址：镜像registry域名+命名空间名称+仓库名称+镜像标签，例如：

hub.kce.ksyun.com/namespace/nginx:latest

其中：

hub.kce.ksyun.com是registry的域名；

namespace是用户命名空间的名称；

nginx是用户镜像仓库的名称；

latest是镜像的标签，非必填，默认为latest。

镜像仓库基本操作

开通镜像仓库

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**镜像仓库**> **我的镜像**，进入我的镜像管理页面。用户若是首次使用“我的镜像”，需要进行初始化，设置镜像仓库密码。
   - **用户名**：用户名默认是金山云账号ID，暂不可变更。
   - **密码**：密码用于通过 docker login 来登陆金山云容器镜像仓库。密码用户自定义，8-32个字符，必须包含字母、数字和特殊字符中至少两项，支持英文特殊字符!$%()*+,-./:;<=>?@[]^_`{|}

创建命名空间

命名空间是您创建的私人镜像地址的前缀。命名空间将用于对您的容器镜像进行分类。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**镜像仓库**> **空间管理**，进入空间管理管理页面。
3. 点击**新建命名空间**，执行创建命名空间的操作。
4. 设置相关信息：
   - **命名空间名称**：用户自定义，长度为4-30位，支持填写小写英文字母、数字，可使用的分隔符包括“_”、“-”（分隔符不能在首位或末位）。
   - **命名空间类型**：命名空间的类型决定了该命名空间下镜像仓库的类型属性。
5. 点击**确定**，完成命名空间创建。

创建镜像仓库

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**镜像仓库**> **我的镜像**，进入我的镜像管理页面。
3. 点击**新建镜像仓库**，执行新建镜像的操作。
4. 设置相关信息：
   - **名称**：镜像的名称，长度为2-30位，支持填写小写英文字母、数字，可使用的分隔符包括“_”、“-”（分隔符不能在首位或末位）。
   - **命名空间**：选择镜像仓库所在的命名空间。
   - **类型**：继承命名空间的类型。
   - **描述**：镜像的描述信息。
5. 点击**确定**，完成镜像仓库的创建。

推送镜像到镜像仓库

登录金山云docker registry

这里我们以北京地域为例：

```
$ sudo docker login --username=[username] hub.kce.ksyun.com
```

[username]是您的金山云账号ID，输入密码后即登陆完成,密码是您注册镜像仓库设置的密码。

上传镜像

```
$ sudo docker tag [ImageId] hub.kce.ksyun.com/[namespace]/[ImageName]:[tag]
$ sudo docker push hub.kce.ksyun.com/[namespace]/[ImageName]:[tag]
```

[ImageId]请根据您的实际镜像 ID 信息进行填写。

[tag]请根据您的镜像版本信息进行填写。

[namespace]是开通镜像仓库时填写的命名空间。

[ImageName]是在控制台创建的镜像名称。

下载镜像

```
$ sudo docker pull hub.kce.ksyun.com/[namespace]/[ImageName]:[tag]
```

[tag]请根据您需要下载的镜像版本信息进行填写。

Helm Charts操作指引

本文将介绍Helm Chart的一些基本操作，目前支持以下两种方式：

控制台操作指引

上传Helm Chart

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**镜像仓库** > **Helm Charts**，进入Helm Charts管理页面。
3. 选择合适的命名空间，点击**上传**，选择需要上传的Chart包进行上传。

命令行操作指引

前提

1. 本地计算机已经成功安装、配置kubectl和Helm客户端。 可参考使用Helm本地客户端连接集群。
2. 本地客户端添加金山云Helm Chart的Repo，这里以华北1（北京）为例。

   ```
   helm repo add mycharts https://hub.kce.ksyun.com/chartrepo/mycharts --username=myname --password=mypassword
   "mycharts" has been added to your repositories
   ```

   其中，

   - mychart:替换为自己仓库的命名空间。
   - myname:替换为用户镜像仓库的id，一般是用户的金山云账号id。
   - mypassword:替换为用户镜像仓库的密码。

3. 安装官方提供helm push插件

   ```
   helm plugin install https://github.com/chartmuseum/helm-push
   ```

上传Chart

1. 上传文件夹

   ```
   helm push ./helm-apps mychart
   ```

2. 上传压缩包

   ```
   helm push ./helm-apps.tgz mychart
   ```

下载Chart

1. 下载最新版本

   ```
   helm fetch mychart/helm-app
   ```

2. 下载指定版本

   ```
   helm fetch mychart/helm-app --version 1.0.0
   ```

镜像安全扫描

针对于Linux镜像，金山云容器服务服务提供了镜像安全扫描的功能。您可以容器控制台一键启动镜像安全扫描。

目前仅支持基于Linux系统的镜像，支持基于以下指定基础镜像构建的镜像进行安全扫描。

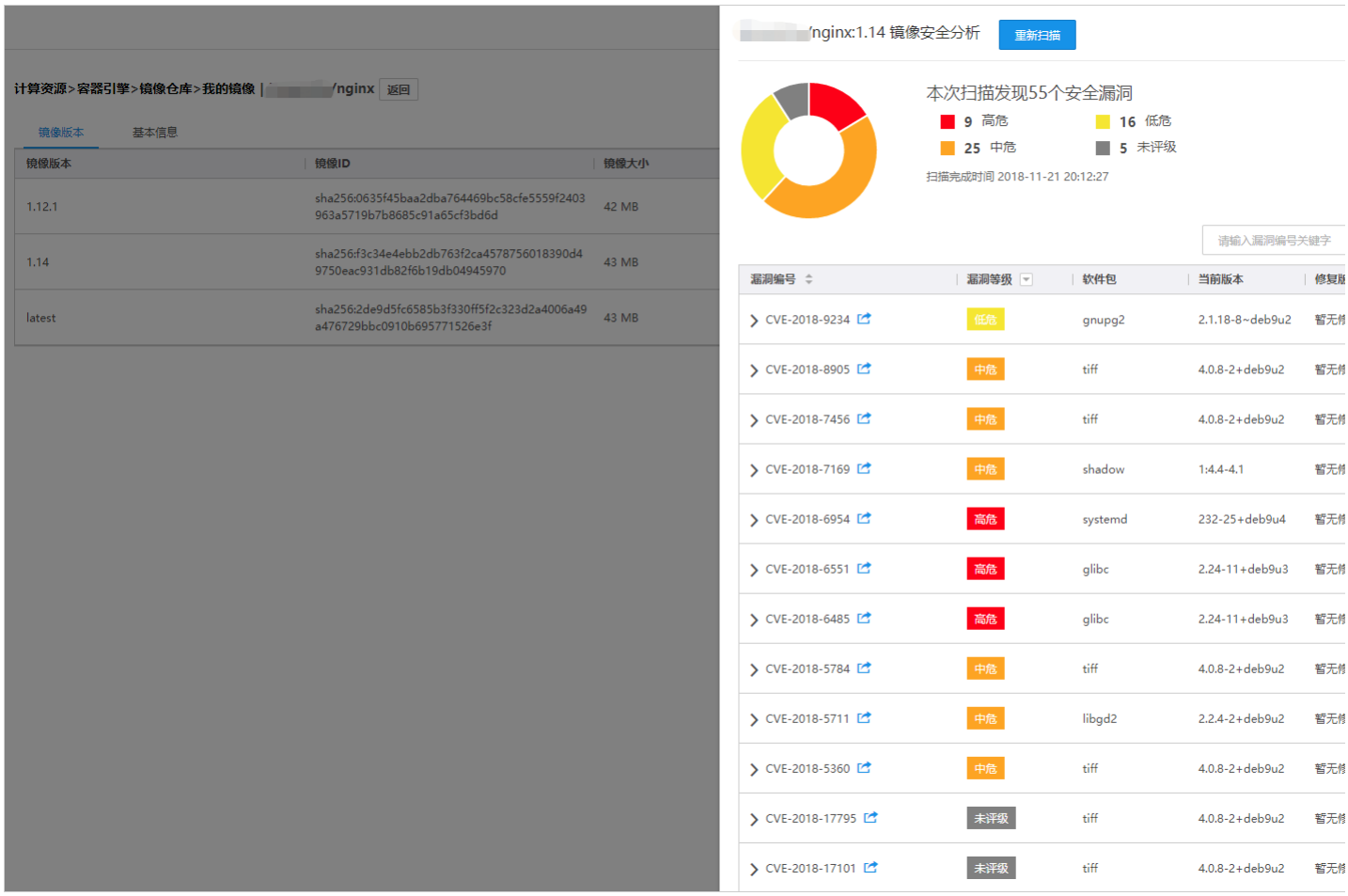
- **Alpine**：3.3、3.4、3.5、3.6、3.7、3.8
- **Debian**：8、9、10、unstable
- **Ubuntu**：12.04以上

