目录

创建Serverless集群

相比于容器集群，Serverless集群的创建无需依赖于节点，仅需完成集群层和网络层的配置即可快速搭建一个Serverless集群。具体操作步骤如下：

1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**Serverless集群**，进入集群管理页面。
3. 点击**新建集群**，进入集群创建流程，根据实际需求进行配置：
   - 基本信息：自定义设置集群名称、选择数据中心、添加集群描述等信息。
   - Kubernetes版本：目前金山云kubernetes版本支持1.17.6、1.19.3、1.21.3，请按需选择。
   - 网络设置
     - 集群网络：选择集群所在的VPC网络，将为集群内控制节点和Pod分配此网络下的IP地址。
     - 普通子网：为集群中的控制节点选择所在子网，将占用该子网网段的3个IP。
     - 终端子网：终端子网用于为API Server创建一个内网负载均衡实例，用于集群内控制节点和虚拟节点的通信。
     - 容器CIDR：将为集群内的容器实例分配此网络地址段的IP，建议选择不同可用区和子网以保证集群和服务的高可用。
     - Service CIDR：将为集群内的Service分配此网络地址段的IP，Service CIDR不能和集群所在VPC的CIDR冲突。
     - 安全组：定义集群中启动的容器实例所属安全组。您可以通过添加安全组规则，允许或禁止安全组内的容器实例对公网或私网的访问。
4. 点击**创建**，即可完成Serverless集群创建流程。

集群生命周期管理

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**Serverless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，集群状态，集群运行区域，Kubernetes版本，集群网络，容器 CIDR，Service CIDR，容器默认安全组，集群Config，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。

删除集群

若您暂时不需要使用Serverlss集群，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**Serverless集群**，进入集群管理页面。
3. 点击**删除集群**，在出现的弹窗中，点击**确认**即可删除集群。

备注：删除集群时，集群中调度的KCI容器实例资源，通过Service/Ingress自动创建的网络资源和动态存储卷资源不会同步删除。请注意提前删除对应资源，避免不必要的计费。

通过kubectl连接Serverless集群

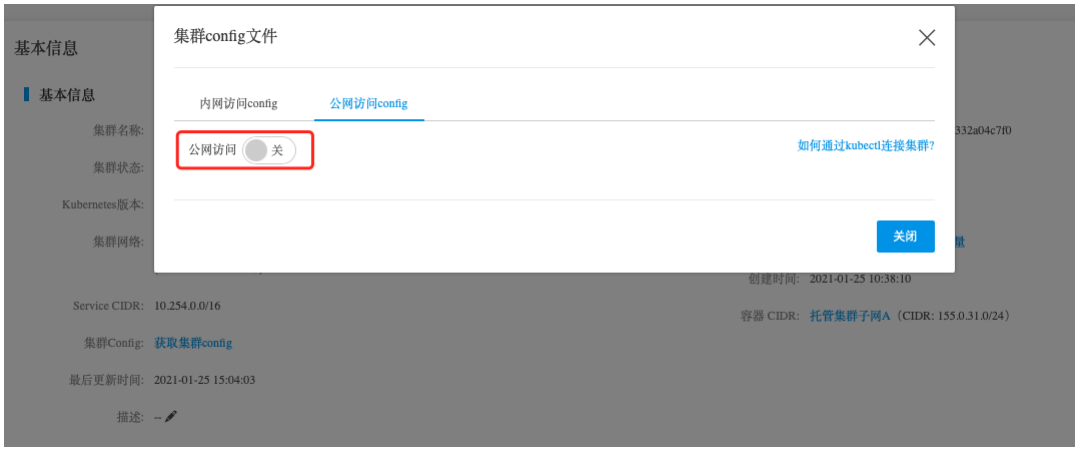
如果用户需要从本地个人计算机连接到金山云的Serverless集群，请使用 Kubernetes 命令行客户端kubectl。

安装kubectl工具

详细安装过程参考Installing and Setting up 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**Serverless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**。
5. 根据您当前所处环境（内网/公网）选择config文件，获取集群凭证信息。公网Config需开启公网访问开关后获得。



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）。

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Serverless集群，如：

```
kubectl get node
```

工作负载管理概述

Serverless集群中的工作负载

Serverless容器服务基于Kubernetes集群构建，完全兼容原生的Kubernetes资源创建与管理方式。在工作负载的创建与管理中，方式与容器集群基本一致。但因Serverless集群中不存在实际节点，以下资源与参数设置中需注意：

- Serverless集群中暂不支持创建DaemonSet类型的工作负载。
- 存储卷类型中，暂不支持主机路径和文件存储。
- 创建工作负载时，容器配置中暂不支持健康检查和设置为特权级容器。
- 工作负载访问方式中，暂不支持NodePort访问。
- 工作负载创建时会根据容器配置中的资源限制为每个Pod分配实际资源。

工作负载类型介绍

您可根据业务场景需求，在Serverless集群中创建如下类型的工作负载，详细创建流程请参考后续操作文档。

Deployment

Deployment适合于管理集群中的无状态应用，为集群内Pod和副本控制器提供声明式更新。

StatefulSet

StatefulSet 用于管理有状态应用，为创建的Pod提供持久型标识符。Pod销毁或迁移后，标识符仍会保留。创建持久化存储时，可以通过标识符实现Pod与存储卷的一一对应。

Job

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。

Cronjob

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。

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**Serverless集群**，进入Serverless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。
- 命名空间：选择Deployment所在集群的命名空间。
- 描述：创建Deployment的相关信息，用户自定义填写。

