云容器引擎

常见问题

天翼云科技有限公司

目 录

1 高频常见问题

计费

- 云容器引擎 CCE 如何收费？

集群管理

- CCE 集群创建失败的原因与解决方法？
- 集群的管理规模和控制节点的数量有关系吗？
- 当集群状态为“不可用”时，如何排查解决？

节点及节点池

- 集群可用，但节点状态为“不可用”？
- 如何变更 CCE 急群众的节点规格？
- CCE 集群创建失败的原因与解决方法？

工作负载

- 工作负载异常：实例调度失败
- 工作负载异常：实例拉取镜像失败
- 工作负载异常：结束中，解决 Terminating 状态的 Pod 删不掉的问题
- CCE 集群中工作负载镜像的拉取策略？

网络管理

- 集群安全组规则配置
- 如何修复出现故障的容器网卡？
- 为什么容器无法连接互联网？

2 计费

2.1 云容器引擎 CCE 如何定价/收费？

计费项

云容器引擎（CCE）本身不收取任何费用，但在使用过程中会创建相关资源（如节点、带宽等），您需要为您使用的这些资源付费。CCE 相关资源的计费项分为如下两部分：

1. **集群：**控制节点资源费用，按照每个集群的类型（虚拟机或裸金属、控制节点数）、集群规模（最大支持的节点数）的差异收取不同的费用。
2. **IaaS 基础设施：**集群工作节点所使用的 IaaS 基础设施费用，包括集群创建使用过程中自动创建或手动加入的相关资源，如云服务器、云硬盘、弹性 IP/带宽、负载均衡等，价格参照相应产品价格说明

计费模式

CCE 支持按需计费、包年/包月两种计费模式，供您灵活选择。

- **按需计费：**一种先使用后付费的方式，从“开通”开启计费到“删除”结束计费，按实际购买时长计费。这种购买方式比较灵活，您可以按需取用资源，随时开启和释放，无需提前购买大量资源。

说明

关于 CCE 集群休眠或节点关机后的收费说明：

- **集群休眠**：集群休眠后，控制节点资源费用将停止收费，集群所属的云硬盘、绑定的弹性 IP、带宽等资源按各自的计费方式（“包年/包月”或“按需付费”）进行收费。
- **节点关机**：集群休眠后，集群中的工作节点（即 ECS）并不会自动关机，如需关机可勾选“关机集群下所有节点”选项。您也可以在集群休眠后自行登录 ECS 控制台将节点关机，具体请参见节点关机。

  大部分节点关机后不再收费，特殊 ECS 实例（包含本地硬盘，如磁盘增强型，超高 I/O 型等）关机后仍然正常收费，具体请参见 ECS 计费模式。

- **包年/包月：**先购买再使用的方式。这种购买方式相对于按需计费能够提供更大的折扣，对于长期使用者，推荐该方式。用户在购买时，系统会根据用户所选的机型对用户云账户中的金额进行扣除。

计费模式更改：计费周期内暂不支持计费模式更改。

须知

- 以集群作为计费量纲，根据集群类型和规模大小，按阶梯计费。
- 天翼云提供给客户进行续费与充值的时间，当您的包周期资源到期未续订或按需资源欠费时提供宽限期和保留期。

变更配置

变更须知：

- 集群中使用代金券购买的云服务器降低规格时，系统不会退还代金券。
- 升级规格配置后需按照与原规格的价差，结合已使用的时间周期，补上差价。
- 集群中的云服务器规格（CPU 或内存）变小，会影响云服务器的性能。
- 降低规格配置后，如需重新升级至原规格，可能需要补交费用。

2.2 CCE 集群的计费方式如何由按需改为包年包月？

当前在 CCE 中购买集群时支持“按需计费”和“包年/包月”（按周期）两种计费方式。按需计费的购买的集群可以转成按周期计费的集群。

约束与限制

- 仅支持默认节点池 DefaultPool 内节点转成按包周期计费，其他创建的节点池中节点不支持转包周期。
- 转成包周期的节点不支持弹性缩容。

转包周期

如果您在购买按需计费集群后，想更换为包周期计费，可按如下步骤进行操作：

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击需要转包周期集群后的  。

图2-1 转包周期

步骤 3 在转包周期页面中，选择需要转包周期的集群控制节点和工作节点。

图2-2 集群及集群下的工作节点转包周期

步骤 4 单击“确定”，等待生成订单并完成支付即可。

在支付时，如果弹出“当前用户无对该资源 API 访问权限”，请退回重新操作一次即可。

----结束

2.3 CCE 创建的节点是否支持按需转包周期？

当前在 CCE 中购买节点时支持“按需计费”和“包年/包月”（按周期）计费。

约束与限制

- 不支持在 ECS 控制台对节点转包周期。
- 仅支持默认节点池 DefaultPool 内节点转成按包周期计费，其他创建的节点池中节点不支持转包周期。
- 转成包周期的节点不支持弹性缩容。

操作步骤

如果您在购买按需计费节点后，想更换为包周期计费，可按如下步骤进行操作：

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“集群管理”。

步骤 2 单击需要转包周期集群后的。

图2-3 转包周期

步骤 3 在转包周期页面中，选择需要转包周期的集群节点。

说明

“集群转包周期”默认为勾选，如果您仅对集群下的节点进行转包周期，请取消“集群转包周期”前的勾选。

图2-4 节点转包周期

如果集群也需要转包周期，请保持“集群转包周期”前的勾选，并选择集群下需要转包周期的节点。

图2-5 集群及集群下的节点转包周期

步骤 4 单击“确定”，等待生成订单并完成支付即可。

----结束

2.4 包周期的 CCE 集群到期可以直接删除吗？

CCE 集群包周期到期后，您可以在备份好所有数据的情况下直接删除该集群。

如果到期后您仍没有续费或删除，系统会根据资源到期时间删除该集群，请及时续费并做好数据备份工作。

2.5 如何退订我的云容器引擎？

客户购买包周期资源后，支持客户退订包周期实例。退订资源实例包括资源续费部分和当前正在使用的部分，退订后资源将无法使用。

注意事项

- 退订该实例是指退订续费部分和当前正在使用的部分，资源退订后将无法使用。
- 如订单中存在主从关系的资源，需分别退订。

操作步骤

⚠ 注意

- **在执行退订操作前，请确保将退订的云资源上的数据已完成备份或者迁移，退订完成后云资源将被删除，数据无法找回，请谨慎操作。**

3. 进入天翼云官网个人中心 > 费用中心。
4. 单击左侧功能菜单“订单管理 > 退订管理”。
5. 在“退订管理”页面中查看退订资源信息，确认无误后单击操作列“退订”。

3 集群

3.1 集群创建

3.1.1 CCE 集群创建失败的原因与解决方法？

概述

本文主要介绍在 CCE 集群创建失败时，如何查找失败的原因，并解决问题。

详细信息

集群创建失败的原因包括：

1. ntpd 没安装或者安装失败、k8s 组件预校验不过、磁盘分区错误等，目前只能尝试重新创建，定位方法请参见定位失败原因。
2. 确认帐号是否欠费：帐号必须是未欠费状态才可以购买资源。

定位失败原因

您也可以参考以下步骤，通过集群日志查看集群创建失败的报错信息，然后根据相应的解决方法解决问题：

步骤 1 登录 CCE 控制台，单击集群列表上方的“操作记录”查看具体的报错信息。

步骤 2 单击“操作记录”窗口中失败状态的报错信息。

步骤 3 根据上一步获取的失败报错信息自行解决后，尝试重新创建集群。

----结束

3.1.2 集群的管理规模和控制节点的数量有关系吗？

集群管理规模是指：当前集群支持管理的最大节点数。若选择 50 节点，表示当前集群最多可管理 50 个节点。

针对不同的集群规模，控制节点的规格不同，但数量不受管理规模的影响。

集群的多控制节点模式开启后将创建三个控制节点，在单个控制节点发生故障后集群可以继续使用，不影响业务功能。

3.1.3 CCE 集群创建时的根证书如何更新？

CCE 集群根证书是 Kubernetes 认证的基础证书，云上的 Kubernetes 集群管理面托管在 CCE 管理平台上，证书也在 CCE 的管理平台上，不对用户开放，这个证书在平台上会定期维护，不会出现过期的情况。

X509 证书在 Kubernetes 集群上也是默认开启的，更新平台自动会维护更新。

获取集群证书

通过 CCE 控制台 > 集群详情页可下载集群证书，使用该证书可以访问 Kubernetes，详情请参见云容器引擎帮助中心。

3.1.4 使用 CCE 需要关注哪些配额限制？

云容器引擎 CCE 配额**只限制了集群个数**，但是使用 CCE 时也会使用其他云服务，包括：弹性云服务器、云硬盘、虚拟私有云、弹性负载均衡、容器镜像服务等。

什么是配额？

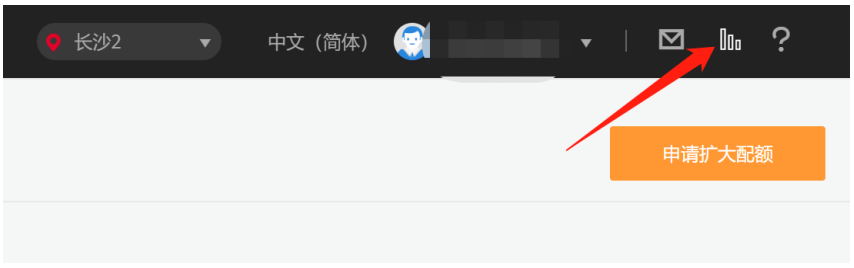
为防止资源滥用，平台限定了各服务资源的配额，对用户的资源数量和容量做了限制。如您最多可以创建多少台弹性云服务器、多少块云硬盘。

如果当前资源配额限制无法满足使用需要，您可以申请扩大配额。

怎样查看我的配额？

1. 登录管理控制台。
2. 单击管理控制台左上角的 ，选择区域和项目。
3. 在页面右上角，单击 “我的配额”。
   系统进入“服务配额”页面。

   图3-1 我的配额



4. 您可以在“服务配额”页面，查看各项资源的总配额、及使用情况。
   如果当前配额不能满足业务要求，请单击“申请扩大配额”。

如何申请扩大配额？

1. 登录管理控制台。
2. 在页面右上角，单击“我的配额”。
   系统进入“服务配额”页面。
3. 单击“申请扩大配额”。
4. 在“新建工单”页面提交工单，根据您的需求，填写相关参数。
   其中，“问题描述”项请填写需要调整的内容和申请原因。
5. 填写完毕后，勾选协议并单击“提交”。

3.2 集群运行

3.2.1 当集群状态为“不可用”时，如何排查解决？

当集群状态显示为“不可用”时，请参照如下方式来排查解决。

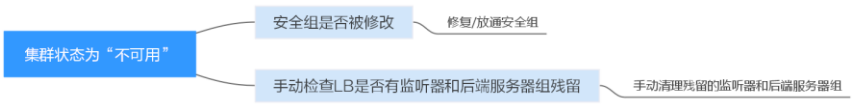
排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频率原因往低频率原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- 排查项一：安全组是否被修改
- 排查项二：手动检查 LB 是否有监听器和后端服务器组残留

图3-2 排查思路



排查项一：安全组是否被修改

步骤 1 登录控制台，选择“服务列表 > 网络 > 虚拟私有云 VPC”，单击左侧导航栏的“访问控制 > 安全组”，找到集群控制节点的安全组。

控制节点安全组名称为：集群名称-cce-**control**-编号。

步骤 2 单击安全组名称，进入详情页面，请确保集群控制节点的安全组规则的正确性。

----结束

排查项二：手动检查 LB 是否有监听器和后端服务器组残留

模拟异常状态：

创删负载均衡（LoadBalancer，简称 LB）类型 service 的任务执行时发生集群异常，恢复后会出现 service 删除成功，但是 LB 的监听器和后端服务器组残留。

步骤 1 预创建 CCE 集群，在集群内使用 nginx 官方镜像创建工作负载、预置 lb、各类型 service、ingress 等资源。

步骤 2 保持集群正常运行，nginx 负载处于稳态。

步骤 3 持续间隔每 20s 创建删除 10 个 lb 类型的 service。

步骤 4 集群出现注入异常：如 etcd 实例不可用、集群休眠等问题。

----结束

问题原因：

异常注入时正在进行创建或删除过程中的 lb-service 被删除了，但是 elb 内有监听器和后端服务器组残留。

解决方案：

可以手动清理残留的监听器和后端服务器组。

步骤 5 登录控制台，单击服务列表中“网络 > 弹性负载均衡 ELB”。

步骤 6 在负载均衡器列表中，单击对应的 ELB 名称进入详情页，在“监听器”页签下找到残留的监听器，单击后方的删除图标进行删除操作。

步骤 7 在“后端服务器组”页签下找到残留的后端服务器组，单击后方的删除图标进行删除操作。

----结束

3.2.2 CCE 集群如何重置或重装？

CCE 中的集群不能重置或重装，如确定集群无法使用，请提交工单或删除后重新购买集群。

CCE 集群中的节点重置功能已上线，详情请参见用户指南 > 节点管理。

3.2.3 如何确认已创建的集群是否为多控制节点模式？

登录 CCE 控制台，进入集群，在集群详情页面右侧查看控制节点数量：

- 3 个节点即为多控制节点模式。
- 1 个节点即为单控制节点模式。

须知

集群一旦创建，便无法更改控制节点数，需要重新创建集群才能调整。

3.2.4 是否可以直接连接集群的控制节点？

CCE 支持使用 Kubectl 工具连接集群，具体请参见用户指南 > 集群管理 > 连接集群。

CCE 不支持登录控制节点执行相关操作。

3.2.5 集群删除之后相关数据能否再次找回？

集群删除之后，部署在集群上的工作负载也会同步删除，无法恢复，请慎重删除集群。

3.2.6 如何更新 Terminating 状态的命名空间？

Kubernetes 中 namespace 有两种常见的状态，即 Active 和 Terminating 状态，其中 Terminating 状态一般会比较少见，当对应的命名空间下还存在运行的资源，但该命名空间被删除时才会出现所谓的 Terminating 状态，这种情况下只要等待 Kubernetes 本身将命名空间下的资源回收后，该命名空间将会被系统自动删除。

但是在某些情况下，即使命名空间下没有运行的资源，但依然无法删除 Terminating 状态的命名空间的情况，它会一直卡在 Terminating 状态下。

解决这个问题的步骤为：

步骤 1 查看命名空间详情

```
$ kubectl  get ns  | grep rdb
rdbms               Terminating   6d21h

$ kubectl  get ns rdbms -o yaml
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |

{"apiVersion":"v1","kind":"Namespace","metadata":{"annotations":{},"name":"rdbms"}}
  creationTimestamp: "2020-05-07T15:19:43Z"
  deletionTimestamp: "2020-05-07T15:33:23Z"
  name: rdbms
  resourceVersion: "84553454"
  selfLink: /api/v1/namespaces/rdbms
  uid: 457788ddf-53d7-4hde-afa3-1fertg21ewe1
spec:
  finalizers:
  - kubernetes
status:
  phase: Terminating
```

步骤 2 查看该命名空间下的资源

```
# 查看集群中可以使用命名空间隔离的资源
$ kubectl api-resources -o name --verbs=list --namespaced | xargs -n 1 kubectl get
--show-kind --ignore-not-found -n rdbms
```