- CentOS：5、6、7
- Oracle：5、6、7

使用说明

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**镜像仓库** > **我的镜像**，进入镜像仓库的镜像版本列表页面，选择需要的镜像版本，点击**安全扫描**，触发对该镜像的安全扫描。
3. 安全扫描完成后，会形成详细的漏洞扫描报告，用户可以查询每个漏洞的详细信息。在这里，我们漏洞的严重情况，将漏洞分为四级：“高危”、“中危”、“低危”、“未评级”。

注：金山云容器服务通过每天同步NVD（美国国家漏洞数据库）、Debian Security Bug Tracker、Ubuntu CVE Tracker等CVE库中漏洞和补丁信息，为您的镜像安全提供第一时间的信息。



使用私有镜像创建服务

如果用户需要使用私有镜像创建服务，需要对应的密钥，我们推荐采用imagePullSecrets的方式来实现。下面详细介绍创建步骤，更多关于imagePullSecrets的信息，请参考Kubetnetes 官方文档。

通过kubectl创建secret

```
$ kubectl create secret docker-registry myregistrykey --docker-server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL --namespace=NAMESPACE
secret "myregistrykey" created
```

其中：

——***myregistrykey***：密钥的名称，用户自定义。

——**docker-server**：镜像仓库的地址。

——**docker-username**：镜像仓库的用户名。

——**docker-password**：镜像仓库的登录密码。

——**docker-email**：指定的邮件地址。

——**namespace**：指定的命名空间。

在pod中引用ImagePullSecrets

在创建pod时，可以在pod的定义中增加ImagePullSecrets来引用刚刚创建的**myregistrykey**。

```
apiVersion: v1
kind: Pod
metadata:
  name: test
  namespace: default
spec:
  containers:
    - name: test
      image: hub.kce.ksyun.com/namespace/test:v1
  imagePullSecrets:
    - name: myregistrykey
```

备注： 对每个使用私有镜像的pod，都需要做以上操作。 Pod只能引用和它相同namespace的imagePullSecrets， 所以需要为每一个namespace做配置。

EFK日志收集系统搭建指南

EFK日志采集系统简介

EFK = ElasticSearch + Fluentd + Kibana

Elasticsearch 是一个分布式的搜索和分析引擎，可以用于全文检索、结构化检索和分析，并能将这三者结合起来。Elasticsearch 基于 Lucene 开发，现在是使用最广的开源搜索引擎之一。

Fluentd是一个优秀的log信息收集的开源免费软件。

Kibana是一个开源的分析与可视化平台，可以用kibana搜索、查看存放在Elasticsearch索引里。

部署Fluentd

DaemonSet fluentd-es 只会调度到设置了标签 **beta.kubernetes.io/fluentd-ds-ready=true**的 Node，需要在期望运行 fluentd 的 Node 上设置该标签。

```
# kubectl get nodes
NAME            STATUS    ROLES     AGE       VERSION
172.31.22.16    Ready     <none>    31d       v1.8.3+f0efb3cb88375
172.31.22.3     Ready     <none>    31d       v1.8.3+f0efb3cb88375
172.31.22.6     Ready     <none>    31d       v1.8.3+f0efb3cb88375

# kubectl label nodes 172.31.22.16 172.31.22.3 172.31.22.6 beta.kubernetes.io/fluentd-ds-ready=true
```

创建fluentd-es configmap。

```
# kubectl apply -f fluentd-es-configmap.yaml
```

fluentd-es-configmap.yaml文件如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluentd-es-config-v0.1.4
  namespace: kube-system
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
data:
  system.conf: |-
    <system>
      root_dir /tmp/fluentd-buffers/
    </system>