部署配置

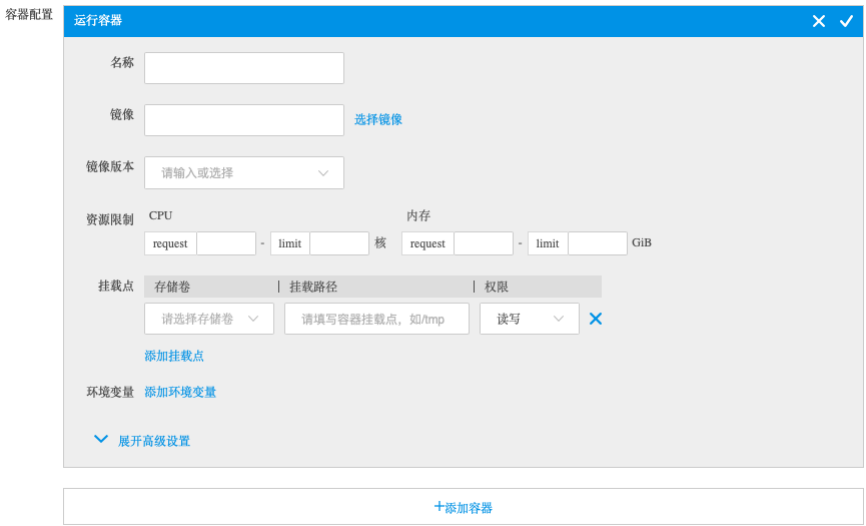
存储卷

- 类型：目前支持临时目录、金山云云硬盘、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：为Secret/Configmap类存储卷指定挂载选项。

备注：

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



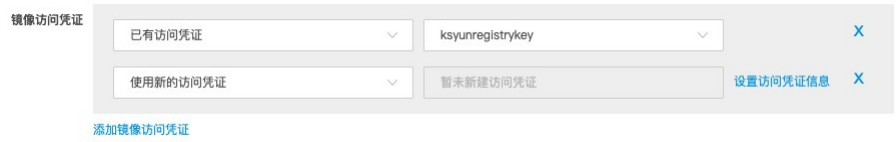
设置运行的容器信息：

- 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。
- 挂载点：当配置了存储卷时，为容器配置挂载点与读写权限。

- 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。

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。

实例数量

目前支持手动调节，直接设定实例数量。

备注：

因存储卷仅支持ReadWriteOnce的访问模式，当所选存储卷类型为EBS时，实例的最大数量为1。

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。

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**，可手动调节实例数量。

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**更多** > **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。

StatefulSet管理

概述

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**Serverless集群**，进入Serverless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。
- 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

- 类型：目前支持临时目录、金山云云硬盘、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：为Secret/Configmap类存储卷指定挂载选项。

备注：

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。

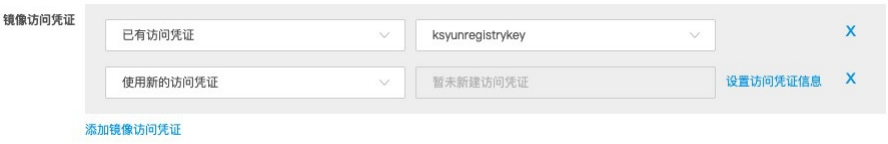
容器配置

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。
- 挂载点：当配置了存储卷时，为容器配置挂载点与读写权限。
- 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。

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。

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

更新部署

点击容器控制台左侧导航栏，进入**StatefulSet**列表页面，点击**更新**，进入更新部署页面，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。

删除StatefulSet

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

Job管理

Job概述

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**Serverless集群**，进入Serverless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。
- 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

- 类型：目前支持临时目录、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：为Secret/Configmap类存储卷指定挂载选项。

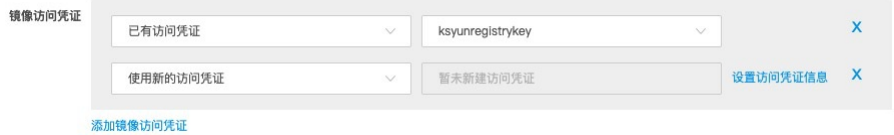
备注： 当使用云硬盘作为存储卷时，存储卷的名称为云硬盘的id，不可修改。

容器配置

- 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。
- 挂载点：当配置了存储卷时，为容器配置挂载点与读写权限。
- 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。

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

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：

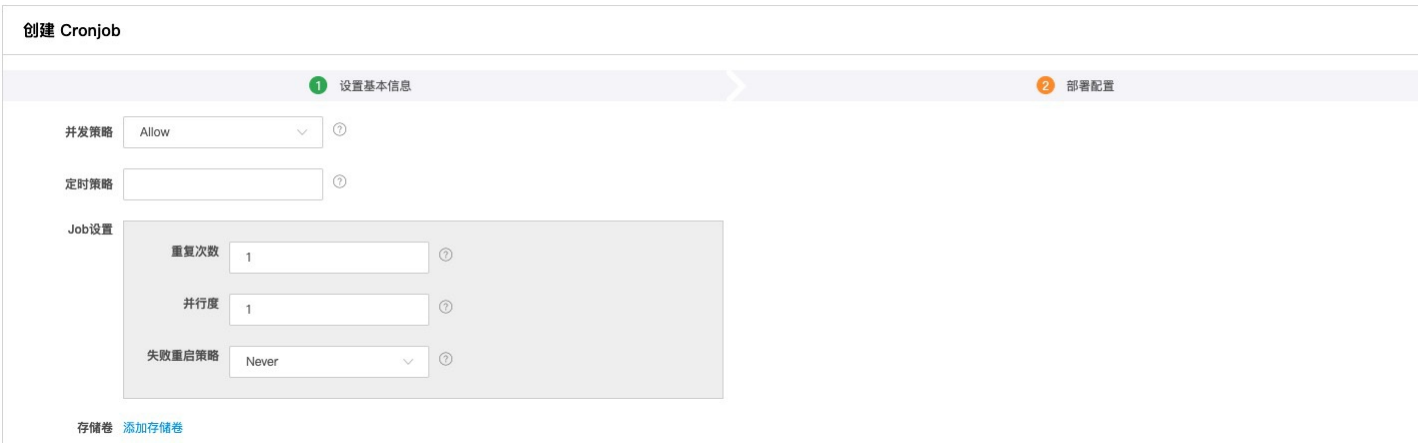
1. 登录容器服务控制台。
2. 在左侧导航栏中，选择**Serverless集群**，进入Serverless集群管理页面。
3. 选择需要新建CronJob的集群，进入该集群操作页面。
4. 选择**工作负载** > **CronJob**，进入CronJob列表页。
5. 单击页面左上角**新建**，根据实际需求完成配置，单击**创建**，即可新建CronJob。