发现 rdbms 命名空间下并无资源占用。

步骤 3 尝试对命名空间进行删除

直接删除命名空间 rdbms

```
$ kubectl  delete ns rdbms
Error from server (Conflict): Operation cannot be fulfilled on namespaces "rdbms":
The system is ensuring all content is removed from this namespace.  Upon completion,
this namespace will automatically be purged by the system.
```

提示删除操作未能完成，说系统会在确定没用资源后将会被自动删除。

步骤 4 使用强制删除

```
$ kubectl  delete ns rdbms --force --grace-period=0
warning: Immediate deletion does not wait for confirmation that the running
resource has been terminated. The resource may continue to run on the cluster
indefinitely.
Error from server (Conflict): Operation cannot be fulfilled on namespaces "rdbms":
The system is ensuring all content is removed from this namespace.  Upon completion,
this namespace will automatically be purged by the system.
```

依然无法删除该命名空间。

步骤 5 但是大部分时候，这些资源也杀不掉，解决办法是使用原生接口删除

获取 namespace 的详情信息

```
$ kubectl  get ns rdbms  -o json > rdbms.json
```

查看 namespace 定义的 json 配置，编辑 json 文件并删除掉 spec 部分。

```
$ cat rdbms.json
{
    "apiVersion": "v1",
    "kind": "Namespace",
    "metadata": {
        "annotations": {
            "kubectl.kubernetes.io/last-applied-configuration":
"{\"apiVersion\":\"v1\",\"kind\":\"Namespace\",\"metadata\":{\"annotations\":{},\"n
ame\":\"rdbms\"}}\n"
        },
        "creationTimestamp": "2019-10-14T12:17:44Z",
        "deletionTimestamp": "2019-10-14T12:30:27Z",
        "name": "rdbms",
        "resourceVersion": "8844754",
        "selfLink": "/api/v1/namespaces/rdbms",
        "uid": "29067ddf-56d7-4cce-afa3-1fbdbb221ab1"
    },
    "spec": {
        "finalizers": [
            "kubernetes"
        ]
    },
    "status": {
        "phase": "Terminating"
```

```
    }
}
```

执行接口 PUT 请求更新后，命名空间将自动删除。

```
$ curl --cacert /root/ca.crt --cert /root/client.crt --key /root/client.key -k -H
"Content-Type:application/json" -X PUT --data-binary @rdbms.json
https://x.x.x.x:5443/api/v1/namespaces/rdbms/finalize
{
  "kind": "Namespace",
  "apiVersion": "v1",
  "metadata": {
    "name": "rdbms",
    "selfLink": "/api/v1/namespaces/rdbms/finalize",
    "uid": "29067ddf-56d7-4cce-afa3-1fbdbb221ab1",
    "resourceVersion": "8844754",
    "creationTimestamp": "2019-10-14T12:17:44Z",
    "deletionTimestamp": "2019-10-14T12:30:27Z",
    "annotations": {
      "kubectl.kubernetes.io/last-applied-configuration":
"{\"apiVersion\":\"v1\",\"kind\":\"Namespace\",\"metadata\":{\"annotations\":{},\"n
ame\":\"rdbms\"}}\n"
    }
  },
  "spec": {

  },
  "status": {
    "phase": "Terminating"
  }
```

如果仍然无法删除命名空间，请查看 metadata 部分是否存在 finalizers 字段，如果存在，需要通过如下命令进入命名空间后删除该字段：

```
kubectl edit ns rdbms
```

说明

- 集群证书获取方法请参见帮助中心 > 集群管理 > 连接集群。
- **https://x.x.x.x:5443**：为连接集群的地址，登录 CCE 控制台，进入集群，在连接信息的内网地址。

步骤 6 再次查看 namespace 发现已经被删除了

```
$ kubectl  get ns  | grep rdb
```

----结束

3.2.7 为什么 CCE 集群界面的节点磁盘监控看起来不准确？

问题描述：

CCE 集群界面的某个节点磁盘监控高达 80%以上，而进入云监控界面看到的磁盘使用率在 40%不到。

后面在该节点上排查，发现有一个 pvc 磁盘使用达到了 92%，将这个盘清理后，集群界面的磁盘使用率和云监控使用率一致了。

集群界面的节点监控是怎么样的原理，是否只报最大磁盘使用率的数据呢？

问题解答：

CCE 集群监控信息中，磁盘使用率为当前节点中使用率最高的硬盘的监控信息。

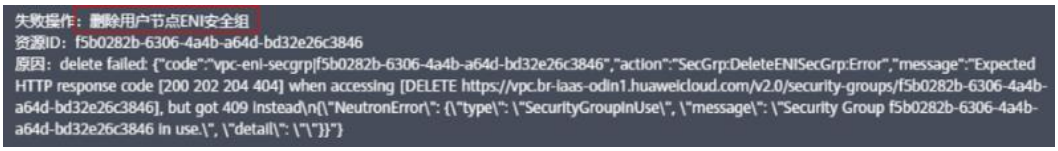
3.3 集群删除

3.3.1 集群删除失败：弹性网卡残留

CCE 在删除集群时，会连接集群的 kube-apiserver 查询集群对接的周边资源信息，例如 Turbo 集群对接的弹性网卡/弹性辅助网卡等，当 CCE 集群的状态为不可用，冻结，休眠等状态时，删除集群有可能会出现查询资源失败而导致集群删除失败的情况。

故障现象

删除集群失败。



问题根因

该场景引起的原因是连接集群的 kube-apiserver 查询集群对接的弹性网卡/弹性辅助网卡失败导致无法删除弹性网卡，CCE 创建的用于弹性网卡/弹性辅助网卡的安全组由于弹性网卡残留删除时报错了 409，最终导致了集群删除失败。

操作步骤

步骤 1 复制报错信息中的资源 ID f5b0282b-6306-4a4b-a64d-bd32e26c3846，进入到 vpc 服务的安全组界面，根据 ID 过滤安全组。

步骤 2 单击进入安全组详情界面，选择关联实例页签。

导致安全组残留的原因是关联了弹性网卡实例，辅助弹性网卡实例，单击其他页签，可以看到有残留的弹性网卡，将残留的弹性网卡（辅助弹性网卡会自动删除）删除。

步骤 3 在弹性网卡界面将上一步查询到的网卡删除。

可以用 ID 过滤需要删除的弹性网卡，也可以通过集群 ID 的名称过滤需要删除的弹性网卡，如示例中残留的集群 ID，在弹性网卡界面通过名称过滤。

步骤 4 清理完成后，到安全组确认 clusterName-cce-eni-xxx 的安全组已经没有关联的实例了，然后到 CCE 控制台就能正常删除集群了。

----结束

3.3.2 非运行中的集群删除后如何清除残留资源

处于非运行状态（例如冻结、不可用状态）中的集群，由于无法获取集群中的 PVC、Service、Ingress 等资源，因此删除集群之后可能会残留网络及存储等资源，您需要前往资源所属服务手动删除。

弹性负载均衡资源

步骤 1 前往弹性负载均衡控制台。

步骤 2 通过集群使用的 VPC ID 进行过滤，得到该虚拟私有云下所有的弹性负载均衡实例。

步骤 3 查看负载均衡实例下的监听器详情，描述中包含集群 ID、Service ID 等信息，说明该监听器由此集群创建。

步骤 4 您可以根据上述信息将集群下残留的弹性负载均衡相关资源删除。

云硬盘资源

通过 PVC 动态创建方式创建的云硬盘的 MetaData 中包含集群 ID 信息，您可以通过集群 ID 筛选出该集群中自动创建的云硬盘，根据需要进行删除。

弹性文件服务资源

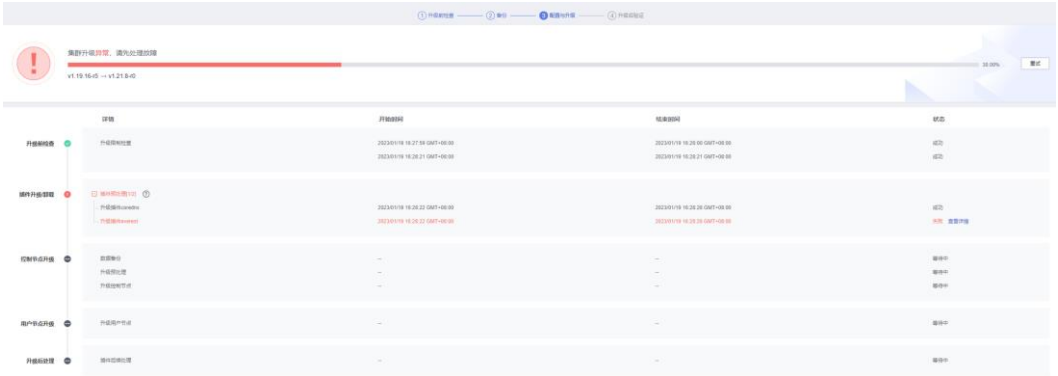
通过 PVC 动态创建方式创建的弹性文件服务 1.0 容量型实例的 MetaData 中包含集群 ID 信息，您可以通过集群 ID 筛选出该集群中自动创建的弹性文件服务 1.0 容量型实例，根据需要进行删除。

3.4 集群升级

3.4.1 CCE 集群升级时，升级集群插件失败如何排查解决？

概述

本文主要介绍在 CCE 在升级集群时，如何查找插件升级失败的原因，并解决问题。



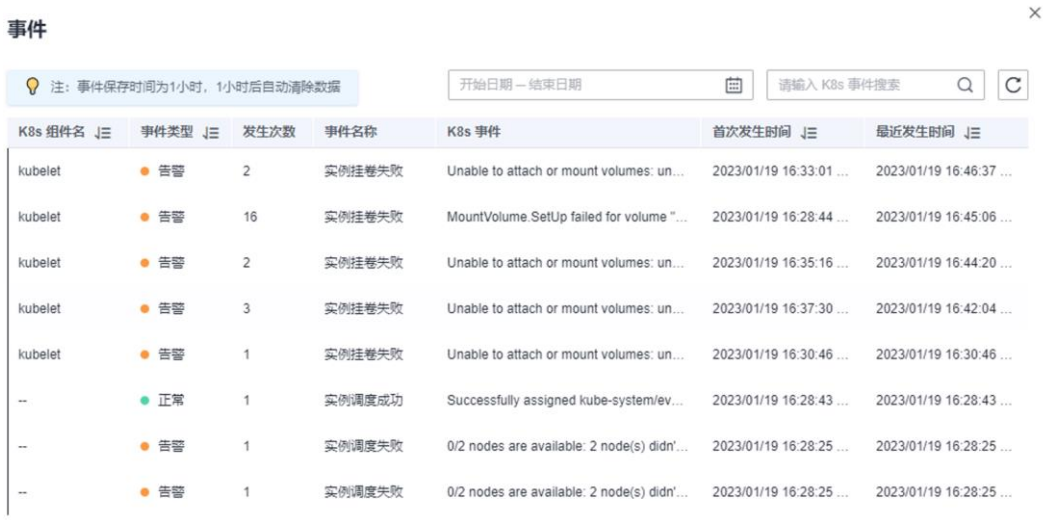
操作步骤

步骤 1 插件升级失败后，请优先进行重试。若重试不成功，则根据后续步骤排查问题。

步骤 2 在升级界面显示失败后，请退出集群升级页面，前往“插件管理”界面查看插件的详细状态。针对异常的插件，单击插件名称查看详情。

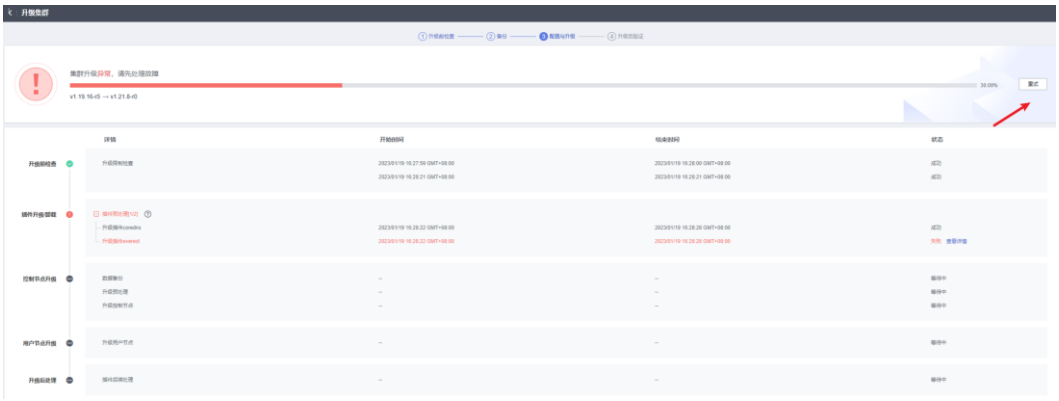
步骤 3 在插件运行实例的详情界面，单击“事件”查看异常实例的信息。

步骤 4 根据具体的异常信息进行相应处理，比如尝试删除未启动的实例让其重启等。



步骤 5 处理成功后，插件状态会变为运行中，需要保证所有插件状态都处于运行中。

步骤 6 此时进入集群升级界面，再次单击“重试”按钮即可。



----结束

4 节点

4.1 节点创建

4.1.1 CCE 集群新增节点时的问题与排查方法？

注意事项

- 同一集群下的节点镜像保证一致，后续新建/添加/纳管节点时需注意。
- 新建节点时，数据盘如需分配用户空间，分配目录注意不要设置关键目录，例如：如需放到 home 下，建议设置为/home/test，不要直接写到/home/下。

说明

请注意“挂载路径”不能设置为根目录“/”，否则将导致挂载失败。挂载路径一般设置为：

- /opt/xxxx（但不能为/opt/cloud）
- /mnt/xxxx（但不能为/mnt/paas）
- /tmp/xxx
- /var/xxx （但不能为/var/lib、/var/script、/var/paas 等关键目录）
- /xxxx（但不能和系统目录冲突，例如 bin、lib、home、root、boot、dev、etc、lost+found、mnt、proc、sbin、srv、tmp、var、media、opt、selinux、sys、usr 等）

注意不能设置为/home/paas、/var/paas、/var/lib、/var/script、/mnt/paas、/opt/cloud，否则会导致系统或节点安装失败。