  containers.input.conf: |-
    # This configuration file for Fluentd / td-agent is used
    # to watch changes to Docker log files. The kubelet creates symlinks that
    # capture the pod name, namespace, container name & Docker container ID
    # to the docker logs for pods in the /var/log/containers directory on the host.
    # If running this fluentd configuration in a Docker container, the /var/log
    # directory should be mounted in the container.
    #
    # These logs are then submitted to Elasticsearch which assumes the
    # installation of the fluent-plugin-elasticsearch & the
    # fluent-plugin-kubernetes_metadata_filter plugins.
    # See https://github.com/uken/fluent-plugin-elasticsearch &
    # https://github.com/fabric8io/fluent-plugin-kubernetes_metadata_filter for
    # more information about the plugins.
    #
    # Example
    # =======
    # A line in the Docker log file might look like this JSON:
    #
    # {"log":"2014/09/25 21:15:03 Got request with path wombat\n",
    #  "stream":"stderr",
    #    "time":"2014-09-25T21:15:03.499185026Z"}
    #
    # The time_format specification below makes sure we properly
    # parse the time format produced by Docker. This will be
    # submitted to Elasticsearch and should appear like:
    # $ curl 'http://elasticsearch-logging:9200/_search?pretty'
    # ...
    # {
    #      "_index" : "logstash-2014.09.25",
    #      "_type" : "fluentd",
    #      "_id" : "VBrbor2QTuGpsQyTCdfzqA",
    #      "_score" : 1.0,
    #      "_source":{"log":"2014/09/25 22:45:50 Got request with path wombat\n",
    #                 "stream":"stderr","tag":"docker.container.all",
    #                 "@timestamp":"2014-09-25T22:45:50+00:00"}
    #    },
    # ...
    #
    # The Kubernetes fluentd plugin is used to write the Kubernetes metadata to the log
    # record & add labels to the log record if properly configured. This enables users
    # to filter & search logs on any metadata.
    # For example a Docker container's logs might be in the directory:
    #
    #  /data/docker/containers/997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b
    #
    # and in the file:
    #
    #  997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b-json.log
    #
    # where 997599971ee6... is the Docker ID of the running container.
    # The Kubernetes kubelet makes a symbolic link to this file on the host machine
    # in the /var/log/containers directory which includes the pod name and the Kubernetes
    # container name:
    #
    #    synthetic-logger-0.25lps-pod_default_synth-lgr-997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b.log
    #    ->
    #    /data/docker/containers/997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b/997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b-json.log
    #
    # The /var/log directory on the host is mapped to the /var/log directory in the container
    # running this instance of Fluentd and we end up collecting the file:
    #
    #   /var/log/containers/synthetic-logger-0.25lps-pod_default_synth-lgr-997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b.log
    #
    # This results in the tag:
    #
    #  var.log.containers.synthetic-logger-0.25lps-pod_default_synth-lgr-997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b.log
    #
    # The Kubernetes fluentd plugin is used to extract the namespace, pod name & container name
    # which are added to the log message as a kubernetes field object & the Docker container ID
    # is also added under the docker field object.
    # The final tag is:
    #
    #   kubernetes.var.log.containers.synthetic-logger-0.25lps-pod_default_synth-lgr-997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b.log
    #
    # And the final log record look like:
    #
    # {
    #   "log":"2014/09/25 21:15:03 Got request with path wombat\n",
    #   "stream":"stderr",
    #   "time":"2014-09-25T21:15:03.499185026Z",
    #   "kubernetes": {
    #     "namespace": "default",
    #     "pod_name": "synthetic-logger-0.25lps-pod",
    #     "container_name": "synth-lgr"
    #   },
    #   "docker": {
    #     "container_id": "997599971ee6366d4a5920d25b79286ad45ff37a74494f262e3bc98d909d0a7b"
    #   }
    # }
    #
    # This makes it easier for users to search for logs by pod name or by
    # the name of the Kubernetes container regardless of how many times the
    # Kubernetes pod has been restarted (resulting in a several Docker container IDs).
```

```
    # Json Log Example:
    # {"log":"[info:2016-02-16T16:04:05.930-08:00] Some log text here\n","stream":"stdout","time":"2016-02-17T00:04:05.931087621Z"}
    # CRI Log Example:
    # 2016-02-17T00:04:05.931087621Z stdout F [info:2016-02-16T16:04:05.930-08:00] Some log text here
    <source>
      @id fluentd-containers.log
      @type tail
      path /var/log/containers/*.log
      pos_file /var/log/es-containers.log.pos
      time_format %Y-%m-%dT%H:%M:%S.%NZ
      tag raw.kubernetes.*
      read_from_head true
      <parse>
        @type multi_format
        <pattern>
          format json
          time_key time
          time_format %Y-%m-%dT%H:%M:%S.%NZ
        </pattern>
        <pattern>
          format /^(?<time>.+) (?<stream>stdout|stderr) [^ ]* (?<log>.*)$/
          time_format %Y-%m-%dT%H:%M:%S.%N%:z
        </pattern>
      </parse>
    </source>

    # Detect exceptions in the log output and forward them as one log entry.
    <match raw.kubernetes.**>
      @id raw.kubernetes
      @type detect_exceptions
      remove_tag_prefix raw
      message log
      stream stream
      multiline_flush_interval 5
      max_bytes 500000
      max_lines 1000
    </match>

  system.input.conf: |-
    # Example:
    # 2015-12-21 23:17:22,066 [salt.state       ][INFO    ] Completed state [net.ipv4.ip_forward] at time 23:17:22.066081
    <source>
      @id minion
      @type tail
      format /^(?<time>[^ ]* [^ ,]*)[^\[]*\[[^\]]*\]\[(?<severity>[^ \]]*) *\] (?<message>.*)$/
      time_format %Y-%m-%d %H:%M:%S
      path /var/log/salt/minion
      pos_file /var/log/salt.pos
      tag salt
    </source>

    # Example:
    # Dec 21 23:17:22 gke-foo-1-1-4b5cbd14-node-4eoj startupscript: Finished running startup script /var/run/google.startup.script
    <source>
      @id startupscript.log
      @type tail
      format syslog
      path /var/log/startupscript.log
      pos_file /var/log/es-startupscript.log.pos
      tag startupscript
    </source>

    # Examples:
    # time="2016-02-04T06:51:03.053580605Z" level=info msg="GET /containers/json"
    # time="2016-02-04T07:53:57.505612354Z" level=error msg="HTTP Error" err="No such image: -f" statusCode=404
    # TODO(random-liu): Remove this after cri container runtime rolls out.
    <source>
      @id docker.log
      @type tail
      format /^time="(?<time>[^)]*)" level=(?<severity>[^ ]*) msg="(?<message>[^"]*)"( err="(?<error>[^"]*)")?( statusCode=($<status_code>\d+))?/
      path /var/log/docker.log
      pos_file /var/log/es-docker.log.pos
      tag docker
    </source>

    # Example:
    # 2016/02/04 06:52:38 filePurge: successfully removed file /var/etcd/data/member/wal/00000000000006d0-00000000010a23d1.wal
    <source>
      @id etcd.log
      @type tail
      # Not parsing this, because it doesn't have anything particularly useful to
      # parse out of it (like severities).
      format none
      path /var/log/etcd.log
      pos_file /var/log/es-etcd.log.pos
      tag etcd
    </source>

    # Multi-line parsing is required for all the kube logs because very large log
    # statements, such as those that include entire object bodies, get split into
    # multiple lines by glog.