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。
- 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。

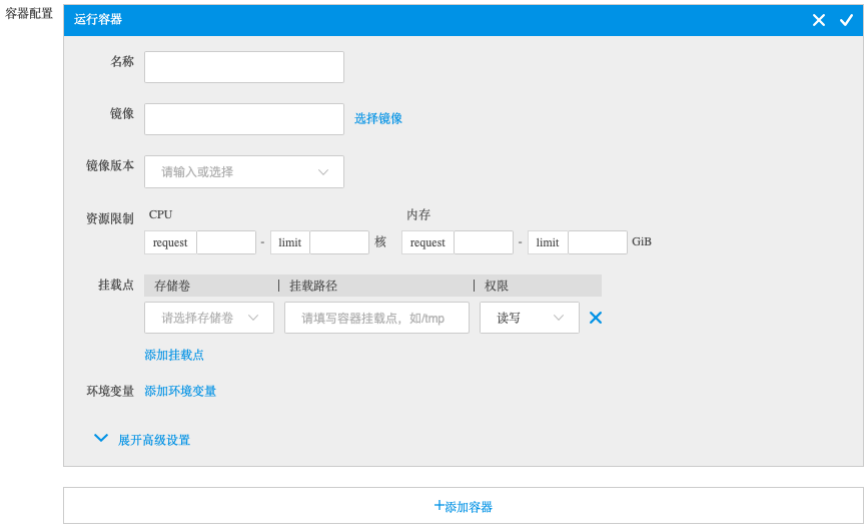
存储卷

- 类型：目前支持临时目录、金山云云硬盘、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：为Secret/Configmap类存储卷指定挂载选项。

备注：

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

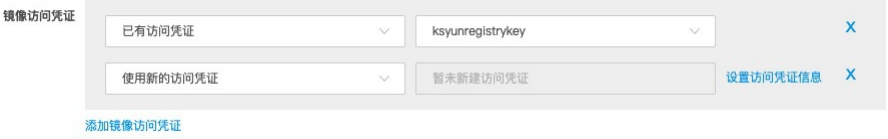


设置运行的容器信息：

- 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。
- 挂载点：当配置了存储卷时，为容器配置挂载点与读写权限。
- 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。

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
```

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组件会在固定时间周期从集群中的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**Serverless集群**，进入Serverless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：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**Serverless集群**，进入Serverless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**Serverless集群**，进入Serverless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**Serverless集群**，进入Serverless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**Serverless集群**，进入Serverless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**Serverless集群**，进入Serverless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**Serverless集群**，进入Serverless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。

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。因Serverless集群中不存在实际节点，与节点相关的访问方式受限。目前Serverless集群中服务主要支持通过ClusterIP, LoadBalancer类型暴露：

- ClusterIP：通过为Kubernetes的Service分配一个集群内部可访问的固定虚拟IP（ClusterIP），实现集群内的访问。
- LoadBalancer：Service包含的容器实例将作为负载均衡器后端的real server，支持通过金山云的负载均衡服务对外暴露，实现公网/VPC内访问。

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**Serverless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。

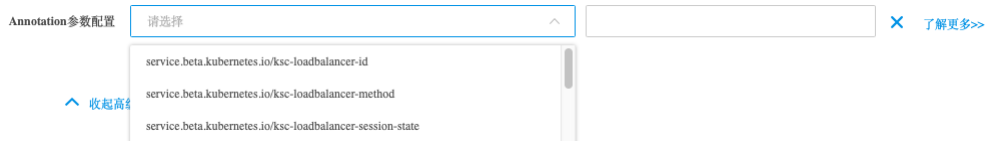
6. 设置访问方式：当前Serverless容器服务支持两种访问方式，详细配置方式将在后文进行说明。
7. 关联工作负载：可通过手动设置selector，或引用workload设置两种方式选择Service关联的工作负载。
8. 点击**创建**，即可完成Service创建。

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。Serverless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。Serverless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。
ClusterIP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。 - 创建完成的服务可以在集群内通过服务名+服务端口访问。

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：

- 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。

- 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。
- 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暴露服务

您可以使用金山云负载均衡来访问服务，以下提供Serverless集群中的部署示例。

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
```

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

Nginx-ingress支持

本文将为您介绍基于金山云容器服务搭建 Nginx-ingress 服务的操作指南。

Nginx-ingress 服务部署

使用Nginx-ingress服务的前提是在集群内部署nginx-ingress controller，此示例将在金山云serverless集群中部署Nginx-ingress 0.41.2版本。

注意：

1. serverless集群中需通过环境变量为pod声明Api server信息，部署时请将部署资源的yaml中环境变量KUBERNETES_SERVICE_HOST，KUBERNETES_SERVICE_PORT分别替换为所部属集群的Api server内网地址和服务端口。
2. 为同时满足serverless集群及API资源（admissionregistration.k8s.io/v1）的要求，请确保所建集群版本高于1.16。