排查项一：提示子网配额不足

问题现象：

CCE 集群中新增节点时无法添加新的节点，提示子网配额不足。

原因分析：

例：

VPC 网段为：192.168.66.0/24

子网网段为：192.168.66.0/24

当前 192.168.66.0/24 子网内私有 IP 已占用 251 个。

解决方法：

步骤 1 如需扩容需先扩容 VPC。

登录控制台，在服务列表中单击“虚拟私有云 VPC”，在虚拟私有云列表中找到需要扩容的 VPC，单击“操作”栏中的“编辑网段”。

如下：

步骤 2 修改子网掩码为 16 位，单击“确定”按钮。

步骤 3 单击 VPC 名称，在“基本信息”页签下单击右侧子网后的数字，在子网页面中创建新的子网规划。

步骤 4 返回 CCE 新增节点页面，选择新的子网即可创建。

说明

1. 扩容后原 VPC 内子网 192.168.66.0/24 网段正常使用不受影响。

   创建 CCE 时选择新的子网网段即可，新子网网段上限也为 251 个私有 IP，如仍无法满足业务需求，可继续新增子网。

2. 同 VPC 下不同子网间内网也是可以互信通信的。

----结束

排查项二：提示弹性 IP 不足

问题现象：

在 CCE 集群中新增节点时，在“弹性公网 IP”处选择“自动创建”，但创建节点失败，提示弹性 IP 不足。

解决方法：

您可以有两种方法解决弹性 IP 不足的问题。

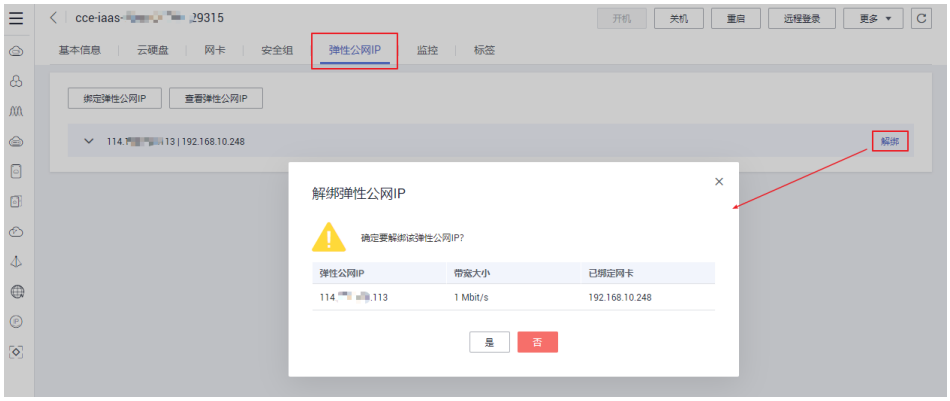
- **方法一：**解绑已绑定弹性 IP 的虚拟机，再重新添加节点。
  a. 登录控制台。

b. 选择“计算> 弹性云服务 ECS”。

c. 在弹性云服务器列表中，找到待解绑云服务器，单击云服务器名称。

d. 在打开的弹性云服务器详情页中，单击“弹性公网 IP”页签，在公网 IP 列表中单击待解绑 IP 后的“解绑”，为该云服务器解绑弹性 IP，单击“确定”。

图4-1 解绑弹性公网 IP



e. 返回 CCE 控制台新增节点页面中，选择“使用已有”重新执行新增节点的操作。

- **方法二：**提高弹性 IP 的配额。

排查项三：节点安全组是否被修改或删除

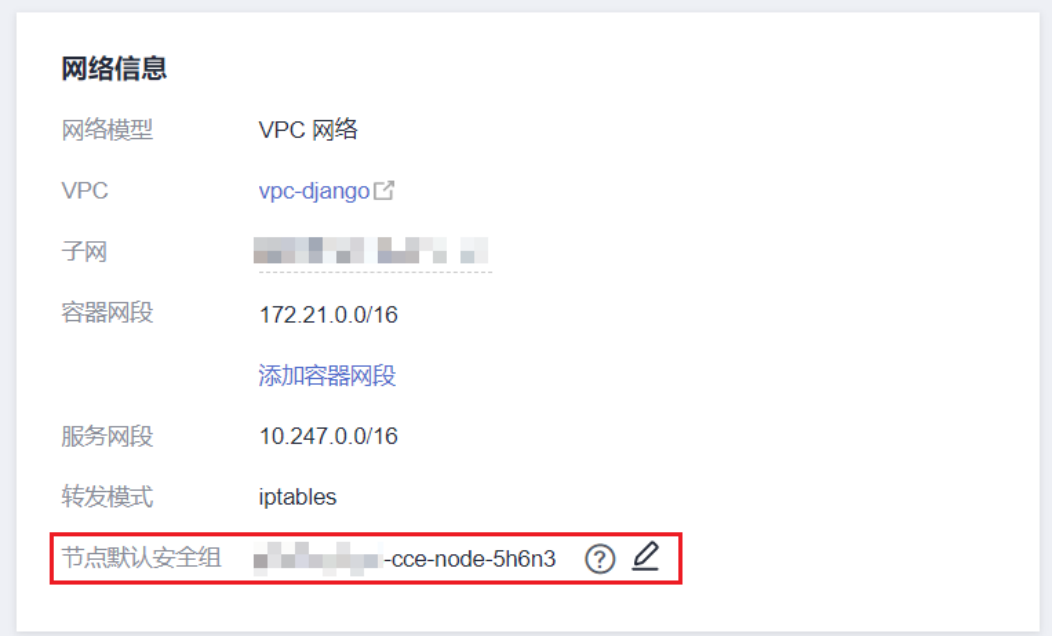
问题现象：

在 CCE 集群中新增节点时创建失败。

解决方法：

您可单击集群名称，查看“集群信息”页面。在“网络信息”中单击“节点默认安全组”后的按钮，检查集群的节点默认安全组是否被删除，且安全组规则需要满足集群安全组规则配置。

如果您的帐号下含有多个集群，需要统一管理节点的网络安全策略，您也可以指定自定义的安全组，具体操作方法及约束限制请参见更改集群节点的默认安全组。



4.1.2 CCE 集群纳管节点时的常见问题及排查方法？

概述

本文主要介绍纳管/添加已有的 ECS 实例到 CCE 集群的常见问题。

须知

- 纳管时，会将所选弹性云服务器的操作系统重置为 CCE 提供的标准镜像，以确保节点的稳定性，请选择操作系统及重置后的登录方式。
- 所选弹性云服务器挂载的系统盘、数据盘都会在纳管时被格式化，请确保信息已备份。
- 纳管过程中，请勿在弹性云服务器控制台对所选虚拟机做任何操作。

约束与限制

- 老版本控制台集群版本需 1.13 及以上，新版本控制台集群版本需 1.15 及以上。
- 暂不支持纳管 HECS（云耀云服务器）节点。
- 集群开启 IPv6 后，只支持纳管所在的子网开启了 IPv6 功能的节点；集群未开启 IPv6，只支持纳管所在的子网未开启 IPv6 功能的节点。
- 原虚拟机节点创建时若已设置密码或密钥，需等待虚拟机节点可用 10 分钟后方可纳管。
- CCE Turbo 集群要求节点支持 Sub-ENI 或可以绑定至少 16 张 ENI 网卡，具体规格请参见创建节点时控制台上可以选择的节点。

前提条件

支持纳管符合如下条件的云服务器：

- 待纳管节点必须状态为“运行中”，未被其他集群所使用，且不携带 CCE 专属节点标签 CCE-Dynamic-Provisioning-Node。
- 待纳管节点需与集群在同一虚拟私有云内（若集群版本低于 1.13.10，纳管节点还需要与 CCE 集群在同一子网内）。
- 待纳管节点需挂载数据盘，数据盘需满足至少有 1 块，容量不少于 100GB。关于节点挂载数据盘的操作说明，请参考 CCE 帮助中心。
- 待纳管节点规格要求：CPU 必须 2 核及以上，内存必须 4GB 及以上，网卡有且仅能有一个。
- 如果使用了企业项目，则待纳管节点需要和集群在同一企业项目下，不然在纳管时会识别不到资源，导致无法纳管。
- 批量纳管仅支持添加相同规格、相同可用区、相同数据盘配置的云服务器。

排查步骤

您也可以参考以下步骤，通过集群日志查看节点纳管失败的报错信息，然后根据相应的解决方法解决问题：

步骤 1 登录 CCE 控制台，单击集群列表上方的“操作记录”查看具体的报错信息。

步骤 2 单击“操作记录”窗口中失败状态的报错信息。

步骤 3 根据上一步获取的失败报错信息自行解决后，尝试重新纳管节点。

----结束

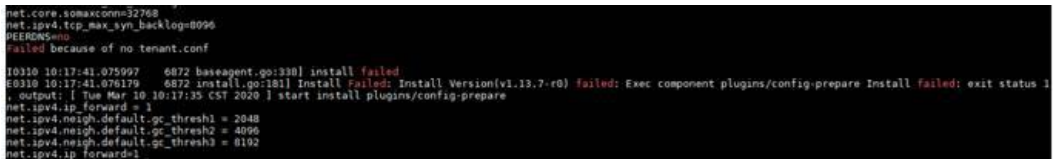
4.1.3 纳管节点时失败，报错“安装节点失败”

问题描述

节点纳管失败报错安装节点失败。

问题原因

登录节点，查看/var/paas/sys/log/baseagent/baseagent.log 安装日志，发现如下报错：



查看节点 LVM 设置，发现/dev/vdb 没有创建 LVM 逻辑卷。

解决方案

手工创建逻辑卷：

```
pvcreate /dev/vdb
vgcreate vgpaas /dev/vdb
```

然后在界面重置节点后节点状态正常。

4.2 节点运行

4.2.1 集群可用，但节点状态为“不可用”？

当集群状态为“可用”，而集群中部分节点状态为“不可用”时，请参照如下方式来排查解决。

节点不可用检测机制说明

Kubernetes 节点发送的心跳确定每个节点的可用性，并在检测到故障时采取行动。检测的机制和间隔时间详细说明请参见心跳。

使用 NPD 插件排查故障

CCE 提供节点故障检测 NPD 插件，NPD 插件从 1.16.0 版本开始增加了大量检查项，能对节点上各种资源和组件的状态检测，帮助发现节点故障。

强烈建议您安装该插件，如已安装请查看插件版本并升级到 **1.16.0 及以上**版本。

安装 NPD 插件后，当节点出现异常时，控制台上可以查看到指标异常。

您还可以在节点事件中查看到 NPD 上报的事件，根据事件信息可以定位故障。

表4-1 故障事件说明

故障事件	说明
OOMKilling	检查 oom 事件发生并上报。 可能原因：用户在 ECS 侧误操作卸载数据盘。 处理建议：排查项一：节点负载过高。
TaskHung	检查 taskHung 事件发生并上报
KernelOops	检查内核 0 指针 panic 错误
ConntrackFull	检查连接跟踪表是否满
FrequentKubeletRestart	检测 kubelet 频繁重启
FrequentDockerRestart	检测 docker 频繁重启
FrequentContainerdRestart	检测 containerd 频繁重启
CRIProblem	检查容器 CRI 组件状态
KUBELETProblem	检查 Kubelet 状态
NTPProblem	检查 ntp 服务状态
PIDProblem	检查 Pid 是否充足
FDProblem	检查文件句柄数是否充足
MemoryProblem	检查节点整体内存是否充足
CNIProblem	检查容器 CNI 组件状态
KUBEPROXYProblem	检查 Kube-proxy 状态
ReadonlyFilesystem	检查系统内核是否有 Remount root filesystem read-only 错误。 可能原因：用户在 ECS 侧误操作卸载数据盘、节点 vdb 盘被删除。 处理建议：

故障事件	说明
	• 排查项六：检查磁盘是否异常 • 排查项九：检查节点中的 vdb 盘是否被删除
DiskReadonly	检查系统盘、docker 盘、kubelet 盘是否只读 可能原因：用户在 ECS 侧误操作卸载数据盘、节点 vdb 盘被删除。 处理建议： • 排查项六：检查磁盘是否异常 • 排查项九：检查节点中的 vdb 盘是否被删除
DiskProblem	检查磁盘使用量与关键逻辑磁盘挂载 检查系统盘、docker 盘、kubelet 盘磁盘使用率，检查 docker 盘、kubelet 盘是否正常挂载在虚拟机上。
PIDPressure	检查 PID 是否充足。 处理建议：PID 不足时可调整 PID 上限。
MemoryPressure	检查容器可分配空间（allocable）内存是否充足
DiskPressure	检查 kubelet 盘和 docker 盘的磁盘使用量及 inodes 使用量。 处理建议：扩容数据盘。

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频率原因往低频率原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- 排查项一：节点负载过高
- 排查项二：弹性云服务器是否删除或故障
- 排查项三：弹性云服务器能否登录
- 排查项四：安全组是否被修改
- 排查项五：检查安全组规则中是否包含 Master 和 Node 互通的安全组策略
- 排查项六：检查磁盘是否异常
- 排查项七：内部组件是否正常
- 排查项八：DNS 地址配置错误
- 排查项九：检查节点中的 vdb 盘是否被删除
- 排查项十：排查 Docker 服务是否正常

图4-2 排查思路

排查项一：节点负载过高

问题描述：

集群中节点连接异常，多个节点报写入错误，业务未受影响。

问题定位：

步骤 1 登录 CCE 控制台，进入集群，在不可用节点所在行单击“监控”。

步骤 2 单击“监控”页签顶部的“查看更多”，前往运维管理页面查看历史监控记录。

当节点 cpu 和内存负载过高时，会导致节点网络时延过高，或系统 OOM，最终展示为不可用。

----结束

解决方案：

1. 建议迁移业务，减少节点中的工作负载数量，并对工作负载设置资源上限，降低节点 CPU 或内存等资源负载。
2. 将集群中对应的 cce 节点进行数据清理。
3. 限制每个容器的 CPU 和内存限制配额值。
4. 对集群进行节点扩容。
5. 您也可以重启节点，请至 ECS 控制台对节点进行重启。
6. 增加节点，将高内存使用的业务容器分开部署。
7. 对负载过高的节点进行重置操作。

节点恢复为可用后，工作负载即可恢复正常。

排查项二：弹性云服务器是否删除或故障

步骤 1 确认集群是否可用。

登录 CCE 控制台，确定集群是否可用。

- 若集群非可用状态，如错误等，请参见当集群状态为“不可用”时，如何排查解决？。
- 若集群状态为“运行中”，而集群中部分节点状态为“不可用”，请执行步骤 2。

步骤 2 登录 ECS 控制台，查看对应的弹性云服务器状态。

- 若弹性云服务器状态为“已删除”：请在 CCE 中删除对应节点，再重新创建节点。
- 若弹性云服务器状态为“关机”或“冻结”：请先恢复弹性云服务器，约 3 分钟后集群节点可自行恢复。
- 若弹性云服务器出现故障：请先重启弹性云服务器，恢复故障。
- 若弹性云服务器状态为“可用”：请参考排查项七：内部组件是否正常登录弹性云服务器进行本地故障排查。

----结束

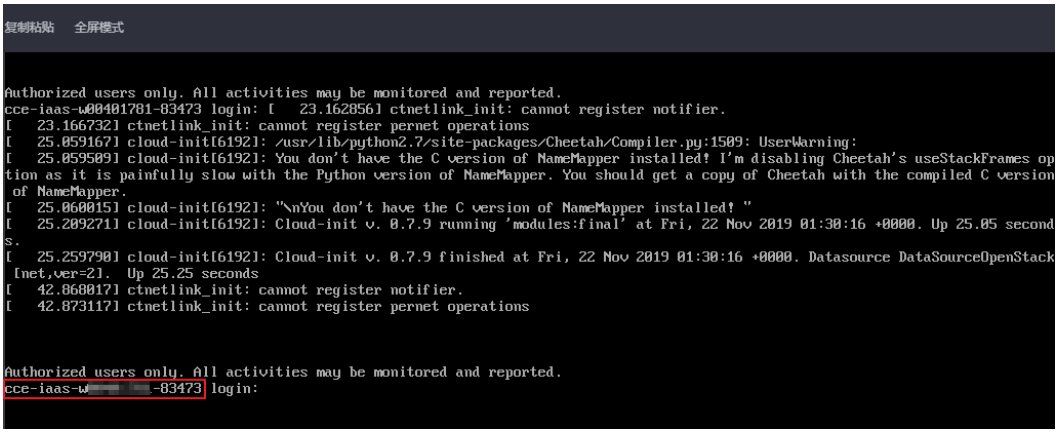
排查项三：弹性云服务器能否登录

步骤 1 登录 ECS 控制台。

步骤 2 确认界面显示的节点名称与虚机内的节点名称是否一致，并且密码或者密钥能否登录。

图4-3 确认界面显示的名称

图4-4 确认虚机内的节点名称和能否登录



如果节点名称不一致，并且密码和密钥均不能登录，说明是 ECS 创建虚机时的 cloudinit 初始化问题，临时规避可以尝试重启节点，之后再提交工单确认问题根因。