    # Example:
    # I0204 07:32:30.020537    3368 server.go:1048] POST /stats/container/: (13.972191ms) 200 [[Go-http-client/1.1] 10.244.1.3:40537]
    <source>
      @id kubelet.log
      @type tail
      format multiline
      multiline_flush_interval 5s
      format_firstline /^\w\d{4}/
      format1 /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
      time_format %m%d %H:%M:%S.%N
      path /var/log/kubelet.log
      pos_file /var/log/es-kubelet.log.pos
      tag kubelet
    </source>

    # Example:
    # I1118 21:26:53.975789       6 proxier.go:1096] Port "nodePort for kube-system/default-http-backend:http" (:31429/tcp) was open before and is still needed
    <source>
      @id kube-proxy.log
      @type tail
      format multiline
      multiline_flush_interval 5s
      format_firstline /^\w\d{4}/
      format1 /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
      time_format %m%d %H:%M:%S.%N
      path /var/log/kube-proxy.log
      pos_file /var/log/es-kube-proxy.log.pos
      tag kube-proxy
    </source>

    # Example:
    # I0204 07:00:19.604280       5 handlers.go:131] GET /api/v1/nodes: (1.624207ms) 200 [[kube-controller-manager/v1.1.3 (linux/amd64) kubernetes/6a81b50] 127.0.0.1:38266]
    <source>
      @id kube-apiserver.log
      @type tail
      format multiline
      multiline_flush_interval 5s
      format_firstline /^\w\d{4}/
      format1 /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
      time_format %m%d %H:%M:%S.%N
      path /var/log/kube-apiserver.log
      pos_file /var/log/es-kube-apiserver.log.pos
```

```
    tag kube-apiserver
  </source>

  # Example:
  # I0204 06:55:31.872680       5 servicecontroller.go:277] LB already exists and doesn't need update for service kube-system/kube-ui
  <source>
    @id kube-controller-manager.log
    @type tail
    format multiline
    multiline_flush_interval 5s
    format_firstline /^\w\d{4}/
    formatl /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
    time_format %m%d %H:%M:%S.%N
    path /var/log/kube-controller-manager.log
    pos_file /var/log/es-kube-controller-manager.log.pos
    tag kube-controller-manager
  </source>

  # Example:
  # W0204 06:49:18.239674       7 reflector.go:245] pkg/scheduler/factory/factory.go:193: watch of *api.Service ended with: 401: The event in requested index is outdated and cleared (the requested history has been cleared [2578313/2577886]) [2579312]
  <source>
    @id kube-scheduler.log
    @type tail
    format multiline
    multiline_flush_interval 5s
    format_firstline /^\w\d{4}/
    formatl /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
    time_format %m%d %H:%M:%S.%N
    path /var/log/kube-scheduler.log
    pos_file /var/log/es-kube-scheduler.log.pos
    tag kube-scheduler
  </source>

  # Example:
  # I1104 10:36:20.242766       5 rescheduler.go:73] Running Rescheduler
  <source>
    @id rescheduler.log
    @type tail
    format multiline
    multiline_flush_interval 5s
    format_firstline /^\w\d{4}/
    formatl /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
    time_format %m%d %H:%M:%S.%N
    path /var/log/rescheduler.log
    pos_file /var/log/es-rescheduler.log.pos
    tag rescheduler
  </source>

  # Example:
  # I0603 15:31:05.793605       6 cluster_manager.go:230] Reading config from path /etc/gce.conf
  <source>
    @id glbc.log
    @type tail
    format multiline
    multiline_flush_interval 5s
    format_firstline /^\w\d{4}/
    formatl /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
    time_format %m%d %H:%M:%S.%N
    path /var/log/glbc.log
    pos_file /var/log/es-glbc.log.pos
    tag glbc
  </source>

  # Example:
  # I0603 15:31:05.793605       6 cluster_manager.go:230] Reading config from path /etc/gce.conf
  <source>
    @id cluster-autoscaler.log
    @type tail
    format multiline
    multiline_flush_interval 5s
    format_firstline /^\w\d{4}/
    formatl /^(?<severity>\w)(?<time>\d{4} [^\s]*)\s+(?<pid>\d+)\s+(?<source>[^ \]]+)\] (?<message>.*)/
    time_format %m%d %H:%M:%S.%N
    path /var/log/cluster-autoscaler.log
    pos_file /var/log/es-cluster-autoscaler.log.pos
    tag cluster-autoscaler
  </source>

  # Logs from systemd-journal for interesting services.
  # TODO(random-liu): Remove this after cri container runtime rolls out.
  <source>
    @id journald-docker
    @type systemd
    filters [{ "_SYSTEMD_UNIT": "docker.service" }]
    <storage>
      @type local
      persistent true
    </storage>
    read_from_head true
    tag docker
  </source>

  <source>
    @id journald-container-runtime
    @type systemd
    filters [{ "_SYSTEMD_UNIT": "{{ container_runtime }}.service" }]
    <storage>
      @type local
      persistent true
    </storage>
    read_from_head true
    tag container-runtime
  </source>

  <source>
    @id journald-kubelet
    @type systemd
    filters [{ "_SYSTEMD_UNIT": "kubelet.service" }]
    <storage>
      @type local
      persistent true
    </storage>
    read_from_head true
    tag kubelet
  </source>

  <source>
    @id journald-node-problem-detector
    @type systemd
    filters [{ "_SYSTEMD_UNIT": "node-problem-detector.service" }]
    <storage>
      @type local
      persistent true
    </storage>
    read_from_head true
    tag node-problem-detector
  </source>

  <source>
    @id kernel
    @type systemd
    filters [{ "_TRANSPORT": "kernel" }]
    <storage>
```

```
        @type local
        persistent true
      </storage>
      <entry>
        fields_strip_underscores true
        fields_lowercase true
      </entry>
      read_from_head true
      tag kernel
    </source>

  forward.input.conf: |-
    # Takes the messages sent over TCP
    <source>
      @type forward
    </source>

  monitoring.conf: |-
    # Prometheus Exporter Plugin
    # input plugin that exports metrics
    <source>
      @type prometheus
    </source>

    <source>
      @type monitor_agent
    </source>

    # input plugin that collects metrics from MonitorAgent
    <source>
      @type prometheus_monitor
      <labels>
        host ${hostname}
      </labels>
    </source>

    # input plugin that collects metrics for output plugin
    <source>
      @type prometheus_output_monitor
      <labels>
        host ${hostname}
      </labels>
    </source>

    # input plugin that collects metrics for in_tail plugin
    <source>
      @type prometheus_tail_monitor
      <labels>
        host ${hostname}
      </labels>
    </source>

  output.conf: |-
    # Enriches records with Kubernetes metadata
    <filter kubernetes.**>
      @type kubernetes_metadata
    </filter>

    <match **>
      @id elasticsearch
      @type elasticsearch
      @log_level info
      include_tag_key true
      host elasticsearch-logging
      port 9200
      logstash_format true
      <buffer>
        @type file
        path /var/log/fluentd-buffers/kubernetes.system.buffer
        flush_mode interval
        retry_type exponential_backoff
        flush_thread_count 2
        flush_interval 5s
        retry_forever
        retry_max_interval 30
        chunk_limit_size 2M
        queue_limit_length 8
        overflow_action block
      </buffer>
    </match>
```

使用**fluentd-es-ds.yaml** 部署fluentd daemonset，部署在kube-system这个namespace下。