执行以下文件部署Nginx-ingress服务中静态资源及Adminssion webhook服务：

```
apiVersion: v1
kind: Namespace
metadata:
  name: ingress-nginx
  labels:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx

---
# Source: ingress-nginx/templates/controller-serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx
  namespace: ingress-nginx
---
# Source: ingress-nginx/templates/controller-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller
  namespace: ingress-nginx
data:
---
# Source: ingress-nginx/templates/clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
  name: ingress-nginx
rules:
  - apiGroups:
      - ''
    resources:
      - configmaps
      - endpoints
      - nodes
      - pods
      - secrets
    verbs:
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - nodes
    verbs:
      - get
```

```
  - apiGroups:
      - ''
    resources:
      - services
    verbs:
      - get
      - list
      - update
      - watch
  - apiGroups:
      - extensions
      - networking.k8s.io   # k8s 1.14+
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - events
    verbs:
      - create
      - patch
  - apiGroups:
      - extensions
      - networking.k8s.io   # k8s 1.14+
    resources:
      - ingresses/status
    verbs:
      - update
  - apiGroups:
      - networking.k8s.io   # k8s 1.14+
    resources:
      - ingressclasses
    verbs:
      - get
      - list
      - watch
---
# Source: ingress-nginx/templates/clusterrolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
  name: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx
subjects:
  - kind: ServiceAccount
    name: ingress-nginx
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/controller-role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx
  namespace: ingress-nginx
rules:
  - apiGroups:
      - ''
    resources:
      - namespaces
    verbs:
      - get
  - apiGroups:
      - ''
    resources:
      - configmaps
      - pods
      - secrets
      - endpoints
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - services
    verbs:
      - get
      - list
      - update
      - watch
  - apiGroups:
      - extensions
      - networking.k8s.io   # k8s 1.14+
    resources:
      - ingresses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - extensions
      - networking.k8s.io   # k8s 1.14+
    resources:
      - ingresses/status
    verbs:
      - update
  - apiGroups:
      - networking.k8s.io   # k8s 1.14+
    resources:
      - ingressclasses
    verbs:
      - get
      - list
      - watch
  - apiGroups:
      - ''
    resources:
      - configmaps
    resourceNames:
      - ingress-controller-leader-nginx
    verbs:
      - get
      - update
```

```
  - apiGroups:
      - ''
    resources:
      - configmaps
    verbs:
      - create
  - apiGroups:
      - ''
    resources:
      - endpoints
    verbs:
      - create
      - get
      - update
  - apiGroups:
      - ''
    resources:
      - events
    verbs:
      - create
      - patch
---
# Source: ingress-nginx/templates/controller-rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx
  namespace: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ingress-nginx
subjects:
  - kind: ServiceAccount
    name: ingress-nginx
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/controller-service-webhook.yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller-admission
  namespace: ingress-nginx
spec:
  type: ClusterIP
  ports:
    - name: https-webhook
      port: 443
      targetPort: webhook
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/component: controller
---
# Source: ingress-nginx/templates/admission-webhooks/validating-webhook.yaml
# before changing this value, check the required kubernetes version
# https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#prerequisites
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
  name: ingress-nginx-admission
webhooks:
  - name: validate.nginx.ingress.kubernetes.io
    matchPolicy: Equivalent
    rules:
      - apiGroups:
          - networking.k8s.io
        apiVersions:
          - v1beta1
        operations:
          - CREATE
          - UPDATE
        resources:
          - ingresses
    failurePolicy: Fail
    sideEffects: None
    admissionReviewVersions:
      - v1
      - v1beta1
    clientConfig:
      service:
        namespace: ingress-nginx
        name: ingress-nginx-controller-admission
        path: /networking/v1beta1/ingresses
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/serviceaccount.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: ingress-nginx-admission
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
  namespace: ingress-nginx
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: ingress-nginx-admission
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
```

```
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
rules:
  - apiGroups:
      - admissionregistration.k8s.io
    resources:
      - validatingwebhookconfigurations
    verbs:
      - get
      - update
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/clusterrolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: ingress-nginx-admission
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: ingress-nginx-admission
subjects:
  - kind: ServiceAccount
    name: ingress-nginx-admission
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/role.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: ingress-nginx-admission
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
  namespace: ingress-nginx
rules:
  - apiGroups:
      - ''
    resources:
      - secrets
    verbs:
      - get
      - create
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/rolebinding.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: ingress-nginx-admission
  annotations:
    helm.sh/hook: pre-install,pre-upgrade,post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
  namespace: ingress-nginx
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: ingress-nginx-admission
subjects:
  - kind: ServiceAccount
    name: ingress-nginx-admission
    namespace: ingress-nginx
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/job-createSecret.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: ingress-nginx-admission-create
  annotations:
    helm.sh/hook: pre-install,pre-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
  namespace: ingress-nginx
spec:
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-instance-cpu: "1"
        k8s.ksyun.com/kci-instance-memory: "2"
      name: ingress-nginx-admission-create
      labels:
        helm.sh/chart: ingress-nginx-3.10.1
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/version: 0.41.2
        app.kubernetes.io/managed-by: Helm
        app.kubernetes.io/component: admission-webhook
    spec:
      containers:
        - name: create
          image: docker.io/jettech/kube-webhook-certgen:v1.5.0
          imagePullPolicy: IfNotPresent
          args:
            - create
            - --host=ingress-nginx-controller-admission,ingress-nginx-controller-admission.$(POD_NAMESPACE).svc
            - --namespace=$(POD_NAMESPACE)
            - --secret-name=ingress-nginx-admission
          env:
            - name: KUBERNETES_SERVICE_HOST
              value: "10.0.1.77"    # 所在集群Api server的内网地址
            - name: KUBERNETES_SERVICE_PORT
              value: "6443"  #所在集群Api server的内网服务端口
            - name: POD_NAMESPACE
              valueFrom:
```

```
              fieldRef:
                fieldPath: metadata.namespace
      restartPolicy: OnFailure
      serviceAccountName: ingress-nginx-admission
      securityContext:
        runAsNonRoot: true
        runAsUser: 2000
---
# Source: ingress-nginx/templates/admission-webhooks/job-patch/job-patchWebhook.yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: ingress-nginx-admission-patch
  annotations:
    helm.sh/hook: post-install,post-upgrade
    helm.sh/hook-delete-policy: before-hook-creation,hook-succeeded
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: admission-webhook
  namespace: ingress-nginx
spec:
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-instance-cpu: "1"
        k8s.ksyun.com/kci-instance-memory: "2"
      name: ingress-nginx-admission-patch
      labels:
        helm.sh/chart: ingress-nginx-3.10.1
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/version: 0.41.2
        app.kubernetes.io/managed-by: Helm
        app.kubernetes.io/component: admission-webhook
    spec:
      containers:
        - name: patch
          image: docker.io/jettech/kube-webhook-certgen:v1.5.0
          imagePullPolicy: IfNotPresent
          args:
            - patch
            - --webhook-name=ingress-nginx-admission
            - --namespace=$(POD_NAMESPACE)
            - --patch-mutating=false
            - --secret-name=ingress-nginx-admission
            - --patch-failure-policy=Fail
          env:
            - name: KUBERNETES_SERVICE_HOST
              value: "10.0.1.77"   # 所在集群Api server的内网地址
            - name: KUBERNETES_SERVICE_PORT
              value: "6443"  #所在集群Api server的内网服务端口
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
      restartPolicy: OnFailure
      serviceAccountName: ingress-nginx-admission
      securityContext:
        runAsNonRoot: true
        runAsUser: 2000
```