----结束

排查项四：安全组是否被修改

登录 VPC 控制台，在左侧栏目树中单击“访问控制 > 安全组”，找到集群控制节点的安全组。

控制节点安全组名称为：集群名称-cce-**control**-编号。您可以通过**集群名称**查找安全组，再进一步在名称中区分“-cce-control-”字样，即为本集群安全组。

排查安全组中规则是否被修改，安全的详细说明请参见集群安全组规则配置

排查项五：检查安全组规则中是否包含 Master 和 Node 互通的安全组策略

请检查安全组规则中是否包含 Master 和 Node 互通的安全组策略。

已有集群添加节点时，如果子网对应的 VPC 新增了扩展网段且子网是扩展网段，要在控制节点安全组（即集群名称-cce-control-随机数）中添加如下三条安全组规则，以保证集群添加的节点功能可用（新建集群时如果 VPC 已经新增了扩展网段则不涉及此场景）。

安全的详细说明请参见集群安全组规则配置

排查项六：检查磁盘是否异常

新建节点会给节点绑定一个 100G 的 docker 专用数据盘。若数据盘卸载或损坏，会导致 docker 服务异常，最终导致节点不可用。

图4-5 集群新建节点时的数据盘

请检查节点挂载的数据盘是否已被卸载。若已卸载请重新挂载数据盘，再重启节点，节点可恢复。

图4-6 磁盘检查

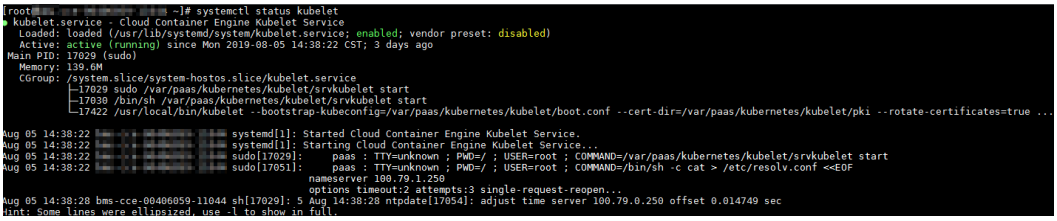
排查项七：内部组件是否正常

步骤 1 登录不可用节点对应的弹性云服务器。

步骤 2 执行以下命令判断 paas 组件是否正常。

systemctl status kubelet

执行成功，可查看到各组件的状态为 Active，如下图：



若服务的组件状态不是 Active，执行如下命令：

重启命令根据出错组件指定，如 canal 组件出错，则命令为：systemctl restart canal

重启后再查看状态：systemctl status canal

步骤 3 若执行失败，请执行如下命令，查看 monitrc 进程的运行状态。

ps -ef | grep monitrc

若存在此进程，请杀死此进程，进程杀死后会自动重新拉起。

kill -s 9 `ps -ef | grep monitrc | grep -v grep | awk '{print $2}'`

----结束

排查项八：DNS 地址配置错误

步骤 1 登录节点，在日志/var/log/cloud-init-output.log 中查看是否有域名解析失败相关的报错。

cat /var/log/cloud-init-output.log | grep resolv

如果回显包含如下内容则说明无法解析该域名。

Could not resolve host: test.obs.cn-gz1.ctyun.cn; Unknown error

步骤 2 在节点上 ping 上一步无法解析的示例域名，确认节点上能否解析此域名。

ping test.obs.cn-gz1.ctyun.cn

- 如果不能，则说明 DNS 无法解析该地址。请确认/etc/resolv.conf 文件中的 DNS 地址与配置在 VPC 的子网上的 DNS 地址是否一致，通常是由于此 DNS 地址配置错误，导致无法解析此域名。请修改 VPC 子网 DNS 为正确配置，然后重置节点。
- 如果能，则说明 DNS 地址配置没有问题，请排查其他问题。

----结束

排查项九：检查节点中的 vdb 盘是否被删除

如果节点中的 vdb 盘被删除，可参考此章节内容恢复节点。

排查项十：排查 Docker 服务是否正常

步骤 1 执行以下命令确认 docker 服务是否正在运行：

```
systemctl status docker
```

```
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2021-02-03 16:07:02 CST; 1 day 23h ago
     Docs: https://docs.docker.com
 Main PID: 3673 (dockerd)
    Tasks: 46 (limit: 24004)
   Memory: 491.2M
   CGroup: /system.slice/docker.service
           ├─3673 /usr/bin/dockerd --live-restore --log-opt max-size=50m --log-opt max-file=20 --log-driver=json-fil
           ├─3680 containerd --config /var/run/docker/containerd/containerd.toml --log-level info
           ├─5961 containerd-shim -namespace moby -workdir /var/lib/docker/containerd/daemon/io.containerd.runtime.v
           └─6811 containerd-shim -namespace moby -workdir /var/lib/docker/containerd/daemon/io.containerd.runtime.v

Warning: Journal has been rotated since unit was started. Log output is incomplete or unavailable.
```

若执行失败或服务状态非 active，请确认 docker 运行失败原因，必要时可提交工单联系技术支持。

步骤 2 执行以下命令检查当前节点上所有容器数量：

```
docker ps -a | wc -l
```

若命令卡死、执行时间过长或异常容器数过多（1000 以上），请确认外部是否存在重复不断地创删负载现象，在大量容器频繁创删过程中有可能出现大量异常容器且难以及时清理。

在此场景下可考虑停止重复创删负载或采用更多的节点去分摊负载，一般等待一段时间后节点会恢复正常，必要情况可执行 docker rm {container_id}手动清理异常容器。

----结束

4.2.2 CCE 集群中的节点无法远程登录，如何排查解决？

CCE 创建节点成功后，无法 ssh 远程登录。ssh 回显提示“所选的用户密钥未在远程主机上注册”，即 root 用户不能直接登录到节点。

出现上述问题的原因是 CCE 创建的节点安装了 cloudinit，有默认的 linux 用户，并且该密钥也是用于 linux。

解决方法

使用 linux 用户登录，使用 **sudo su** 命令切换到 root 用户。

4.2.3 如何重置 CCE 集群中节点的密码？

问题背景

在 CCE 中创建节点时，您选择了使用密钥对或者密码作为登录方式，当密钥对或密码丢失时，您可以登录 ECS 控制台对节点进行密码重置操作，重置密码后即可使用密码登录 CCE 服务中的节点。

操作步骤

步骤 1 登录 ECS 控制台。

步骤 2 在左侧弹性云服务器列表中，选择待操作节点对应的云服务器，单击后方操作列中的“更多 > 关机”。

步骤 3 待云服务器关机后，单击待操作节点后方操作列中的“更多 > 重置密码”，按照界面提示进行操作即可重置密码。

步骤 4 密码重置完成后，单击待操作节点后方操作列中的“更多 > 开机”，单击后方的“远程登录”即可通过密码登录该节点。

----结束

4.2.4 如何收集 CCE 集群中节点的日志？

CCE 节点日志文件如下表所示。

表4-2 1.21 及以上版本节点日志列表

日志名称	路径
kubelet 日志	/var/log/cce/kubernetes/kubelet.log
kube-proxy 日志	/var/log/cce/kubernetes/kube-proxy.log
everest 日志（存储）	/var/log/cce/everest-csi-driver
yangtse 日志（网络）	/var/log/cce/yangtse
canal 日志	/var/log/cce/canal
系统日志	/var/log/messages

表4-3 1.19 及以下版本节点日志列表

日志名称	路径
kubelet 日志	/var/paas/sys/log/kubernetes/kubelet.log
kube-proxy 日志	/var/paas/sys/log/kubernetes/kube-proxy.log
everest 日志（存储）	/var/log/cce/everest-csi-driver
yangtse 日志（网络）	/var/paas/sys/log/yangtse
canal 日志	/var/paas/sys/log/canal
系统日志	/var/log/messages

4.2.5 如何解决 yum update 升级操作系统导致的容器网络不可用问题？

CCE 控制台不提供针对节点的操作系统升级，也不建议您通过 yum 方式进行升级。

如果您在节点上通过 yum update 升级了操作系统，会导致容器网络的组件不可用。

您可以通过如下方式手动恢复：

须知

当前该恢复方式仅针对 EulerOS 2.2 有效。

步骤 1 root 下执行如下脚本：

```
#!/bin/bash
function upgrade_kmod()
{
   openvswicth_mod_path=$(rpm -qal openvswitch-kmod)
   rpm_version=$(rpm -qal openvswitch-kmod|grep -w openvswitch|head -1|awk -F "/"
'{print $4}')
   sys_version=`cat /boot/grub2/grub.cfg | grep EulerOS|awk 'NR==1{print $3}' | sed
's/[()]//g'`

   if [[ "${rpm_version}" != "${sys_version}" ]];then
      mkdir -p /lib/modules/"${sys_version}"/extra/openvswitch
      for path in ${openvswicth_mod_path[@]};do
         name=$(echo "$path" | awk -F "/" '{print $NF}')
         rm -f /lib/modules/"${sys_version}"/updates/"${name}"
            rm -f /lib/modules/"${sys_version}"/extra/openvswitch/"${name}"
         ln -s "${path}" /lib/modules/"${sys_version}"/extra/openvswitch/"${name}"
      done
   fi
    depmod ${sys_version}
}
```

```
upgrade_kmod
```

步骤 2 执行完成后，重启虚拟机。

----结束

4.2.6 Node 节点 vdb 盘受损，通过重置节点仍无法恢复节点？

问题现象

客户 node 节点 vdb 盘受损，通过重置节点，无法恢复节点。

问题过程：

- 在一个正常的 node 节点上，删除 lv，删除 vg，节点不可用。
- 重置异常节点，重置过程中，报语法错误，而且节点不可用。
  如下图：

问题定位

node 节点中 vg 被删除或者损坏无法识别，为了避免重置的时候误格式化用户的数据盘，需要先手动恢复 vg，这样重置的时候就不会去格式化其余的数据盘。

解决方案

步骤 1 登录节点。

步骤 2 重新创建 PV 和 VG，但是创建时报错：

```
root@host1:~# pvcreate /dev/vdb
Device /dev/vdb excluded by a filter
```

这是由于添加的磁盘是在另一个虚拟机中新建的，已经存在了分区表，当前虚拟机并不能识别磁盘的分区表，运行 parted 命令重做分区表，中途需要输入三次命令。

```
root@host1:~# parted /dev/vdb
GNU Parted 3.2
Using /dev/vdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel msdos
Warning: The existing disk label on /dev/vdb will be destroyed and all data on this
disk will be lost. Do you want to continue?
Yes/No? yes
(parted) quit
Information: You may need to update /etc/fstab.
```

再次运行 pvcreate，当询问是否擦除 dos 签名时，输入 **y**，就可以将磁盘创建为 PV。

```
root@host1:~# pvcreate /dev/vdb
WARNING: dos signature detected on /dev/vdb at offset 510. Wipe it? [y/n]: y
Wiping dos signature on /dev/vdb.
Physical volume "/dev/vdb" successfully created
```

步骤 3 创建 VG。

判断该节点的 docker 盘，如果是/dev/vdb 和/dev/vdc 两个盘，则执行下面的命令：

```
root@host1:~# vgcreate vgpaas /dev/vdb /dev/vdc
```

如果只有/dev/vdb 盘，则执行下面的命令：

```
root@host1:~# vgcreate vgpaas /dev/vdb
```

创建完成后，重置节点即可恢复。

----结束

4.2.7 CCE 集群节点中安装 kubelet 的端口主要有哪些？

CCE 集群节点中安装 kubelet 的端口主要有如下几个：

- 10250 –port：kubelet 服务监听的端口，api 会检测他是否存活。
- 10248 –healthz-port：健康检查服务的端口。
- 10255 –read-only-port：只读端口，可以不用验证和授权机制，直接访问。
- 4194 –cadvisor-port：当前节点 cadvisor 运行的端口。

4.2.8 如何配置 Pod 使用 GPU 节点的加速能力？

问题描述

我已经购买了 GPU 节点，但运行速度还是很慢，请问如何配置 Pod 使用 GPU 节点的加速能力。

解答

方案 1：

建议您将集群中 GPU 节点的不可调度的污点去掉，以便 GPU 插件驱动能够正常安装，同时您需要安装高版本的 GPU 驱动。

如果您的集群中有非 GPU 的容器，可以通过亲和、反亲和策略将这个容器不调度到 GPU 节点上。

方案 2：

建议您安装高版本的 GPU 驱动，通过 kubectl 更新 GPU 插件的配置，增加配置如下：

```
tolerations:
- operator: "Exists"
```

增加该配置后，可以使 GPU 插件驱动能够正常安装到打了污点的 GPU 节点上。

4.2.9 容器使用 SCSI 类型云硬盘偶现 IO 卡住

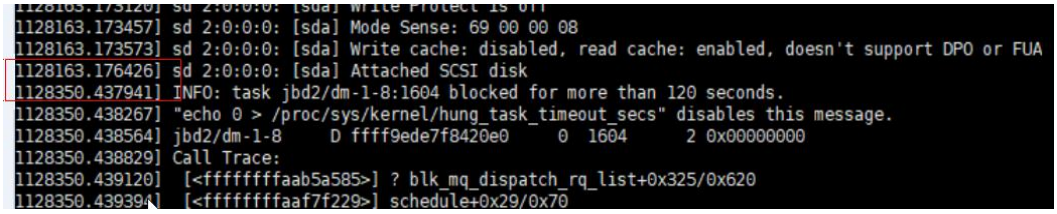
问题描述

容器使用 SCSI 类型的云硬盘存储，在 CentOS 节点上创建和删除容器触发磁盘频繁挂载卸载的场景，有概率会出现系统盘读写瞬时冲高，然后系统卡住的问题，影响节点正常工作。

出现该问题时，可在 dmesg 日志中观察到：

```
Attached SCSI disk
task jdb2/xxx blocked for more than 120 seconds.
```

如下图红框所示：



问题原理

BUS 0 上热插 PCI 设备后，Linux 内核会多次遍历挂载在 BUS 0 上的所有 PCI-Bridge，且 PCI-Bridge 在被更新期间无法正常工作。在此期间，若设备使用的 PCI-Bridge 被更新，由于内核缺陷，该设备会认为 PCI-Bridge 异常，设备进入故障模式进而无法正常工作。如果此时前端正要写 PCI 配置空间让后端处理磁盘 IO，那么这个写配置空间操作就可能会被剔除，导致后端接收不到通知去处理 IO 环上的新增请求，最终表现为前端 IO 卡住。