```
# kubectl apply -f fluentd-es-ds.yaml

# kubectl get ds -n kube-system
NAME              DESIRED   CURRENT   READY     UP-TO-DATE   AVAILABLE   NODE SELECTOR                              AGE
fluentd-es-v2.0.4   3         3         3         3            3           beta.kubernetes.io/fluentd-ds-ready=true   11d

[root@vm172-31-22-16 EFK]# kubectl get pods -n kube-system -o wide
NAME                                READY     STATUS    RESTARTS   AGE       IP           NODE
fluentd-es-v2.0.4-465rh             1/1       Running   3          11d       10.0.97.8    172.31.22.16
fluentd-es-v2.0.4-9qtvr             1/1       Running   3          11d       10.0.71.8    172.31.22.3
fluentd-es-v2.0.4-qm6bb             1/1       Running   4          11d       10.0.75.18   172.31.22.6
```

fluentd-es-ds.yaml文件如下：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: fluentd-es
  namespace: kube-system
  labels:
    k8s-app: fluentd-es
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: fluentd-es
  labels:
    k8s-app: fluentd-es
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
rules:
- apiGroups:
  - ""
  resources:
  - "namespaces"
  - "pods"
  verbs:
  - "get"
  - "watch"
  - "list"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: fluentd-es
  labels:
    k8s-app: fluentd-es
    kubernetes.io/cluster-service: "true"
```

```
    addonmanager.kubernetes.io/mode: Reconcile
subjects:
- kind: ServiceAccount
  name: fluentd-es
  namespace: kube-system
  apiGroup: ""
roleRef:
  kind: ClusterRole
  name: fluentd-es
  apiGroup: ""
---
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd-es-v2.0.4
  namespace: kube-system
  labels:
    k8s-app: fluentd-es
    version: v2.0.4
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  selector:
    matchLabels:
      k8s-app: fluentd-es
      version: v2.0.4
  template:
    metadata:
      labels:
        k8s-app: fluentd-es
        kubernetes.io/cluster-service: "true"
        version: v2.0.4
      # This annotation ensures that fluentd does not get evicted if the node
      # supports critical pod annotation based priority scheme.
      # Note that this does not guarantee admission on the nodes (#40573).
      annotations:
        scheduler.alpha.kubernetes.io/critical-pod: ''
    spec:
      priorityClassName: system-node-critical
      serviceAccountName: fluentd-es
      containers:
      - name: fluentd-es
        image: hub.kce.ksyun.com/ksyun/fluentd-elasticsearch:v2.0.4
        env:
        - name: FLUENTD_ARGS
          value: --no-supervisor -q
        resources:
          limits:
            memory: 500Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: datadockercontainers
          mountPath: /data/docker/containers
          readOnly: true
        - name: config-volume
          mountPath: /etc/fluent/config.d
      nodeSelector:
        beta.kubernetes.io/fluentd-ds-ready: "true"
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: datadockercontainers
        hostPath:
          path: /data/docker/containers
      - name: config-volume
        configMap:
          name: fluentd-es-config-v0.1.4
```

部署ElasticSearch服务

考虑到日志存储对于磁盘空间的要求比较大，这里我们建议采用金山云云硬盘作为ES的存储。

首先创建2个PV（注意：将yaml文件中的volumeId、name替换成自己的）。

```
# kubectl apply -f disk-pv-1.yaml
# kubectl apply -f disk-pv-2.yaml
```

disk-pv-1.yaml文件如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "844f520e-82fd-49c2-83e8-XXXXXXXX"
  namespace: kube-system
spec:
  capacity:
    storage: 20Gi
  storageClassName: disk
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "ksc/ebs"
    fsType: "ext4"
    options:
      volumeId: "844f520e-82fd-49c2-83e8-XXXXXXXX"
```

disk-pv-2.yaml文件如下：

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "b3de4d13-1c3a-4c88-9b64-XXXXXXXX"
  namespace: kube-system
spec:
  capacity:
    storage: 20Gi
  storageClassName: disk
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "ksc/ebs"
    fsType: "ext4"
    options:
      volumeId: "b3de4d13-1c3a-4c88-9b64-XXXXXXXX"
```

使用 **es-statefulset.yaml**、**es-service.yaml**部署ES服务

> 备注：目前金山云售卖云主机中，通用性N1、通用型N2、IO优化型I2和IO优化型I3云主机支持挂载云硬盘（详见云硬盘使用限制）。如您选择云硬盘作为存储卷，建议在创建有状态服务时，将pod调度到以上机型，否则可能存在云硬盘无法挂载的情况出现。具体调度方式参考Assigning Pods to Nodes。

```
# kubectl apply -f es-statefulset.yaml
# kubectl apply -f es-service.yaml
```

es-statefulset.yaml文件如下：

```
# RBAC authn and authz
apiVersion: v1
kind: ServiceAccount
metadata:
  name: elasticsearch-logging
  namespace: kube-system
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: elasticsearch-logging
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
rules:
- apiGroups:
  - ""
  resources:
  - "services"
  - "namespaces"
  - "endpoints"
  verbs:
  - "get"
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: kube-system
  name: elasticsearch-logging
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
subjects:
- kind: ServiceAccount
  name: elasticsearch-logging
  namespace: kube-system
  apiGroup: ""
roleRef:
  kind: ClusterRole
  name: elasticsearch-logging
  apiGroup: ""
---
# Elasticsearch deployment itself
apiVersion: apps/v1beta1
kind: StatefulSet
metadata:
  name: elasticsearch-logging
  namespace: kube-system
  labels:
    k8s-app: elasticsearch-logging
    version: v5.6.4
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  serviceName: elasticsearch-logging
  replicas: 2
  selector:
    matchLabels:
      k8s-app: elasticsearch-logging
      version: v5.6.4
  template:
    metadata:
      labels:
        k8s-app: elasticsearch-logging
        version: v5.6.4
        kubernetes.io/cluster-service: "true"
    spec:
      serviceAccountName: elasticsearch-logging
      containers:
      - image: hub.kce.ksyun.com/ksyun/elasticsearch:v5.6.4
        name: elasticsearch-logging
        resources:
          # need more cpu upon initialization, therefore burstable class
          limits:
            cpu: 1000m
          requests:
            cpu: 100m
        ports:
        - containerPort: 9200
          name: db
          protocol: TCP
        - containerPort: 9300
          name: transport
          protocol: TCP
        volumeMounts:
        - name: elasticsearch-logging
          mountPath: /data
        env:
        - name: "NAMESPACE"
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
        - name: "ES_JAVA_OPTS"
          value: "-XX:-AssumeMP -Xms2g -Xmx2g"
      # Elasticsearch requires vm.max_map_count to be at least 262144.
      # If your OS already sets up this number to a higher value, feel free
      # to remove this init container.
      initContainers:
      - image: alpine:3.6
        command: ["/sbin/sysctl", "-w", "vm.max_map_count=262144"]
        name: elasticsearch-logging-init
        securityContext:
          privileged: true
  volumeClaimTemplates:
  - metadata:
      name: elasticsearch-logging
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "disk"
      resources:
        requests:
          storage: 20Gi
```

es-service.yaml文件如下：

```
apiVersion: v1
kind: Service
metadata:
  name: elasticsearch-logging
  namespace: kube-system
  labels:
    k8s-app: elasticsearch-logging
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "Elasticsearch"
spec:
  ports:
  - port: 9200
```

```
    protocol: TCP
    targetPort: db
  selector:
    k8s-app: elasticsearch-logging
```