执行以下文件部署nginx-ingress controller：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: ingress-nginx
      app.kubernetes.io/instance: ingress-nginx
      app.kubernetes.io/component: controller
  revisionHistoryLimit: 10
  minReadySeconds: 0
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-instance-cpu: "1"
        k8s.ksyun.com/kci-instance-memory: "2"
      labels:
        app.kubernetes.io/name: ingress-nginx
        app.kubernetes.io/instance: ingress-nginx
        app.kubernetes.io/component: controller
    spec:
      dnsPolicy: ClusterFirst
      containers:
        - name: controller
          image: hub-cn-shanghai-2.kce.ksyun.com/aohan/ingress-nginx-controller:v0.41.2
          imagePullPolicy: IfNotPresent
          lifecycle:
            preStop:
              exec:
                command:
                  - /wait-shutdown
          args:
            - /nginx-ingress-controller
            - --election-id=ingress-controller-leader
            - --ingress-class=nginx
            - --configmap=$(POD_NAMESPACE)/ingress-nginx-controller
            - --validating-webhook=:8443
            - --validating-webhook-certificate=/usr/local/certificates/cert
            - --validating-webhook-key=/usr/local/certificates/key
          securityContext:
            capabilities:
              drop:
                - ALL
              add:
                - NET_BIND_SERVICE
            runAsUser: 101
            allowPrivilegeEscalation: true
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
            - name: LD_PRELOAD
              value: /usr/local/lib/libmimalloc.so
```

```
            - name: KUBERNETES_SERVICE_HOST
              value: "10.0.1.77"   # 所在集群Api server的内网地址
            - name: KUBERNETES_SERVICE_PORT
              value: "6443"  #所在集群Api server的内网服务端口
          livenessProbe:
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            initialDelaySeconds: 10
            periodSeconds: 10
            timeoutSeconds: 1
            successThreshold: 1
            failureThreshold: 5
          readinessProbe:
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            initialDelaySeconds: 10
            periodSeconds: 10
            timeoutSeconds: 1
            successThreshold: 1
            failureThreshold: 3
          ports:
            - name: http
              containerPort: 80
              protocol: TCP
            - name: https
              containerPort: 443
              protocol: TCP
            - name: webhook
              containerPort: 8443
              protocol: TCP
          volumeMounts:
            - name: webhook-cert
              mountPath: /usr/local/certificates/
              readOnly: true
          resources:
            requests:
              cpu: 100m
              memory: 90Mi
      nodeSelector:
        kubernetes.io/os: linux
      serviceAccountName: ingress-nginx
      terminationGracePeriodSeconds: 300
      volumes:
        - name: webhook-cert
          secret:
            secretName: ingress-nginx-admission
---

apiVersion: v1
kind: Service
metadata:
  annotations:
  labels:
    helm.sh/chart: ingress-nginx-3.10.1
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/version: 0.41.2
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/component: controller
  name: ingress-nginx-controller
  namespace: ingress-nginx
spec:
  type: LoadBalancer
  ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: 80
    - name: https
      port: 443
      protocol: TCP
      targetPort: 443
  selector:
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/component: controller
```

检查创建情况，获取nginx-ingress服务的IP地址：

```
$ kubectl get all -n ingress-nginx
NAME                                            READY   STATUS      RESTARTS   AGE
pod/ingress-nginx-admission-create-hbbtm        0/1     Completed   0          16h
pod/ingress-nginx-admission-patch-wcj94         0/1     Completed   0          26m
pod/ingress-nginx-controller-74df696cd9-d7gfh   1/1     Running     0          16h

NAME                                         TYPE           CLUSTER-IP       EXTERNAL-IP       PORT(S)                      AGE
service/ingress-nginx-controller             LoadBalancer   10.254.38.103    120.92.xx.xx    80:30734/TCP,443:31251/TCP   16h
service/ingress-nginx-controller-admission   ClusterIP      10.254.3.213     <none>            443/TCP                      16h

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/ingress-nginx-controller   1/1     1            1           16h

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/ingress-nginx-controller-74df696cd9   1         1         1       16h

NAME                                       COMPLETIONS   DURATION   AGE
job.batch/ingress-nginx-admission-create   1/1           16s        16h
job.batch/ingress-nginx-admission-patch    1/1           10s        26m
```

通过EXTERNAL-IP（120.92.xx.xx），外部流量将访问到集群中的nginx-ingress-controller，进而实现Ingress规则中的路由转发。

创建测试应用

以下创建两个应用，用于测试。

hello-world.yaml如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-instance-cpu: "1"
        k8s.ksyun.com/kci-instance-memory: "2"
      labels:
        app: hello-world
    spec:
      containers:
      - name: hello-world
        image: hub.kce.ksyun.com/kingsoft/hello-world:latest
---
```