该问题由 linux 内核缺陷引起，详见 Linux 发行版缺陷列表：
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=51c48b310183ab6ba5419edfc6a8de889cc04521。

影响范围

对 CentOS Linux 内核 3.10.0-1127.el7 之前的版本有影响。

解决方法

通过**重置节点**将内核升级至高版本，具体请参见重置节点。

4.2.10 docker 审计日志量过大影响磁盘 IO

问题描述

部分集群版本的存量节点 docker 审计日志量较大，由于操作系统内核缺陷，会低概率出现 IO 卡住。该问题可通过优化审计日志规则，降低问题出现的概率。

影响范围

受影响的集群版本：

- v1.15.11-r1
- v.1.17.9-r0

须知

- 只需对已有节点进行修复，新建节点默认无此问题。
- 升级过程需要重启 auditd 组件。

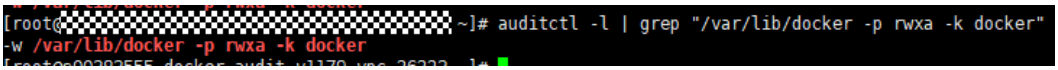
检查方法

步骤 1 以 root 用户登录 node 节点。

步骤 2 执行以下命令检查当前节点是否存在该问题：

auditctl -l | grep "/var/lib/docker -p rwxa -k docker"

如果有类似下图的回显，则说明存在该问题，需要进行修复，如果无回显则说明节点不受此问题影响。



----结束

解决方法

步骤 1 以 root 用户登录 node 节点。

步骤 2 执行如下命令：

sed -i "/\/var\/lib\/docker -k docker/d" /etc/audit/rules.d/docker.rules

service auditd restart

----结束

验证方法

执行以下命令检查问题是否修复：

auditctl -l | grep "/var/lib/docker -p rwxa -k docker"

如果无回显，则说明问题已修复。

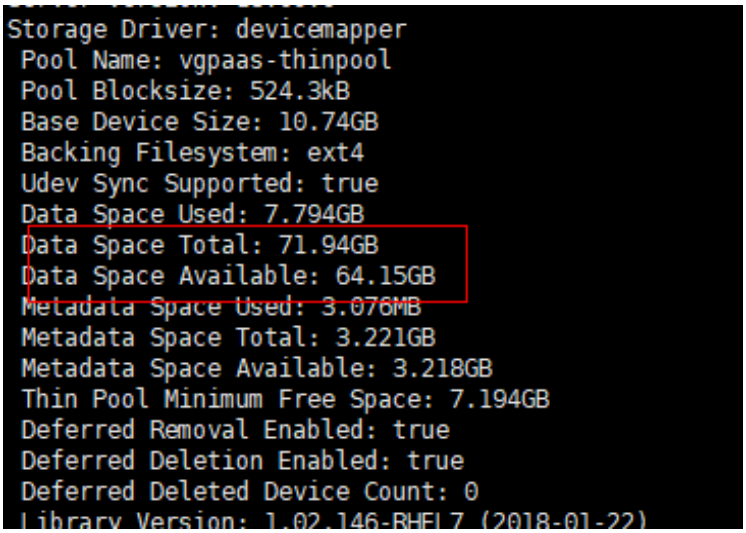
4.2.11 thinpool 磁盘空间耗尽导致容器或节点异常时，如何解决？

问题描述

当节点上的 thinpool 磁盘空间接近写满时，概率性出现以下异常：

在容器内创建文件或目录失败、容器内文件系统只读、节点被标记 disk-pressure 污点及节点不可用状态等。

用户可手动在节点上执行 docker info 查看当前 thinpool 空间使用及剩余量信息，从而定位该问题。如下图：



问题原理

docker devicemapper 模式下，尽管可以通过配置 basesize 参数限制单个容器的主目录大小（默认为 10GB），但节点上的所有容器还是共用节点的 thinpool 磁盘空间，并不是完全隔离，当一些容器使用大量 thinpool 空间且总和达到节点 thinpool 空间上限时，也会影响其他容器正常运行。

另外，在容器的主目录中创删文件后，其占用的 thinpool 空间不会立即释放，因此即使 basesize 已经配置为 10GB，而容器中不断创删文件时，占用的 thinpool 空间会不断增加一直到 10GB 为止，后续才会复用这 10GB 空间。如果节点上的**业务容器数*basesize > 节点 thinpool 空间大小**，理论上有概率出现节点 thinpool 空间耗尽的场景。

解决方案

当节点已出现 thinpool 空间耗尽时，可将部分业务迁移至其他节点实现业务快速恢复。但对于此类问题，建议采用以下方案从根因上解决问题：

方案 1：

合理规划业务分布及数据面磁盘空间，避免和减少出现**业务容器数*basesize > 节点 thinpool 空间大小**场景。

方案 2：

容器业务的创删文件操作建议在容器挂载的本地存储（如 emptyDir、hostPath）或云存储的目录中进行，这样不会占用 thinpool 空间。

方案 3：

使用 overlayfs 存储模式的操作系统，可将业务部署在此类节点上，避免容器内创删文件后占用的磁盘空间不立即释放问题。

4.2.12 节点上监听了哪些端口

表4-4 Node 节点监听端口

目的端口	协议	端口说明
10248	TCP	kubelet 健康检查端口
10250	TCP	kubelet 服务端口，提供节点上工作负载的监控信息和容器的访问通道
10255	TCP	kubelet 只读端口，提供节点上工作负载的监控信息
动态端口（与宿主机限制的范围有关，比如 内核参数 net.ipv4.ip_local_port_range）	TCP	kubelet 随机监听一个端口，与 CRI Shim 通信获取 Exec URL
10249	TCP	kube-proxy metric 端口，提供 kube-proxy 组件的监控信息
10256	TCP	kube-proxy 健康检查端口
动态端口（32768~65535）	TCP	docker exec 等功能的 websocket 监听端口
动态端口（32768~65535）	TCP	containerd exec 等功能的 websocket 监听端口
28001	TCP	icagent 本地侦听端口，接受节点 syslog 日志

目的端口	协议	端口说明
28002	TCP	icagent 健康检查端口
20101	TCP	yangtse-agent/canal-agent 健康检查端口(容器隧道网络模式涉及)
20104	TCP	yangtse-agent/canal-agent 的 metric 端口，提供组件的监控信息(容器隧道网络模式涉及)
3125	TCP	everest-csi-driver 侦听健康检查端口
3126	TCP	everest-csi-driver pprof 端口
20256	TCP	node-problem-detector server 端口
20257	TCP	node-problem-detector 对接普罗采集监控数据端口
4789	UDP	ovs 侦听端口，容器网络 vxlan 报文的传输通道(容器隧道网络模式涉及)
4789	UDPv6	ovs 侦听端口，容器网络 vxlan 报文的传输通道(容器隧道网络模式涉及)
动态端口 30000~32767	TCP	kube-proxy 侦听端口，做 4 层负载均衡。K8s 会给 NodePort 和 Loadbalancer 类型的服务分配一个随机端口，默认范围在 30000~32767
动态端口 30000~32767	UDP	kube-proxy 侦听端口，做 4 层负载均衡。K8s 会给 NodePort 和 Loadbalancer 类型的服务分配一个随机端口，默认范围在 30000~32767
123	UDP	ntpd 侦听端口，负责时间同步
20202	TCP	PodLB 侦听端口，做 7 层负载均衡，负责转发容器镜像拉取请求

4.2.13 GPU 节点使用 nvidia 驱动启动容器排查思路

集群中的节点是否有资源调度失败的事件？

问题现象：

节点运行正常且有 GPU 资源，但报如下失败信息：

0/9 nodes are aviable: 9 insufficient nvida.com/gpu

排查思路：

1. 确认节点标签是否已经打上 nvidia 资源。

2. 查看 nvidia 驱动运行是否正常。

   到插件运行所在的节点上，查看驱动的安装日志，路径如下所示：

   ```
   /opt/cloud/cce/nvidia/nvidia_installer.log
   ```

   查看 nvidia 容器标准输出日志：

   过滤容器 id

   ```
   docker ps -a | grep nvidia
   ```

   查看日志

   ```
   docker logs 容器id
   ```

业务上报 nvidia 版本和 cuda 版本不匹配？

容器中查看 cuda 的版本，执行如下命令：

```
cat /usr/local/cuda/version.txt
```

然后查看容器所在节点的 nvidia 驱动版本支持的 cuda 版本范围，是否包含容器中的 cuda 版本。

4.2.14 节点 NTP 时间不同步

故障现象

节点上的 ntpd 在长时间无法连接 ntpserver 等特殊场景下，可能导致偏移量过大，无法自动恢复。

问题根因

EulerOS 和 CentOS 类型的节点存在由 NTP 引起的已知问题，其他类型的节点不涉及该问题。

说明

上述问题在 v1.19.16-r7、v1.21.9-r10、v1.23.7-r10 版本的集群中被修复。

解决办法

- 若您的集群版本为 v1.19.16-r7、v1.21.9-r10、v1.23.7-r10 及以上，该版本的节点已经切换至 chronyd 时间同步，请将节点重置为最新版本的操作系统即可修复该问题。
- 若您的集群版本不满足要求，建议您将集群升级至 v1.19.16-r7、v1.21.9-r10、v1.23.7-r10 及以上版本，再将节点重置为最新版本的操作系统。

4.3 规格配置变更

4.3.1 如何变更 CCE 集群中的节点规格？

操作方法

⚠ 注意

如果需要变更规格的节点是纳管到集群中的，可将节点从 CCE 集群中移除后再变更节点规格，避免影响业务。

步骤 1 登录 CCE 控制台，进入集群，在左侧选择“节点管理”，在右侧单击节点名称，跳转到弹性云服务器详情页。

步骤 2 在弹性云服务器详情页中，单击右上角的“关机”，关机完成后单击“更多 > 变更规格”。

步骤 3 在“云服务器变更规格”页面中根据业务需求选择相应的规格，单击“提交”完成节点规格的变更，返回弹性云服务器列表页，将该云服务器执行“开机”操作。

步骤 4 登录 CCE 控制台，进入集群，在节点管理列表中找到该节点，并单击操作栏中的“同步云服务器”，同步后即可看到节点规格已与弹性云服务器中变更的规格一致。

----结束

4.3.2 CCE 节点变更规格后，为什么无法重新拉起或创建工作负载？

问题背景

kubelet 启动参数中默认将 CPU Manager 的策略设置为 static，允许为节点上具有某些资源特征的 pod 赋予增强的 CPU 亲和性和独占性。用户如果直接在 ECS 控制台对 CCE 节点变更规格，会由于变更前后 CPU 信息不匹配，导致节点上的负载无法重新拉起，也无法创建新负载。

更多信息请参见 Kubernetes 控制节点上的 CPU 管理策略。

影响范围

集群开启了 CPU 管理策略的集群。

解决方案

步骤 1 登录 CCE 节点（弹性云服务器）并删除 **cpu_manager_state** 文件。

删除命令示例如下：

```
rm -rf /mnt/paas/kubernetes/kubelet/cpu_manager_state
```

步骤 2 重启节点或重启 kubelet，重启 kubelet 的方法如下：

```
systemctl restart kubelet
```

步骤 3 此时重新拉起或创建工作负载，已可成功执行。

----结束

4.3.3 CCE 集群的节点可以更改 IP 吗？

- 不支持修改私网 IP：当前 CCE 的集群中把节点私网 IP 作为 kubernetes node 名称，kubernetes node 名称不支持修改，修改后会导致节点不可用，所以不支持更换节点私网 IP。
- 支持修改公网 IP：节点上的公网 IP 可以在 ECS 控制台更换。

修改节点私网 IP 后如何恢复

节点私网 IP 修改后，会导致节点不可用。这时您需要将节点的私网 IP 修改回原来使用的 IP。

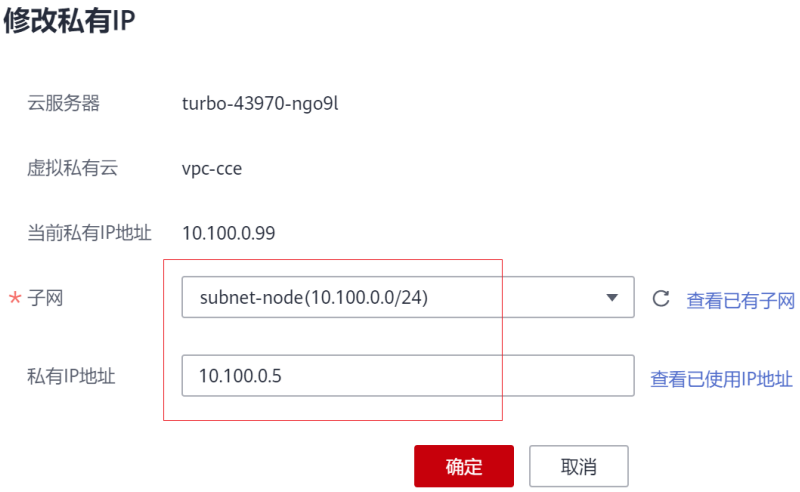
步骤 1 在 CCE 控制台，查看节点详情，找到该节点之前使用的 IP 和子网。

图4-7 节点私网 IP 地址和所在子网

步骤 2 登录 ECS 控制台，找到节点，先关机，然后进入节点详情页，在弹性网卡页签修改私网 IP。注意此时要选择对应的子网。

图4-8 修改私有 IP

图4-9 修改私有 IP



步骤 3 修改完成后再次开机。

----结束

4.4 节点内核

4.4.1 低版本内核的 CentOS 节点反复创删应用时，偶现 cgroup kmem 泄露问题

故障现象

CentOS 7.6 节点内核低于 3.10.0-1062.12.1.el7.x86_64 的场景下（主要为 1.17.9 版本集群），反复创建应用时出现 cgroup kmem 泄露，导致节点内存有空余，但是无法创建新的节点，并提示报错 Cannot allocate memory。

问题根因

在反复创建应用时会创建的临时 memory cgroup，但在应用删除时，内核已经删除了 cgroup （/sys/fs/cgroup/memory 下对应的 cgroup 目录已经删除）， 但在内核中没有释放 cssid，导致内核认为的 cgroup 的数量实际数量不一致，残留的 cgroup 达到节点上限后，导致该节点无法继续新建 Pod。

解决方法

- 该问题可以通过可以在内核层全局使用 “cgroup.memory=nokmem” 参数关闭 kmem 使用防止发生泄漏。
- 1.17 集群版本已停止维护，修复该问题建议升级至 1.19 及以上集群版本，并通过节点重置为最新版本的操作系统修复该问题，确保内核版本高于 3.10.0-1062.12.1.el7.x86_64。

4.4.2 CCE 集群 IPVS 转发模式下 conn_reuse_mode 问题说明

问题说明

CCE 集群在 IPVS 模式下，通过 Service 方式访问集群内部服务，偶现 1 秒延时的情况，引起该问题的主要原因为社区 IPVS 连接复用 Bug。

IPVS 连接复用参数说明

IPVS 对端口的复用策略主要由内核参数 net.ipv4.vs.conn_reuse_mode 决定。

1. 当 net.ipv4.vs.conn_reuse_mode=0 时，IPVS 不会对新连接进行重新负载，而是复用之前的负载结果，将新连接转发到原来的 RS（IPVS 的后端）上。
2. 当 net.ipv4.vs.conn_reuse_mode=1 时，IPVS 则会对新连接进行重新负载。