通过ES服务的clusterIP和port，检查ES服务是否正常。

```
# kubectl get svc --all-namespaces -o wide
NAMESPACE     NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE       SELECTOR
kube-system   elasticsearch-logging   ClusterIP   10.254.183.42    <none>        9200/TCP         11d       k8s-app=elasticsearch-logging

# curl 10.254.183.42:9200
{
  "name" : "elasticsearch-logging-1",
  "cluster_name" : "kubernetes-logging",
  "cluster_uuid" : "h3h9IPGZRsKZMaZLgOAvRw",
  "version" : {
    "number" : "5.6.4",
    "build_hash" : "8bbedf5",
    "build_date" : "2017-10-31T18:55:38.105Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

部署Kibana

使用kibana-deployment.yaml和kibana-service.yaml部署Kibana服务。

```
# kubectl apply -f kibana-deployment.yaml
# kubectl apply -f kibana-service.yaml
```

kibana-deployment.yaml文件如下：

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: kibana-logging
  namespace: kube-system
  labels:
    k8s-app: kibana-logging
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
spec:
  replicas: 1
  selector:
    matchLabels:
      k8s-app: kibana-logging
  template:
    metadata:
      labels:
        k8s-app: kibana-logging
    spec:
      containers:
      - name: kibana-logging
        image: docker.elastic.co/kibana/kibana:5.6.4
        resources:
          # need more cpu upon initialization, therefore burstable class
          limits:
            cpu: 1000m
          requests:
            cpu: 100m
        env:
          - name: ELASTICSEARCH_URL
            value: http://elasticsearch-logging:9200
          #- name: SERVER_BASEPATH
          #  value: /api/v1/namespaces/kube-system/services/kibana-logging/proxy
          - name: XPACK_MONITORING_ENABLED
            value: "false"
          - name: XPACK_SECURITY_ENABLED
            value: "false"
        ports:
        - containerPort: 5601
          name: ui
          protocol: TCP
```

kibana-service.yaml文件如下：

```
apiVersion: v1
kind: Service
metadata:
  name: kibana-logging
  namespace: kube-system
  labels:
    k8s-app: kibana-logging
    kubernetes.io/cluster-service: "true"
    addonmanager.kubernetes.io/mode: Reconcile
    kubernetes.io/name: "Kibana"
spec:
  ports:
  - port: 5601
    protocol: TCP
    targetPort: ui
  selector:
    k8s-app: kibana-logging
  type: NodePort
```

访问Kibana

kibana 第一次启动时会用较长时间(10-20分钟)来优化和 Cache 状态页面。kibana启动完成后，可以通过节点的Public IP+NodePort来访问kibana。

```
# kubectl get svc --all-namespaces -o wide
kube-system   kibana-logging          NodePort    10.254.97.159    <none>        5601:31981/TCP   11d       k8s-app=kibana-logging
```

容器日志管理

金山云容器服务（KCE）为用户提供Kubernetes集群内容器日志采集服务，并支持将采集的容器日志数据推送到金山云日志服务KLog，以满足用户收集、存储、分析和呈现业务日志信息的需求，达到提升运维效率和快速定位线上问题的目标。

用户需手动为每个Kubernetes集群启用容器日志采集服务，金山云KCE将以DaemonSet方式在集群中部署容器日志采集客户端，然后基于用户配置的容器日志采集规则，从日志源（容器标准输出日志、容器内文件日志、主机内文件日志）将数据收集并发送到日志消费端KLog进行各类统计和分析。

前提条件

- 启用容器日志采集服务会占用您集群中的部分资源（默认：CPU=0.1至1核，Memory=100至500MB；可根据需求自行调整）。
- 请确保集群内各节点可以访问日志消费端KLog。

操作步骤

启用容器日志采集服务

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**日志管理**，进入日志管理页面。

3. 按需选择**地域**和**集群**，单击**申请开通**，进行目标集群启用容器日志采集服务的配置。
4. 在弹出的**开通日志收集服务**对话框中，单击**开通**，即完成容器日志采集服务的启用。

配置日志采集规则

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**日志管理**，进入日志管理页面。
3. 按需选择**地域**和**集群**，单击**新建**，为已经启用容器日志采集服务的集群配置日志采集规则。
4. 在**新建日志采集规则**页面中，输入规则名称、选择日志类型、配置日志源和日志消费端。
   - 日志采集规则名称：必须集群内全局唯一。
   - 日志类型支持：容器标准输出、容器文件路径、主机文件路径。
     - 容器标准输出类型时：
       - 支持选择**所有容器**：此集群内所有容器。
       - 支持选择**指定负载**：按命名空间选择指定工作负载的容器。
     - 容器文件路径类型时：
       - 支持集群内任意指定容器内的文件日志。
     - 主机文件路径类型时：
       - 支持集群内任意指定节点内的文件日志，并支持用户自定义添加Label（以Key-Value形式），以便后续进行分析归类。
   - 日志源：根据选择的日志类型，进行配置。
   - 日志消费端：支持金山云日志服务KLog：**日志项目**+**日志池**。

查看和更新日志采集规则

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**日志管理**，进入日志管理页面。
3. 按需选择**地域**和**集群**，可以查看目标集群中已经配置的日志采集规则。
4. 单击右侧的**编辑采集规则**，可以进入修改页面，按需求更新日志采集规则。

事件中心

Kubernetes中的Events用于记录集群的状态变更，包括Kubernetes集群的运行和各类资源的调度情况，对用户日常观察资源的变更以及定位问题均有帮助。 容器服务事件中心能够实时将系统中产生的事件采集到日志服务的存储端，同时内置多种可视化报表，通过报表中的各类统计图表，可以迅速了解集群中发生的异常情况，并支持各种维度的筛选过滤。