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

hello-k8s.yaml如下：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-k8s
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-k8s
  template:
    metadata:
      annotations:
        k8s.ksyun.com/kci-instance-cpu: "1"
        k8s.ksyun.com/kci-instance-memory: "2"
      labels:
        app: hello-k8s
    spec:
      containers:
      - name: hello-k8s
        image: hub.kce.ksyun.com/kingsoft/hello-k8s:latest
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: hello-k8s
  name: hello-k8s-svc
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
  selector:
    app: hello-k8s
  type: ClusterIP
```

创建对应的deploy和service：

```
$ kubectl create -f hello-k8s.yaml
deployment.extensions/hello-k8s created
service/hello-k8s-svc created

$ kubectl create -f hello-world.yaml
deployment.extensions/hello-world created
service/hello-world-svc created

$ kubectl get deploy
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-k8s     1         1         1            1           5m2s
hello-world   1         1         1            1           4m50s

$ kubectl get svc
NAME              TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
hello-k8s-svc     ClusterIP   10.254.131.29   <none>        8080/TCP   5m31s
hello-world-svc   ClusterIP   10.254.244.96   <none>        80/TCP     5m19s
kubernetes        ClusterIP   10.254.0.1      <none>        443/TCP    52d
```

Ingress配置策略

为了支持灵活的分发策略，ingress策略可以按照多种分发方式进行配置，下面对几种常见的ingress转发策略简单介绍。

同一域名下，不同的URL的路径转发到不同服务上

这种配置常用于一个网站通过不同的路径提供不同服务的场景。

通过如下的访问配置：

- 对 http://my.nginx.test/hello-k8s 的访问将被路由到后端名为"hello-k8s-svc" 的Service。
- 对 http://my.nginx.test/hello-world 的访问将被路由到后端名为"hello-world-svc" 的Service。

ingress.yaml如下：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-test
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: my.nginx.test
    http:
      paths:
      - path: /hello-world
        backend:
          serviceName: hello-world-svc
          servicePort: 80
      - path: /hello-k8s
        backend:
          serviceName: hello-k8s-svc
          servicePort: 8080
```

创建ingress规则：

```
$ kubectl apply -f ingress.yaml
ingress.extensions/nginx-test created

$ $ kubectl get ingress
NAME         HOSTS           ADDRESS       PORTS   AGE
nginx-test   my.nginx.test   33.21.28.24   80      19s
```

备注：

- 这里我们将自有域名my.nginx.test解析到负载均衡的IP。
- Ingress规则与nginx-ingress-controller的对应关系通过注解kubernetes.io/ingress.class: nginx来指定，表示此条ingress规则由nginx-ingress-controller处理。

在浏览器的访问验证如下：


```
Hello world!
```


```
Hello k8s!
```

不同的域名转发到不同的服务

这种配置常用于一个网站通过不同的域名或者虚拟主机名提供不同的服务的场景。

通过如下的访问配置：

- 对 http://nginx.hello.k8s 的访问将被路由到后端名为"hello-k8s-svc" 的Service。
- 对 http://nginx.hello.world 的访问将被路由到后端名为"hello-world-svc" 的Service。

ingress2.yaml如下：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-test-2
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  rules:
  - host: nginx.hello.k8s
    http:
      paths:
      - path: /
        backend:
          serviceName: hello-k8s-svc
          servicePort: 8080
  - host: nginx.hello.world
    http:
      paths:
      - path: /
        backend:
          serviceName: hello-world-svc
          servicePort: 80
```

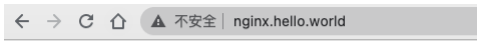
创建Ingress规则：

```
$ kubectl apply -f ingress2.yaml
ingress.extensions/nginx-test-2 created

$ kubectl get ingress
NAME           HOSTS                                 ADDRESS   PORTS   AGE
nginx-test-2   nginx.hello.k8s,nginx.hello.world               80      19s
```

在浏览器的访问验证如下：


```
Hello k8s!
```




```
Hello world!
```

HTTPS访问

当Ingress配置TLS时，服务将以HTTPS协议的方式对外暴露。

准备证书

这里我们作为测试用例使用自签名证书，使用如下命令快速创建。

```
$ 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) []:nginx.hello.k8s    #配置需使用证书的域名
Email Address []:ksyun@kingsoft.com
```

将会在当前路径下创建证书(tls.crt)和私钥文件(tls.key)。

创建Secret资源

根据现有证书和私钥创建Secret资源，将会创建一个类型为kubernetes.io/tls的Secret。

```
$ kubectl create secret tls secret-https --key tls.key --cert tls.crt
secret/secret-https created
```

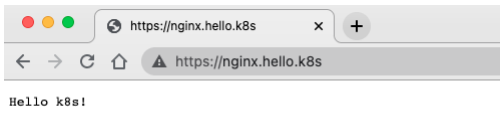
创建Ingress规则

对上述示例中的http访问方式进行升级，实现对nginx.hello.k8s的https访问。ingress-https.yaml如下：

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx-test-2
```

```
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "true"    #开启重定向功能
spec:
  rules:
  - host: nginx.hello.k8s
    http:
      paths:
      - path: /
        backend:
          serviceName: hello-k8s-svc
          servicePort: 8080
  tls:
  - hosts:
    - nginx.hello.k8s
    secretName: secret-https
```

在浏览器的访问验证如下：




```
Hello k8s!
```

更多Nginx ingress controller的特性，请参考NGINX Ingress Controller。

云硬盘存储卷

您可以在金山云Serverless容器服务中使用云硬盘存储卷。

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 KCI Pod 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、KCI Pod只有与云盘在同一个可用区（Zone）才可以挂载云盘。