IPVS 连接复用参数带来的问题

- **问题 1**

  当 net.ipv4.vs.conn_reuse_mode=0 时，对于端口复用的连接，IPVS 不会主动进行新的调度，也并不会触发结束连接或 DROP 操作，新连接的数据包会被直接转发到之前使用的 RS。如果此时后端 pod 已经被删除就会出现异常。根据当前的实现逻辑，只要不断的有端口复用的连接请求发来，RS 就不会被 kube-proxy 删除。

- **问题 2**

  当 net.ipv4.vs.conn_reuse_mode=1 时，高并发场景下发生源端口与之前链接重复的情况，不会复用链接，而是会重新进行调度。根据 ip_vs_in()的处理逻辑，当开启了 net.ipv4.vs.conntrack 时，这种情况会先 DROP 掉第一个 SYN 包，导致 SYN 的重传，有 1 秒延迟。导致性能下降。

社区当前行为及在 cce 集群中影响

社区 1.17 及以下的版本默认将 net.ipv4.vs.conn_reuse_mode 设置为 0，从 1.19 版本开始会根据内核的版本选择是否开启 net.ipv4.vs.conn_reuse_mode，内核版本大于 4.1 的情况下会将 net.ipv4.vs.conn_reuse_mode 设置为 1，其他情况会保持系统原有的默认值。

对 CCE 节点的影响

1. CCE 1.17 及以下版本的集群，使用 IPVS 作为服务转发模式的情况下，net.ipv4.vs.conn_reuse_mode=0 存在**•问题 1**，即端口复用下的 RS 无法移除问题。
2. CCE 1.19 版本集群。
   - 当节点的 OS 版本为：EulerOS 2.5 和 CentOS 7.6 时，net.ipv4.vs.conn_reuse_mode=0，存在**•问题 1**，即端口复用下的 RS 无法移除问题。
   - 当节点的 OS 版本为：EulerOS 2.9 和 Ubuntu 18.04 时，net.ipv4.vs.conn_reuse_mode=1，存在**•问题 2**，即高并发场景存在 1 秒延时。

建议

如果上述问题会对您的业务产生影响建议您采取以下措施：

1. 使用服务转发模式为 iptables 的集群。
2. 如果您评估该问题影响有限，1.19 版本的集群可根据问题影响选择您需要的操作系统版本。

修复计划

CCE 已经在 EulerOS 2.9 上修复相关的问题，您可以选择重置节点或者重新创建节点解决该问题。

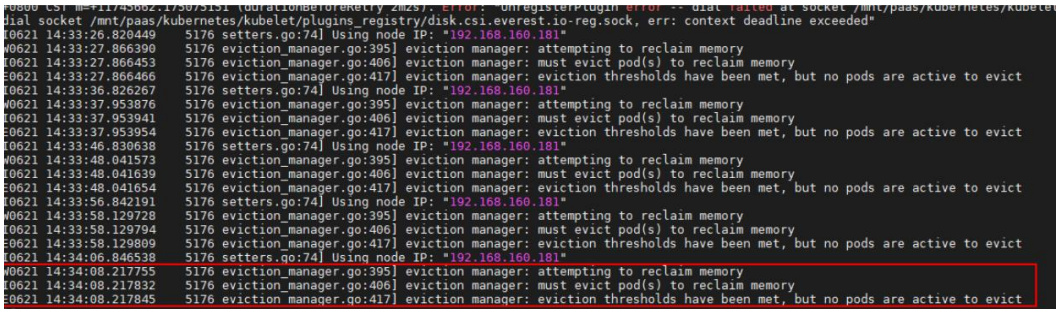
K8s 社区 issue：https://github.com/kubernetes/kubernetes/issues/81775

4.4.3 cgroup 统计资源异常导致 kubelet 驱逐 Pod

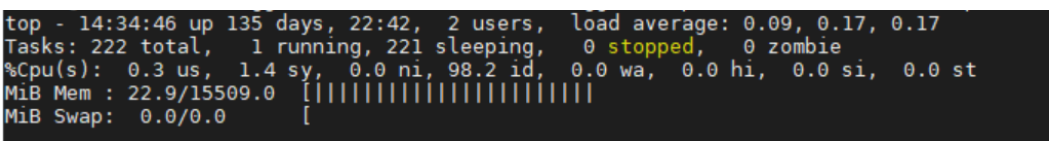
故障现象

ARM 架构节点上，cgroup 统计资源异常导致 kubelet 驱逐 Pod，节点无法正常使用。

kubelet 一直在驱逐 pod，把容器全杀掉之后还是认为内存不足。



此时实际资源使用正常。



查看/sys/fs/cgroup/memory 目录下 cgroup 的 usage_in_bytes 统计值有问题，与实际不符。

```
# cd /sys/fs/cgroup/memory
# cat memory.usage_in_bytes
17618837504
```

问题根因

ARM 架构节点的 EulerOS 2.8 和 EulerOS 2.9 操作系统内核存在 Bug，会触发 kubelet 驱逐 Pod 导致业务不可用。

说明

该问题在以下版本中已被修复：

- EulerOS 2.8：内核版本 kernel-4.19.36-vhulk1907.1.0.h1088.eulerosv2r8.aarch64
- EulerOS 2.9：内核版本 kernel-4.19.90-vhulk2103.1.0.h539.eulerosv2r9.aarch64

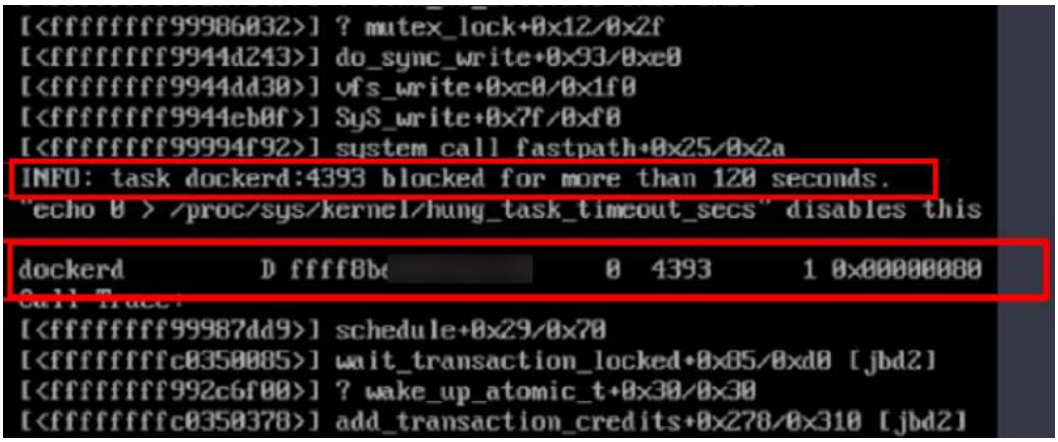
解决方法

- 若您的集群版本为 1.19.16-r0、1.21.7-r0、1.23.5-r0、1.25.1-r0 及以上，请将节点重置为最新版本的操作系统即可修复该问题。
- 若您的集群版本不满足要求，请将集群升级到上述指定的版本后，再将节点重置为最新版本的操作系统。

4.4.4 低版本内核的 CentOS 节点出现容器 OOM 时，偶现 ext4 文件系统卡死问题

故障现象

CentOS 7.6 节点内核低于 3.10.0-1160.66.1.el7.x86_64 的场景下，节点上容器出现 OOM 后，可能遇到节点上所有容器无法访问，docker、jdb 等相关进程处于 D 状态，节点重启后恢复。



问题根因

业务容器内存使用超过容器的内存限制量时，触发 cgroup OOM，被系统内核杀死。容器 cgroup OOM 在 CentOS 7 会偶现触发 ext4 文件系统卡死，ext4/jbd2 会因为死锁而永远挂起。在文件系统上执行 I/O 的所有任务都将受到影响。

详情请参见 CentOS 开源社区：ext4/jbd2 hang forever due to deadlock when OOM occurs

解决方法

- 临时解决方案：该问题触发后可以通过重启节点临时恢复。
- 长久解决方案：
  - 若您的集群版本为 1.19.16-r0、1.21.7-r0、1.23.5-r0、1.25.1-r0 及以上，请将节点重置为最新版本的操作系统即可修复该问题。
  - 若您的集群版本不满足要求，请将集群升级到上述指定的版本后，再将节点重置为最新版本的操作系统。

5 节点池

5.1.1 节点池一直在扩容中，但“操作记录”里却没有创建节点的记录？

问题现象

节点池的状态一直处于“扩容中”，但是“操作记录”里面没有看到有对应创建节点的记录。

原因排查：

检查如下问题并修复：

- 查看此规格节点是否售罄
- 租户的 ECS 或内存配额是否不足。
- 租户的 ECS 容量是否存在校验不过的问题，如果一次创建节点太多，可能会有容量校验不过的情况发生。
- 租户是否欠费，用户 token 的 role 是否已经受限。

解决方案：

- 若租户已经欠费，请尽快续费。
- 若 ECS 节点资源售罄，使用其他规格节点替代。
- 若 ECS 或内存配额不足，请扩大配额。
- 若 ECS 容量校验不通过，请重新校验。

6 工作负载

6.1 工作负载异常

6.1.1 工作负载状态异常定位方法

工作负载状态异常时，建议先查看 Pod 的事件以便于确定导致异常的初步原因，再参照下表中的内容针对性解决问题。

表6-1 排查思路列表

事件信息	实例状态	处理措施
实例调度失败	Pending	请参考工作负载异常：实例调度失败
拉取镜像失败	ImagePullBackOff	请参考工作负载异常：实例拉取镜像失败
启动容器失败	CreateContainerError CrashLoopBackOff	请参考工作负载异常：启动容器失败
实例状态为“Evicted”，pod 不断被驱逐	Evicted	请参考工作负载异常：实例驱逐异常（Evicted）
实例挂卷失败	Pending	请参考工作负载异常：存储卷无法挂载或挂载超时
实例状态一直为“创建中”	Creating	请参考工作负载异常：一直处于创建中
实例状态一直为“结束中”	Terminating	请参考工作负载异常：结束中，解决 Terminating 状态的 Pod 删不掉的问题
实例状态为“已停止”	Stopped	请参考工作负载异常：已停止

Pod 事件查看方法

Pod 的事件可以使用 kubectl describe pod *podname* 命令查看，或在 CCE 控制台，工作负载详情页面中查看。

```
$ kubectl describe pod prepare-58bd7bdf9-fthrp
...
Events:
  Type     Reason            Age   From               Message
  ----     ------            ----  ----               -------
  Warning  FailedScheduling  49s   default-scheduler  0/2 nodes are available: 2
Insufficient cpu.
  Warning  FailedScheduling  49s   default-scheduler  0/2 nodes are available: 2
Insufficient cpu.
```

图6-1 查看 Pod 事件

6.1.2 工作负载异常：实例调度失败

问题定位

当 Pod 状态为“Pending”，事件中出现“实例调度失败”的信息时，可根据具体事件信息确定具体问题原因。事件查看方法请参见工作负载状态异常定位方法。

排查思路

根据具体事件信息确定具体问题原因，如下表所示。

表6-2 实例调度失败

事件信息	问题原因与解决方案
no nodes available to schedule pods.	集群中没有可用的节点。 排查项一：集群内是否无可用节点
0/2 nodes are available: 2 Insufficient cpu. 0/2 nodes are available: 2 Insufficient memory.	节点资源（CPU、内存）不足。 排查项二：节点资源（CPU、内存等）

事件信息	问题原因与解决方案
	是否充足
0/2 nodes are available: 1 node(s) didn't match node selector, 1 node(s) didn't match pod affinity rules, 1 node(s) didn't match pod affinity/anti-affinity.	节点与 Pod 亲和性配置互斥，没有满足 Pod 要求的节点。 排查项三：检查工作负载的亲和性配置
0/2 nodes are available: 2 node(s) had volume node affinity conflict.	Pod 挂载云硬盘存储卷与节点不在同一个可用区。 排查项四：挂载的存储卷与节点是否处于同一可用区
0/1 nodes are available: 1 node(s) had taints that the pod didn't tolerate.	节点存在污点 Tanits，而 Pod 不能容忍这些污点，所以不可调度。 排查项五：检查 Pod 污点容忍情况
0/7 nodes are available: 7 Insufficient ephemeral-storage.	节点临时存储不足。 排查项六：检查临时卷使用量
0/1 nodes are available: 1 everest driver not found at node	节点上 everest-csi-driver 不在 running 状态。 排查项七：检查 everest 插件是否工作正常。
Failed to create pod sandbox: ... Create more free space in thin pool or use dm.min_free_space option to change behavior	节点 thinpool 空间不足。 排查项八：检查节点 thinpool 空间是否充足。

排查项一：集群内是否无可用节点

登录 CCE 控制台，检查节点状态是否为可用。或使用如下命令查看节点状态是否为 Ready。

```
$ kubectl get node
NAME          STATUS   ROLES    AGE   VERSION
192.168.0.37   Ready    <none>   21d   v1.19.10-r1.0.0-source-121-gb9675686c54267
192.168.0.71   Ready    <none>   21d   v1.19.10-r1.0.0-source-121-gb9675686c54267
```

如果状态都为不可用（Not Ready），则说明集群中无可用节点。

解决方案：

- 新增节点，若工作负载未设置亲和策略，pod 将自动迁移至新增的可用节点，确保业务正常。
- 排查不可用节点问题并修复，排查修复方法请参见集群可用，但节点状态为“不可用”？。
- 重置不可用的节点。

排查项二：节点资源（CPU、内存等）是否充足

0/2 nodes are available: 2 Insufficient cpu. CPU 不足。

0/2 nodes are available: 2 Insufficient memory. 内存不足。

当“实例资源的申请量”超过了“实例所在节点的可分配资源总量”时，节点无法满足实例所需资源要求导致调度失败。

如果节点可分配资源小于 Pod 的申请量，则节点无法满足实例所需资源要求导致调度失败。

解决方案：

资源不足的情况主要解决办法是扩容，建议在集群中增加节点数量。

排查项三：检查工作负载的亲和性配置

当亲和性配置出现如下互斥情况时，也会导致实例调度失败：

例如：

workload1、workload2 设置了工作负载间的反亲和，如 workload1 部署在 Node1，workload2 部署在 Node2。

workload3 部署上线时，既希望与 workload2 亲和，又希望可以部署在不同节点如 Node1 上，这就造成了工作负载亲和与节点亲和间的互斥，导致最终工作负载部署失败。

0/2 nodes are available: 1 node(s) didn't match **node selector**, 1 node(s) didn't match **pod affinity rules**, 1 node(s) didn't match **pod affinity/anti-affinity**.