操作说明

创建事件中心

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**运维管理**> **事件中心**，进入事件中心列表页。
3. 选择需要创建事件中心的集群，点击**创建事件中心**。 
4. 在创建事件中心弹窗中，根据以下提示进行事件中心配置。
   - 事件中心名称：输入要创建事件中心的名称。
   - 事件采集：按需选择项目，默认在您选择的项目中创建名称为k8s-event-{唯一标识码}的日志池。
   - 安装node-problem-detector组件：建议勾选，该组件为集群节点的故障监测组件，可以实时检测节点上的各种异常情况。
5. 单击**确认**，即可创建事件中心，并自动创建相关联的报表和告警等。

关闭事件采集

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**运维管理**> **事件中心**，选择需要关闭事件采集的集群，进入事件中心列表页。
3. 当事件采集状态为**已开启**时，点击**关闭事件采集**。 
4. 在关闭事件采集弹窗中，点击**确认**即可关闭事件采集。

   注：关闭本功能将暂停集群事件采集，即无法推送事件数据至存储端。

删除事件中心

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**运维管理**> **事件中心**，选择需要删除事件中心的集群，进入事件中心列表页。
3. 点击**删除事件中心**，在弹窗中点击**确认**即可删除。

   注：删除事件中心时会删除相关日志池，日志池内存储的数据、日志池相关联的图表和告警会被删除且不可恢复，请谨慎操作！

使用事件中心

当创建容器服务事件中心后，容器服务自动为您提供开箱即用的事件仪表盘，您即可使用容器服务事件中心。包括两大功能模块：**事件观察**和**事件告警**，其中您可以根据需求查看事件总览、异常事件观察、日志检索、事件告警等操作。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**运维管理**> **事件中心**，选择需要查看事件中心的集群，进入事件中心列表页。
3. 点击**事件中心名称**，跳转到日志服务控制台，查看该事件仪表盘。

事件观察-事件总览

在**事件总览**页面，您可根据时间、命名空间、节点、事件类型、资源类型、事件源、原因等维度过滤事件，查看核心事件的汇总统计信息，并展示一个周期内的数据对比情况。

- 事件总数及事件类型分布等情况： 
- 各类常见事件的汇总信息检索情况： 
- 异常Top事件列表：

事件观察-异常事件观察

查看某个时间段内各类异常事件分布趋势。在趋势下方展示了可供搜索的异常事件列表，帮助您快速定位到问题。 

事件观察-日志检索

点击**日志检索**，跳转到日志检索页面，方便您在日志服务控制台快速检索全部事件。 

事件告警

注：开启告警前，请先**添加告警通知模板**；在全部告警事件区域，单击**修改**，找到待开启的告警事件，单击开启图标，并选择合适的告警通知。

免费活动说明

金山云容器服务（Kingsoft Cloud Container Engine，KCE）基于原生的Kubernetes进行开发和适配，整合了金山云虚拟化、网络、存储等能力，为客户提供高可靠、高性能、高度可扩展的容器管理服务。针对容器集群日常运维复杂度高，排障困难等问题，日志服务与容器服务联合推出容器服务事件中心，借助日志服务的日志数据处理能力，用户可通过可视化图表实时查看集群事件，轻松提升容器集群运维效率。针对容器服务事件日志，日志服务提供以下免费活动：

活动对象

由容器服务控制台事件中心创建的日志池，该日志池带有**KCE**标识，用户可在日志服务控制台点击**项目列表** > **项目详情页** > **概览**，日志池列表页进行查看。 

活动规则

1. 日志服务提供免费的日志存储服务，用户只需在容器服务控制台创建事件中心，开启事件采集即可；
2. 符合活动对象的集群事件日志，日志服务全部功能均免费开放，包括存储、检索、可视化、分析、告警等一站式服务。
3. 容器服务事件中心关联的日志池免费存储周期为30天。

活动时间

提供免费服务至2022年6月30日。

监控告警概述

金山云云监控是一项针对金山云资源进行监控的服务。通过金山云云监控，您可以查询容器集群、节点、实例等维度的统计数据，实时监测集群资源使用情况、性能和运行状态。

监控

目前云监控为容器服务提供了丰富的监控指标，具体可查询容器服务监控指标。

告警

用户可以针对于关心的集群、实例、容器等监控指标，配置相应的告警规则。告警服务会及时通知您关心资源的异常情况，帮助您快速发现云资源异常并做出反应。

监控数据

金山云默认为所有用户提供云监控服务，只要使用了容器服务，云监控即可帮助你采集相关额度监控数据。

目前，仅可以通过以下方式查询容器服务相关监控指标：

- 金山云容器服务控制台获取监控数据。
- 通过API获取监控数据。

金山云容器服务控制台查询监控数据

登录金山云-容器服务控制台，即可查询对应的监控数据。

通过API获取监控数据

您可以通过云监控`GetMetricStatistics`接口来查询容器服务查询相关监控数据，详情请参考获取数据接口。

使用告警服务

当我们想检测一个服务的状态变化，需要创建告警策略来及时感知变化。

设置告警包含如下的步骤：

创建告警策略

输入策略名称，选择对应的产品类型（如容器服务-集群），设定对应的告警规则。

- 一个完整的规则，包含监控项名称、统计周期、统计方法、比较、阈值。例如：监控项名-CPU利用率、统计周期-分钟、统计方法-平均值、比较方法-> 、阈值-70%。这样的意思是，每当1分钟统计周期内，cpu利用率的平均值>70%就触发告警。
- 一个策略可以支持多个规则，任一规则触发都会发出警报。

容器服务目前支持以下对象的告警策略：

- 容器集群
- 容器集群内的实例
- 容器集群中的容器

选择关联对象

根据用户的告警需求，自定义选择对应的对象与告警规则关联。

选择告警接收人

选择对应的告警接收人，可以接收到实例触发规则后产生的警报信息。

容器服务监控指标

目前，金山云容器服务提供以下维度的监控：

1. 集群维度
2. 实例维度
3. 容器维度

集群维度监控

监控项	监控指标	单位	解释
Api server健康状态	cluster.health.apiserver.status	-	集群控制面组件Api server健康状态
Controller manager健康状态	cluster.health.controllermanager.status	-	集群控制面组件Controller manager健康状态
Scheduler健康状态	cluster.health.scheduler.status	-	集群控制面组件Scheduler健康状态
Etcd健康状态	cluster.health.etcd.status	-	集群控制面组件Etcd健康状态
集群CPU利用率	cluster.cpu.usage	%	集群内节点的平均CPU利用率
集群CPU分配率	cluster.cpu.allocation	%	集群内节点的平均CPU分配率
集群内存利用率	cluster.memory.usage	%	集群内节点的平均内存利用率
集群内存分配率	cluster.memory.allocation	%	集群内节点的平均内存分配率
集群系统盘使用率	cluster.systemdisk.usage	%	集群内节点的平均系统盘使用率