5、文件系统类型（fsType）支持ext3、ext4、xfs。

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: "dblbd24b-609b-4f3d-9b16-96249a809023"
            mountPath: "/data"
      volumes:
        - name: "dblbd24b-609b-4f3d-9b16-96249a809023"
          flexVolume:
            driver: "ksc/ebs"
            fsType: "ext4"
            options:
              volumeId: "dblbd24b-609b-4f3d-9b16-96249a809023"
```

通过PV/PVC使用

定义PV。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: "dblbd24b-609b-4f3d-9b16-96249a809023"
spec:
  capacity:
    storage: 20Gi
  storageClassName: disk
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "ksc/ebs"
    fsType: "ext4"
    options:
      volumeId: "dblbd24b-609b-4f3d-9b16-96249a809023"
```

定义PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pvc-disk
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: disk
```

```
  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--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Open Api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
```

KCI实例概述

Serverless容器服务（Kingsoft Cloud Serverless Kubernetes，简称KSK）是一种基于KCI容器实例，同时兼容Kubernetes的容器服务。

KSK集群中的Pod基于金山云容器实例KCI运行在安全隔离的容器运行环境中。每个Pod容器实例底层通过轻量级虚拟化技术完全强隔离，容器实例间互不影响。

资源规格

KCI容器实例支持的规格，请参考这里

如何指定KCI Pod的规格，请参考这里

存储

支持多种类型的存储卷：

- EmptyDir
- 金山云云硬盘

网络

KCI Pod默认使用VPC网络，会占用子网中的一个网卡资源，具有以下特性：

- 集群中KCI Pod和KEC节点上的Pod、节点互联互通
- 支持Loadbalancer类型的service，对接金山云的负载均衡暴露服务
- KCI Pod支持直接挂载EIP

- 支持Ingress

监控

支持Pod实例维度和Container维度的监控，详见容器实例监控指标

KCI实例的限制

Serverless容器服务集群中没有真实的节点，部分依赖 Node、Kubelet、Kube-proxy 的功能暂不支持：

- 不支持在虚拟节点上运行DaemonSet Pod
- 不支持hostPath存储
- 不支持容器开启privileged权限
- 不支持liveness/readiness probe
- 不支持init container
- 不支持type=NodePort的Service
- 不支持Pod设置为 HostNetwork

指定KCI Pod规格

Serverless容器集群中的Pod使用KCI容器实例承载，你可以通过以下两种方式创建KCI Pod实例

- 指定KCI实例的规格
- 指定KCI实例中的容器规格，自动核算KCI Pod的规格

指定KCI实例的规格

支持通过annotation的方式指定KCI Pod的规格，详情请参考资源规格

指定KCI实例中容器的规格

用户可以通过Kubernetes中原生的方式指定容器的Request和Limit的值，由容器服务自动核算KCI Pod的规格。

目前单KCI实例中支持运行5个容器，实例内的每个容器均可以自定义配置，但汇总到KCI实例级别时需要CPU和Memory的规格约束，对于不满足的情况，金山云容器服务会自动执行规整操作，计费按照规整后CPU和Memory进行计费。

备注：

- 如果在Annotation中指定了实例的CPU和Memory，则以Annotation中配置的为准，不再进行KCI实例的规整
- 如果在Pod中未设置Request和Limit值，则未设置项作为0运算
- 如果在Pod内所有容器均没有设置Request和Limit，则Pod的规格默认为2C4GiB
- Request和Limit的合计数值需要与资源规格匹配，若设定的值和资源规格差距过大，自动规整后的规格可能导致某项资源分配超预期，造成资源的浪费，请合理设置Request和Limit值

1. 分别计算KCI Pod中容器的CPU和Memory的合计数值。

合计数值分别为Pod被所有容器的Limit之和（若没有Limit，则以Request为准）

2. 根据上述计算的数值，自动规整规则如下：

CPU和Memory的合计数值	自动规整规格
CPU和Memory的合计数值均为0	默认KCI pod的规格为2C4GiB
CPU和Memory的合计数值有一个为0	按照非0项的合计数值匹配最小规格的KCI实例 例如：CPU的合计数值为0，Memory的合计数值为7.8GiB，目前不支持7.8GiB的规格，向上规整到8GiB，在Memory为8GiB的允许的规格中选择CPU的最小值，则规整后的KCI实例的规格为2C8GiB
CPU和Memory的合计数值均不为0	需要与金山云容器实例支持的规格进行匹配。这里我们优先匹配CPU，首先匹配与CPU合计数值一致或者相近的较大规格（A规格），再匹配与Memory合计数值一致或者相近的较大规格： - 如果Memory的合计数值<A规格中Memory的最小值，则选择A规格的Memory区间的最小值 - 如果Memory的合计数值>A规格中Memory的最大值，则选择与Memory相近的较大规格（B规格），并将CPU合计数值改为B规格的CPU的最小值 - 如果CPU和Memory合计数值有一个超过允许的最大规格，则会出现错误，无法进行创建
CPU和Memory合计数值有一个超过允许的最大规格	报错，无法创建

示例

示例1

```
resources:
  limits:
    cpu: "1"
    memory: "2Gi"
  requests:
    cpu: "0.5"
    memory: "1Gi"
```

计算Limit的合计数值，此KCI Pod的规格为1C2GiB

示例2