- **node selector** 表示节点亲和不满足。
- **pod affinity rules** 表示 Pod 亲和没不满足。
- **pod affinity/anti-affinity** 表示 Pod 亲和/反亲和没不满足。

解决方案：

- 在设置“工作负载间的亲和性”和“工作负载和节点的亲和性”时，需确保不要出现互斥情况，否则工作负载会部署失败。
- 若工作负载配置了节点亲和性，需确保亲和的节点标签中 supportContainer 设置为 true，否则会导致 pod 无法调动到节点上，查看事件提示如下错误信息：

  ```
  No nodes are available that match all of the following predicates: MatchNode Selector, NodeNotSupportsContainer
  ```

  节点标签为 false 时将会调度失败

排查项四：挂载的存储卷与节点是否处于同一可用区

0/2 nodes are available: 2 node(s) had volume node affinity conflict. 存储卷与节点之间存在亲和性冲突，导致无法调度。

这是因为云硬盘不能跨可用区挂载到节点。例如云硬盘存储卷在可用区 1，节点在可用区 2，则会导致无法调度。

CCE 中创建云硬盘存储卷，默认带有亲和性设置，如下所示。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: pvc-c29bfac7-efa3-40e6-b8d6-229d8a5372ac
spec:
  ...
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: failure-domain.beta.kubernetes.io/zone
              operator: In
              values:
                - cn-gz1a
```

解决方案：

重新创建存储卷，可用区选择与节点同一分区，或重新创建工作负载，存储卷选择自动分配。

排查项五：检查 Pod 污点容忍情况

0/1 nodes are available: 1 node(s) had taints that the pod didn't tolerate. 是因为节点打上了污点，不允许 Pod 调度到节点上。

查看节点的上污点的情况。如下则说明节点上存在污点。

```
$ kubectl describe node 192.168.0.37
Name:            192.168.0.37
...
Taints:          key1=value1:NoSchedule
...
```

在某些情况下，系统会自动给节点添加一个污点。当前内置的污点包括：

- node.kubernetes.io/not-ready：节点未准备好。
- node.kubernetes.io/unreachable：节点控制器访问不到节点。
- node.kubernetes.io/memory-pressure：节点存在内存压力。
- node.kubernetes.io/disk-pressure：节点存在磁盘压力，此情况表明节点上用于存储镜像的磁盘空间已满，需要清理镜像，或扩容磁盘。
- node.kubernetes.io/pid-pressure：节点的 PID 压力，此情况下您可通过修改节点进程 ID 数量上限 kernel.pid_max 进行解决。
- node.kubernetes.io/network-unavailable：节点网络不可用。

- node.kubernetes.io/unschedulable：节点不可调度。
- node.cloudprovider.kubernetes.io/uninitialized：如果 kubelet 启动时指定了一个“外部”云平台驱动， 它将给当前节点添加一个污点将其标志为不可用。在 cloud-controller-manager 初始化这个节点后，kubelet 将删除这个污点。

解决方案：

要想把 Pod 调度到这个节点上，有两种方法：

- 若该污点为用户自行添加，可考虑删除节点上的污点。若该污点为系统自动添加，解决相应问题后污点会自动删除。
- Pod 的定义中容忍这个污点，如下所示。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:alpine
  tolerations:
  - key: "key1"
    operator: "Equal"
    value: "value1"
    effect: "NoSchedule"
```

排查项六：检查临时卷使用量

0/7 nodes are available: 7 Insufficient ephemeral-storage. 节点临时存储不足。

检查 Pod 是否限制了临时卷的大小，如下所示，当应用程序需要使用的量超过节点已有容量时会导致无法调度，修改临时卷限制或扩容节点磁盘可解决此问题。

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests:
        ephemeral-storage: "2Gi"
      limits:
        ephemeral-storage: "4Gi"
    volumeMounts:
    - name: ephemeral
      mountPath: "/tmp"
  volumes:
    - name: ephemeral
      emptyDir: {}
```

排查项七：检查 everest 插件是否工作正常。

0/1 nodes are available: 1 everest driver not found at node。集群 everest 插件的 everest-csi-driver在节点上未正常启动。

检查 kube-system 命名空间下名为 everest-csi-driver 的守护进程，查看对应 Pod 是否正常启动，若未正常启动，删除该 Pod，守护进程会重新拉起该 Pod。

排查项八：检查节点 thinpool 空间是否充足。

节点在创建时会绑定一个 100G 的 docker 专用数据盘。若数据盘空间不足，将导致实例无法正常创建。

方案一

您可以执行以下命令清理未使用的 Docker 镜像：

```
docker system prune -a
```

说明

该命令会把暂时没有用到的 Docker 镜像都删除，执行命令前请确认。

方案二

或者您也可以选择扩容磁盘，具体步骤如下：

步骤 1 在 EVS 界面扩容数据盘。

步骤 2 登录 CCE 控制台，进入集群，在左侧选择“节点管理”，单击节点后的“同步云服务器”。

步骤 3 登录目标节点。

步骤 4 使用 **lsblk** 命令查看节点块设备信息。

这里存在两种情况，根据容器存储 Rootfs 而不同。

- Overlayfs，没有单独划分 thinpool，在 dockersys 空间下统一存储镜像相关数据。

```
# lsblk
NAME              MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                 8:0    0   50G  0 disk
└─sda1              8:1    0   50G  0 part /
sdb                 8:16   0  200G  0 disk
├─vgpaas-dockersys  253:0    0   90G  0 lvm  /var/lib/docker            # docker 使用的空间
└─vgpaas-kubernetes 253:1    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet  # kubernetes 使用的空间
```

在节点上执行如下命令， 将新增的磁盘容量加到 dockersys 盘上。

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/dockersys
resize2fs /dev/vgpaas/dockersys
```

- Devicemapper，单独划分了 thinpool 存储镜像相关数据。

```
# lsblk
NAME                            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
```

```
sda                              8:0    0   50G  0 disk
└─sda1                           8:1    0   50G  0 part /
sdb                              8:16   0  200G  0 disk
├─vgpaas-dockersys               253:0    0   18G  0 lvm  /var/lib/docker
├─vgpaas-thinpool_tmeta          253:1    0    3G  0 lvm
│ └─vgpaas-thinpool              253:3    0   67G  0 lvm                #
thinpool 空间
│   ...
├─vgpaas-thinpool_tdata          253:2    0   67G  0 lvm
│ └─vgpaas-thinpool              253:3    0   67G  0 lvm
│   ...
└─vgpaas-kubernetes              253:4    0   10G  0 lvm
/mnt/paas/kubernetes/kubelet
```

- 在节点上执行如下命令， 将新增的磁盘容量加到 thinpool 盘上。

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/thinpool
```

- 在节点上执行如下命令， 将新增的磁盘容量加到 dockersys 盘上。

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/dockersys
resize2fs /dev/vgpaas/dockersys
```

----结束

6.1.3 工作负载异常：实例拉取镜像失败

问题定位

当 Pod 状态为“ImagePullBackOff”，说明实例拉取镜像失败。

排查思路

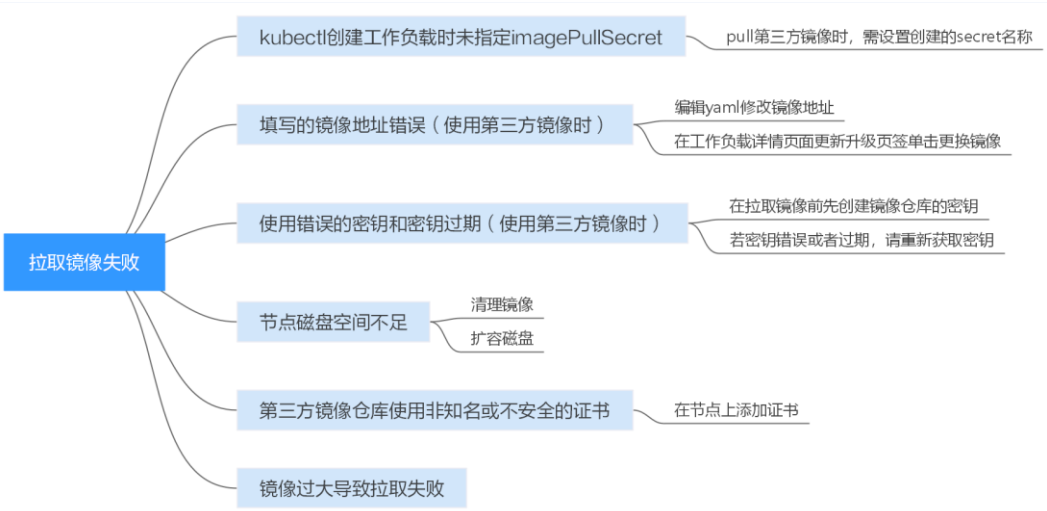
根据具体事件信息确定具体问题原因，如下表所示。

表6-3 实例拉取镜像失败

事件信息	问题原因与解决方案
Failed to pull image "xxx": rpc error: code = Unknown desc = Error response from daemon: Get xxx: denied: You may not login yet	没有登录镜像仓库，无法拉取镜像。 排查项一：kubectl 创建工作负载时未指定 imagePullSecret
Failed to pull image "nginx:v1.1": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: dial tcp: lookup registry-1.docker.io: no such host	镜像地址配置有误找不到镜像导致失败。 排查项二：填写的镜像地址错误（使用第三方镜像时） 排查项三：使用错误的密钥（使用第三方镜像时）
Failed to pull image "docker.io/bitnami/nginx:1.22.0-debian-11-	无法连接镜像仓库，网络不通。

事件信息	问题原因与解决方案
r3": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)	排查项七：无法连接镜像仓库
Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod "nginx-6dc48bf8b6-l8xrw": Error response from daemon: mkdir xxxxx: no space left on device	磁盘空间不足。 排查项四：节点磁盘空间不足
Failed to pull image "xxx": rpc error: code = Unknown desc = error pulling image configuration: xxx x509: certificate signed by unknown authority	从第三方仓库下载镜像时，第三方仓库使用了非知名或者不安全的证书. 排查项五： 远程镜像仓库使用非知名或不安全的证书
Failed to pull image "XXX": rpc error: code = Unknown desc = context canceled	镜像体积过大。 排查项六： 镜像过大导致失败
Failed to pull image "docker.io/bitnami/nginx:1.22.0-debian-11-r3": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)	排查项七：无法连接镜像仓库
ERROR: toomanyrequests: Too Many Requests. 或 you have reached your pull rate limit, you may increase the limit by authenticating an upgrading	由于拉取镜像次数达到上限而被限速。 排查项八：拉取公共镜像达上限

图6-2 排查思路



排查项一：kubectl 创建工作负载时未指定 imagePullSecret

当工作负载状态异常并显示“实例拉取镜像失败”的 K8s 事件时，请排查 yaml 文件中是否存在 **imagePullSecrets** 字段。

排查事项：

- 当 Pull SWR 容器镜像仓库的镜像时，name 参数值需固定为 default-secret。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        imagePullPolicy: Always
        name: nginx
      imagePullSecrets:
      - name: default-secret
```

- Pull 第三方镜像仓库的镜像时，需设置为创建的 secret 名称。

  kubectl 创建工作负载拉取第三方镜像时，需指定的 imagePullSecret 字段，name 表示 pull 镜像时的 secret 名称。

排查项二：填写的镜像地址错误（使用第三方镜像时）

CCE 支持拉取第三方镜像仓库中的镜像来创建工作负载。

在填写第三方镜像的地址时，请参照要求的格式来填写。镜像地址格式为：ip:port/path/name:version 或 name:version，若没标注版本号则默认版本号为 latest。

- 若是私有仓库，请填写 ip:port/path/name:version。
- 若是 docker 开源仓库，请填写 name:version，例如 nginx:latest。

图6-3 第三方镜像

镜像地址配置有误找不到镜像导致失败，Kubernetes Event 中提示如下信息：

```
Failed to pull image "nginx:v1.1": rpc error: code = Unknown desc = Error response
from daemon: Get https://registry-1.docker.io/v2/: dial tcp: lookup registry-
1.docker.io: no such host
```

解决方案：

可编辑 yaml 修改镜像地址，也可在工作负载详情页面更新升级页签单击更换镜像。

排查项三：使用错误的密钥（使用第三方镜像时）

通常第三方镜像仓库都必须经过认证（帐号密码）才可以访问，而 CCE 中容器拉取镜像是使用密钥认证方式，这就要求在拉取镜像前必须先创建镜像仓库的密钥。

解决方案：

若您的密钥错误将会导致镜像拉取失败，请重新获取密钥。

排查项四：节点磁盘空间不足

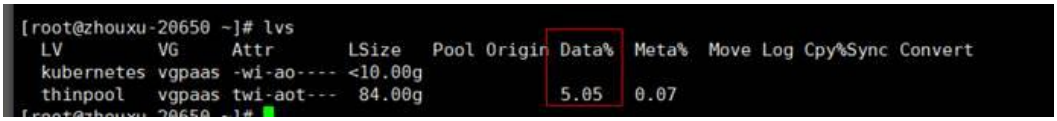
新建节点会给节点绑定一个 100G 的 docker 专用数据盘。若数据盘空间不足，会导致重新拉取镜像失败。

图6-4 数据盘

当 k8s 事件中包含以下信息，表明节点上用于存储镜像的磁盘空间已满，需要清理镜像，或扩容磁盘。

```
Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a
sandbox for pod "nginx-6dc48bf8b6-l8xrw": Error response from daemon: mkdir xxxxx:
no space left on device
```

确认节点上存储镜像的磁盘空间的命令为：lvs



清理镜像的命令为：

```
docker rmi -f {镜像 ID}
```

扩容磁盘的操作步骤如下：

步骤 1 在 EVS 界面扩容数据盘。

步骤 2 登录 CCE 控制台，进入集群，在左侧选择“节点管理”，单击节点后的“同步云服务器”。

步骤 3 登录目标节点。

步骤 4 使用 **lsblk** 命令查看节点块设备信息。

这里存在两种情况，根据容器存储 Rootfs 而不同。

- Overlayfs，没有单独划分 thinpool，在 dockersys 空间下统一存储镜像相关数据。

```
# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                   8:0    0   50G  0 disk
└─sda1                8:1    0   50G  0 part /
sdb                   8:16   0  200G  0 disk
├─vgpaas-dockersys  253:0    0   90G  0 lvm  /var/lib/docker             # docker
使用的空间
└─vgpaas-kubernetes 253:1    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet  #
kubernetes 使用的空间
```

在节点上执行如下命令， 将新增的磁盘容量加到 dockersys 盘上。

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/dockersys
resize2fs /dev/vgpaas/dockersys
```

- Devicemapper，单独划分了 thinpool 存储镜像相关数据。

```
# lsblk
NAME                            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                               8:0    0   50G  0 disk
└─sda1                            8:1    0   50G  0 part /
sdb                               8:16   0  200G  0 disk
├─vgpaas-dockersys                253:0    0   18G  0 lvm  /var/lib/docker
├─vgpaas-thinpool_tmeta           253:1    0    3G  0 lvm
│ └─vgpaas-thinpool               253:3    0   67G  0 lvm                #
thinpool 空间
│   ...
├─vgpaas-thinpool_tdata           253:2    0   67G  0 lvm
│ └─vgpaas-thinpool               253:3    0   67G  0 lvm
│   ...
└─vgpaas-kubernetes               253:4    0   10G  0 lvm
/mnt/paas/kubernetes/kubelet
```

- 在节点上执行如下命令， 将新增的磁盘容量加到 thinpool 盘上。

  ```
  pvresize /dev/sdb
  lvextend -l+100%FREE -n vgpaas/thinpool
  ```

- 在节点上执行如下命令， 将新增的磁盘容量加到 dockersys 盘上。

  ```
  pvresize /dev/sdb
  lvextend -l+100%FREE -n vgpaas/dockersys
  resize2fs /dev/vgpaas/dockersys
  ```