集群内服务器的具体监控指标请参考云服务器监控指标。

实例维度监控

监控项	监控指标	单位	解释
Pod CPU使用率（占节点）	pod.cpu.usage_for_node	%	Pod CPU使用占节点
Pod CPU使用率（占Request）	pod.cpu.usage_for_request	%	Pod CPU使用占Request
Pod CPU使用率（占Limit）	pod.cpu.usage_for_limit	%	Pod CPU使用占Limit
Pod CPU使用情况	pod.cpu.usage	millicores	Pod CPU使用量
Pod 内存使用率（占节点）	pod.memory.usage_for_node	%	Pod 内存使用占节点
Pod 内存使用率（占Request）	pod.memory.usage_for_request	%	Pod 内存使用占Request
Pod 内存使用率（占Limit）	pod.memory.usage_for_limit	%	Pod 内存使用占Limit

Pod 内存使用情况	pod.memory.usage	MiB	Pod 内存使用量
Pod 网络入流量	pod.network.rx_traffic	B	同一Pod内容器共享网络，Pod的网络入流量
Pod 网络出流量	pod.network.tx_traffic	B	Pod的网络出流量
Pod 网络入带宽	pod.network.rx_bandwidth	字节/秒	Pod的网络入带宽
Pod 网络出带宽	pod.network.tx_bandwidth	字节/秒	Pod的网络出带宽
Pod 网络入包量	pod.network.rx_packet	个/秒	Pod的网络入包量
Pod 网络出包量	pod.network.tx_packet	个/秒	Pod的网络出包量

容器维度监控

监控项	监控指标	单位	解释
容器CPU使用率(占节点)	container.cpu.usage_for_node	%	容器CPU使用占节点
容器CPU使用率(占Request)	container.cpu.usage_for_request	%	容器CPU使用占Request
容器CPU使用率(占Limit)	container.cpu.usage_for_limit	%	容器CPU使用占Limit
容器CPU使用情况	container.cpu.usage	millicores	容器CPU使用量
容器内存使用率(占节点)	container.memory.usage_for_node	%	容器内存使用占节点
容器内存使用率(占Request)	container.memory.usage_for_request	%	容器内存使用占Request
容器内存使用率(占Limit)	container.memory.usage_for_limit	%	容器内存使用占Limit
容器内存使用情况	container.memory.usage	MiB	容器内存使用量
容器块设备读IOPS	container.disk.read_iops	次/秒	容器磁盘读IOPS
容器块设备写IOPS	container.disk.write_iops	次/秒	容器磁盘写IOPS

配置子用户RBAC权限

概述

容器服务结合 Kubernetes 原生的 RBAC 授权策略，为您提供授权管理功能。在RBAC权限管理模式下，您可对集群内的Kubernetes 资源进行更细粒度的权限访问控制，例如：赋予某个子用户只读权限或赋予某个子用户下的某个命名空间读写权限等。

> 注：RBAC（Role-Based Access Control）是基于角色的访问控制，如需了解更多信息，请见使用RBAC鉴权。

配置说明

- 对子用户授权之前，请先确保目标子用户至少已被授予指定集群的只读权限。
- 默认情况下子用户（非集群创建者）没有集群内任何Kubernetes资源的访问权限。
- 默认**主账号**及**集群创建者**具备管理员权限。
- 您可使用容器服务提供的预设身份对目标子用户进行授权。
- 主账号可以针对所有集群维度进行一键授权。
- 当子用户给其他子用户进行RBAC授权时，控制台会过滤其可以授权的集群和命名空间的资源范围，只有当子用户有指定集群的管理员角色时，才可以给其他子用户进行RBAC授权。
- 支持对多个目标子用户进行批量添加授权。

权限说明

所有命名空间维度

角色	集群RBAC权限
管理人员（kce:admin）	对集群内所有命名空间下资源的RBAC读写权限，具备集群节点、存储卷、命名空间、配额的读写权限，可配置子用户的读写权限。
运维人员（kce:ops）	对集群内所有命名空间下控制台可见资源的RBAC读写权限，具备集群节点、存储卷、命名空间、配额的读写权限。
开发人员（kce:dev）	对集群内所有命名空间下控制台可见资源的RBAC读写权限。
受限人员（kce:restricted）	对集群内所有命名空间下控制台可见资源的RBAC只读权限。
自定义	权限由您所选择的ClusterRole决定，请在确定所选ClusterRole对各类资源的操作权限后再进行授权，以免子账号获得不符合预期的权限。

指定命名空间维度

角色	集群RBAC权限
开发人员（kce:ns:dev）	对所选命名空间下控制台可见资源的RBAC读写权限， 需要选择指定命名空间。
受限人员（kce:ns:restricted）	对所选命名空间下控制台可见资源的RBAC只读权限， 需要选择指定命名空间。

相关操作说明

管理权限

您可管理子用户的权限，为子用户增加新的权限、删除/修改已有权限。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**授权管理**，进入授权管理页面。
3. 选择需要管理权限的子用户，单击**管理权限**，进入管理权限页面。
4. 单击**添加权限**，您可按需选择集群或命名空间级别的权限配置，并选择相应的预设身份；也可以单击叉号删除目标权限。
5. 单击**完成**，即可完成相关授权操作。

添加权限

您可批量为子账号增加权限，不影响已有权限。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**授权管理**，进入授权管理页面。
3. 选择需要添加权限的子用户，单击**添加权限**，进入添加权限页面。
4. 单击**添加权限**，您可按需选择集群或命名空间级别的权限配置，并选择相应的预设身份；也可以单击叉号删除目标权限。
5. 单击**完成**，即可完成相关授权操作。

一键授权

主账号可以在所有集群维度进行一键授权。

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**授权管理**，进入授权管理页面。
3. 选择需要一键授权的子用户，单击**一键授权**。
4. 在跳出的弹窗中，按需选择单击**确定**，即可完成相关操作。

相关问题

访问接入RBAC授权模式的集群时，请参考通过kubectl连接Kubernetes集群。

升级RBAC授权模式

目前容器服务存在新旧两种授权模式，只有切换成RBAC权限管理模式，您才可对集群内 Kubernetes 资源进行更细粒度的权限控制。 若使用旧授权模式的集群需要升级时，请主账号登录控制台根据以下步骤进行升级：

1. 登录容器服务控制台。

2. 在左侧导航栏中，选择**授权管理**，进入授权管理页面。
3. 在“切换RBAC权限管理模式”弹窗中，单击**确定**即可升级授权模式。

升级说明

- 为了兼容旧模式，容器服务将会对您集群的子用户保留cluster-admin权限。
- 升级完成后需要您对集群内的子用户权限按需收回。

注： 1、升级过程中会对您的所有集群进行ClusterRoleBinding写操作，若其中某个集群访问不通，会导致升级失败，则无法使用RBAC权限管理模式进行授权。 2、升级后无法回滚到旧模式。 3、开启RBAC权限管理模式后，可以选择对该主账号下的所有子用户按需进行权限回收/更改的操作，相关操作请见详情。