```
# container1
resources:
  limits:
    cpu: "1"
    memory: "2Gi"
  requests:
    cpu: "0.5"
    memory: "1Gi"
# container2
resources:
  limits:
    cpu: "2C"
    memory: "4Gi"
  requests:
    cpu: "1C"
    memory: "2Gi"
```

CPU合计数值为1+2=3C，Memory的合计数值为6GiB，优先匹配CPU数值，由于CPU不支持设置为3C，自动向上规整为4C。

当CPU为4C时，支持的Memory最小值为8G大于设定的6GiB，Memory向上规整为8GiB，规整后的KCI Pod的规格为4C8GiB。

KCI Pod Annotation

Serverless容器集群支持用户通过annotation的形式，支持为Pod绑定安全组、定义Pod规格、开启Kube-proxy、为容器实例配置带宽限速等能力。

备注：

- 本文介绍的annotation仅对调度到虚拟节点上的KCI Pod生效。

- annotation需要配置在PodSpec中，而不是DeploymentSpec中。

指定KCI容器实例的规格

支持用户创建工作负载时，指定KCI容器实例的规格。

Annotation Key	是否必填	Annotation Value示例	描述
k8s.ksyun.com/kci-instance-cpu	否	1	指定容器实例CPU核数，单位：核。如填写需要和k8s.ksyun.com/kci-instance-memory注解同时填写
k8s.ksyun.com/kci-instance-memory	否	2	指定容器实例内存，单位：GiB。如填写需要和k8s.ksyun.com/kci-instance-cpu注解同时填写

示例

如下创建一个实例规格1C2GiB的工作负载，实例的规格必须满足KCI容器实例支持的实例规格。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
        annotations:
            "k8s.ksyun.com/kci-instance-cpu" : "1"
            "k8s.ksyun.com/kci-instance-memory" : "2"
        labels:
            app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: virtual-node
```

指定KCI实例的安全组

支持用户指定KCI实例所属的安全组

Annotation Key	是否必填	Annotation Value示例	描述
k8s.ksyun.com/kci-security-group-id	否	xxxxxxxx	支持填写多个，创建Serverless集群，会选择集群默认的安全组。在创建KCI容器实例时，若不指定安全组，则KCI实例默认使用创建集群时选择的安全组。如果用户希望使用同VPC下其他安全组创建KCI实例，需要通过注解方式显示指定安全组

示例

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
        annotations:
            "k8s.ksyun.com/kci-security-group-id" : "${your_security_group_id}"
        labels:
            app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: virtual-node
```

为KCI实例开启Kube-proxy

支持通过annotation定义是否为容器实例开启kube-proxy，以支持该pod具备访问clusterIP类型服务的能力。

Annotation Key	是否必填	Annotation Value示例	描述
k8s.ksyun.com/kci-kube-proxy-enabled	否	'true' / 'false'	默认值：‘false’。当为true时，为该pod开启kube-proxy，使该pod具备访问集群内clusterIP类型服务的能力；否则不开启。

示例：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
        annotations:
          "k8s.ksyun.com/kci-kube-proxy-enabled" : "true"
        labels:
          app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: virtual-node
```

为KCI实例配置带宽限速

在集群中创建资源时，可通过pod annotation为指定pod限制出入方向带宽限速值。

Annotation Key	是否必填	Annotation Value示例	描述
kubernetes.io/ingress-bandwidth	否	100M	指定容器实例入方向带宽限速值。单位支持：G、M、k。如果未填写单位，则默认对应的单位为bit。限速值支持范围1-1024Mbps，默认值为1024Mbps。
kubernetes.io/egress-bandwidth	否	100M	指定容器实例出方向带宽限速值。单位支持：G、M、k。如果未填写单位，则默认对应的单位为bit。限速值支持范围1-1024Mbps，默认值为1024Mbps。

示例

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
```

```
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
        annotations:
          kubernetes.io/ingress-bandwidth: "100M"  # 范围为1-1024Mbps
          kubernetes.io/egress-bandwidth: "200M"   # 范围为1-1024Mbps
        labels:
          app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
      nodeName: virtual-node
```

通过日志服务采集日志

概述

在Serverless集群中创建工作负载时，您可以通过环境变量配置日志采集策略，将集群内容器标准输出日志采集并推送至金山云日志服务KLog，以实现日志存储、检索分析等能力，提升运维、运营效率。

说明：

- 当前日志服务仅支持华北1（北京）地域

您可以通过以下方式配置日志采集策略：

- 通过控制台配置日志采集
- 通过yaml配置日志采集
- 更新日志采集配置

配置方式

通过在容器维度进行环境变量配置，可以将容器标准输出日志推送至日志服务指定消费端。具体配置方式如下：

通过控制台配置日志采集

创建资源同时配置日志采集：

1. 登录容器服务控制台，进入目标Serverless集群的集群操作页面。
2. 选择工作负载，点击**新建**进入新建工作负载流程。
3. 完成工作负载基本信息设置，在**部署配置**>**容器配置**中，进行对容器日志采集策略的配置：
   - 点击**开启**按钮，开启日志采集功能。
   - 选择日志服务实例，分别指定日志信息预期投递到的日志服务项目和日志池。若没有合适的日志项目和日志池，可点击**新建项目和日志池**，前往KLog控制台进行新建。

4. 完成其他部署配置及访问设置，即完成通过控制台对指定容器的日志采集配置。

若需要对已有资源开启日志采集或修改日志采集配置，通过**更新**指定工作负载即可完成。

通过yaml配置日志采集

通过yaml创建/修改资源时，通过containers环境变量声明对应的klog日志项目和日志池即可开启日志采集。

字段名	含义
KCI_KLOG_PROJECT	指定日志服务项目
KCI_KLOG_POOL	指定日志服务项目下日志池

以创建deployment为例，yaml示例如下：

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

```
  labels:
    app: busybox
spec:
  replicas: 1
  selector:
    matchLabels:
      app: busybox
  template:
    metadata:
        annotations:
            "k8s.ksyun.com/kci-instance-cpu" : "1"
            "k8s.ksyun.com/kci-instance-memory" : "2"
        labels:
            app: busybox
    spec:
      containers:
      - name: count
        image: busybox
        args: [/bin/sh, -c,
               'i=0; while true; do echo "$i: $(date)"; i=$((i+1)); sleep 1; done']
        env:
        - name: KCI_KLOG_PROJECT      #指定日志服务项目
          value: klog-test
        - name: KCI_KLOG_POOL         #指定日志服务项目下日志池
          value: kce-log
```

查看日志

1. 登录日志服务KLog控制台。
2. 在项目列表中，进入指定日志项目，左侧目录中选择**日志搜索**，根据日志搜索规则按需进行日志检索。