----结束

排查项五： 远程镜像仓库使用非知名或不安全的证书

从第三方仓库下载镜像时，若第三方仓库使用了非知名或者不安全的证书，节点上会拉取镜像失败，Pod 事件列表中有“实例拉取镜像失败”事件，报错原因为"x509: certificate signed by unknown authority"。

操作步骤：

步骤 1 确认报错 unknown authority 的第三方镜像服务器地址和端口。

从"实例拉取镜像失败"事件信息中能够直接看到报错的第三方镜像服务器地址和端口，如上图中错误信息为：

```
Failed to pull image "bitnami/redis-cluster:latest": rpc error: code = Unknown desc
= error pulling image configuration: Get
https://production.cloudflare.docker.com/registry-
v2/docker/registry/v2/blobs/sha256/e8/e83853f03a2e792614e7c1e6de75d63e2d6d633b4e7c3
9b9d700792ee50f7b56/data?verify=1636972064-AQbl5RActnudDZV%2F3EShZwnqOe8%3D: x509:
certificate signed by unknown authority
```

对应的第三方镜像服务器地址为 *production.cloudflare.docker.com*，端口为 https 默认端口 *443*。

步骤 2 在需要下载第三方镜像的节点上加载第三方镜像服务器的根证书。

CentOS 节点执行如下命令，{server_url}:{server_port}需替换成步骤 1 中地址和端口，如 production.cloudflare.docker.com:443。

若节点的容器引擎为 containerd，最后一步“systemctl restart docker”命令替换为 "systemctl restart containerd"。

```
openssl s_client -showcerts -connect {server_url}:{server_port} < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /etc/pki/ca-trust/source/anchors/tmp_ca.crt
update-ca-trust
systemctl restart docker
```

ubuntu 节点执行如下命令。

```
openssl s_client -showcerts -connect {server_url}:{server_port} < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' > /usr/local/share/ca-certificates/tmp_ca.crt
update-ca-trust
systemctl restart docker
```

----结束

排查项六： 镜像过大导致失败

Pod 事件列表中有“实例拉取镜像失败”事件，报错原因如下。这可能是镜像较大导致的情况。

```
Failed to pull image "XXX": rpc error: code = Unknown desc = context canceled
```

登录节点使用 docker pull 命令手动下拉镜像，镜像下拉成功。

问题根因：

kubernetes 默认的 image-pull-progress-deadline 是 1 分钟，如果 1 分钟内镜像下载没有任何进度更新，下载动作就会取消。在节点性能较差或镜像较大时，可能出现镜像无法成功下载，负载启动失败的现象。

解决方案：

- (推荐)方法一：登录节点使用 docker pull 命令手动下拉镜像，确认负载的镜像拉取策略 imagePullPolicy 为 IfNotPresent（默认策略配置）。此时创建负载会使用已拉取到本地的镜像。
- 方法二：修改 kubelet 配置参数。

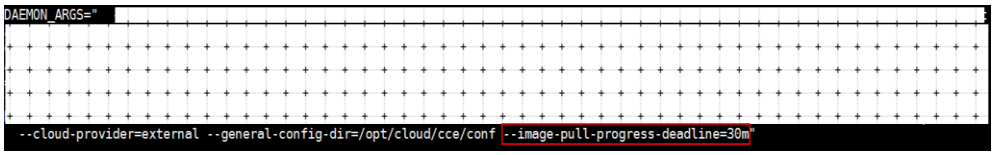
  1.15 及以上集群使用如下命令:

  ```
  vi /opt/cloud/cce/kubernetes/kubelet/kubelet
  ```

  1.15 以下集群使用如下命令：

  ```
  vi /var/paas/kubernetes/kubelet/kubelet
  ```

  在 DAEMON_ARGS 参数末尾追加配置 --image-pull-progress-deadline=30m ，30m 为 30 分钟，可根据需求修改为合适时间。追加配置和前项配置之间由空格分开。

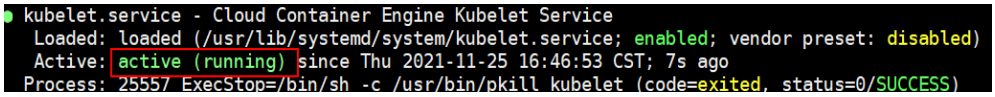
  

  重启 kubelet:

  ```
  systemctl restart kubelet
  ```

等待片刻，确定 kubelet 状态为 running

```
systemctl status kubelet
```



负载正常启动，镜像下拉成功。

排查项七：无法连接镜像仓库

问题现象

创建工作负载时报如下错误。

```
Failed to pull image "docker.io/bitnami/nginx:1.22.0-debian-11-r3": rpc error: code = Unknown desc = Error response from daemon: Get https://registry-1.docker.io/v2/: net/http: request canceled while waiting for connection (Client.Timeout exceeded while awaiting headers)
```

问题原因

无法连接镜像仓库，网络不通。SWR 仅支持直接拉 Docker 官方的镜像，其他仓库的镜像需要连接。

解决方案：

- 给需要下载镜像的节点绑定公网 IP。
- 先将镜像上传到 SWR，然后从 SWR 拉取镜像。

排查项八：拉取公共镜像达上限

问题现象

创建工作负载时报如下错误。

```
ERROR: toomanyrequests: Too Many Requests.
```

或

```
you have reached your pull rate limit, you may increase the limit by authenticating an upgrading: https://www.docker.com/increase-rate-limits.
```

问题原因

DockerHub 对用户拉取容器镜像请求设定了上限，详情请参见 Understanding Docker Hub Rate Limiting。

解决方案：

将常用的镜像上传到 SWR，然后从 SWR 拉取镜像。

6.1.4 工作负载异常：启动容器失败

问题定位

工作负载详情中，若事件中提示“启动容器失败”，请按照如下方式来初步排查原因：

步骤 1 登录异常工作负载所在的节点。

步骤 2 查看工作负载实例非正常退出的容器 ID。

```
docker ps -a | grep $podName
```

步骤 3 查看退出容器的错误日志。

```
docker logs $containerID
```

根据日志提示修复工作负载本身的问题。

步骤 4 查看操作系统的错误日志。

```
cat /var/log/messages | grep $containerID  | grep oom
```

根据日志判断是否触发了系统 OOM。

----结束

排查思路

根据具体事件信息确定具体问题原因，如下表所示。

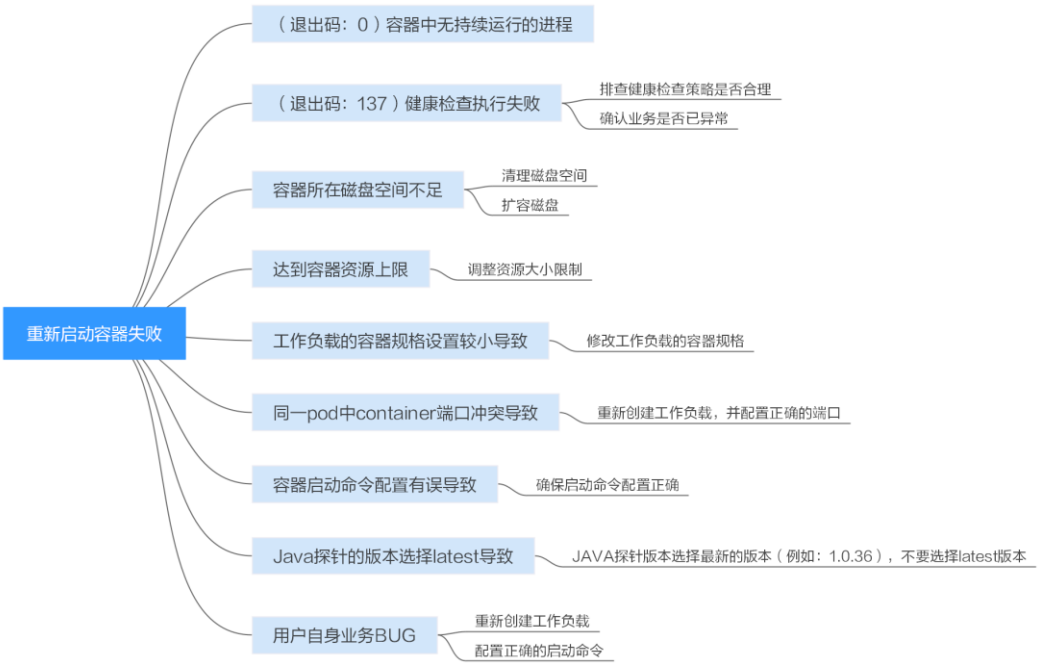
表6-4 容器启动失败

日志或事件信息	问题原因与解决方案
日志中存在 exit(0)	容器中无进程。 请调试容器是否能正常运行。 排查项一：（退出码：0）容器中无持续运行的进程
事件信息：Liveness probe failed: Get http… 日志中存在 exit(137)	健康检查执行失败。 排查项二：（退出码：137）健康检查执行失败
事件信息： Thin Pool has 15991 free data blocks which is less than minimum required 16383 free data blocks. Create more free space in thin pool or use dm.min_free_space option to change behavior	磁盘空间不足，需要清理磁盘空间。 排查项三：容器所在磁盘空间不足
日志中存在 OOM 字眼	内存不足。 排查项四：达到容器资源上限 排查项五：工作负载的容器规格设置较小导致
Address already in use	Pod 中容器端口冲突 排查项六：同一 pod 中 container 端口冲突导致

出上述可能原因外，还可能存在如下原因，请根据顺序排查。

- 排查项七：容器启动命令配置有误导致
- 排查项九：用户自身业务 BUG
- 在 ARM 架构的节点上创建工作负载时未使用正确的镜像版本，使用正确的镜像版本即可解决该问题。

图6-5 排查思路



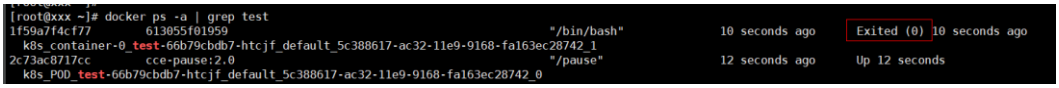
排查项一：（退出码：0）容器中无持续运行的进程

步骤 1 登录异常工作负载所在的节点。

步骤 2 查看容器状态。

```
docker ps -a | grep $podName
```

如下图所示：



当容器中无持续运行的进程时，会出现 exit(0)的状态码，此时说明容器中无进程。

----结束

排查项二：（退出码：137）健康检查执行失败

工作负载配置的健康检查会定时检查业务，异常情况下 pod 会报实例不健康的事件且 pod 一直重启失败。

工作负载若配置 liveness 型（工作负载存活探针）健康检查，当健康检查失败次数超过阈值时，会重启实例中的容器。在工作负载详情页面查看事件，若 K8s 事件中出现“Liveness probe failed: Get http…”时，表示健康检查失败。

解决方案：

请核查“工作负载详情>更新升级>高级配置>健康检查”中的信息，排查健康检查策略是否合理或业务是否已异常。

排查项三：容器所在磁盘空间不足

如下磁盘为创建节点时选择的 docker 专用盘分出来的 thinpool 盘，以 root 用户执行 **lvs** 命令可以查看当前磁盘的使用量。

```
Thin Pool has 15991 free data blocks which is less than minimum required 16383 free data blocks. Create more free space in thin pool or use dm.min_free_space option to change behavior
```

解决方案：

方案一

您可以执行以下命令清理未使用的垃圾镜像：

```
docker system prune -a
```

说明

该命令会把暂时没有用到的 Docker 镜像都删除，执行命令前请确认。

方案二

或者您也可以选择扩容磁盘，具体步骤如下：

步骤 1 在 EVS 界面扩容数据盘。

步骤 2 登录 CCE 控制台，进入集群，在左侧选择“节点管理”，单击节点后的“同步云服务器”。

步骤 3 登录目标节点。

步骤 4 使用 **lsblk** 命令查看节点块设备信息。

这里存在两种情况，根据容器存储 Rootfs 而不同。

- Overlayfs，没有单独划分 thinpool，在 dockersys 空间下统一存储镜像相关数据。

```
# lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
```

```
sda                 8:0    0   50G  0 disk
└─sda1              8:1    0   50G  0 part /
sdb                 8:16   0  200G  0 disk
├─vgpaas-dockersys  253:0    0   90G  0 lvm  /var/lib/docker            # docker 使用的空间
└─vgpaas-kubernetes 253:1    0   10G  0 lvm  /mnt/paas/kubernetes/kubelet  # kubernetes 使用的空间
```

在节点上执行如下命令， 将新增的磁盘容量加到 dockersys 盘上。

```
pvresize /dev/sdb
lvextend -l+100%FREE -n vgpaas/dockersys
resize2fs /dev/vgpaas/dockersys
```

- Devicemapper，单独划分了 thinpool 存储镜像相关数据。

```
# lsblk
NAME                              MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda                                 8:0    0   50G  0 disk
└─sda1                              8:1    0   50G  0 part /
sdb                                 8:16   0  200G  0 disk
├─vgpaas-dockersys                  253:0    0   18G  0 lvm  /var/lib/docker
├─vgpaas-thinpool_tmeta             253:1    0    3G  0 lvm
│ └─vgpaas-thinpool                 253:3    0   67G  0 lvm                # thinpool 空间
│   ...
├─vgpaas-thinpool_tdata             253:2    0   67G  0 lvm
│ └─vgpaas-thinpool                 253:3    0   67G  0 lvm
│   ...
└─vgpaas-kubernetes                 253:4    0   10G  0 lvm
/mnt/paas/kubernetes/kubelet
```

  - 在节点上执行如下命令， 将新增的磁盘容量加到 thinpool 盘上。

    ```
    pvresize /dev/sdb
    lvextend -l+100%FREE -n vgpaas/thinpool
    ```

  - 在节点上执行如下命令， 将新增的磁盘容量加到 dockersys 盘上。

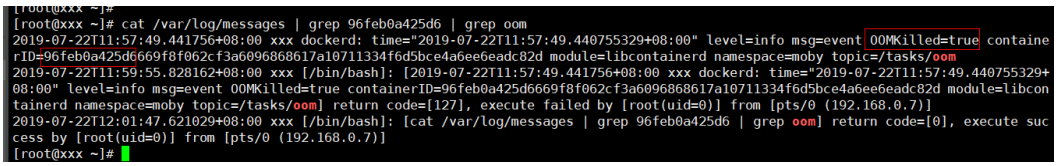
    ```
    pvresize /dev/sdb
    lvextend -l+100%FREE -n vgpaas/dockersys
    resize2fs /dev/vgpaas/dockersys
    ```

----结束

排查项四：达到容器资源上限

事件详情中有 OOM 字样。并且，在日志中也会有记录：

```
cat /var/log/messages | grep 96feb0a425d6 | grep oom
```



创建工作负载时，设置的限制资源若小于实际所需资源，会触发系统 OOM，并导致容器异常退出。

排查项五：工作负载的容器规格设置较小导致

工作负载的容器规格设置较小导致，若创建工作负载时，设置的限制资源少于实际所需资源，会导致启动容器失败。

排查项六：同一 pod 中 container 端口冲突导致

步骤 1 登录异常工作负载所在的节点。

步骤 2 查看工作负载实例非正常退出的容器 ID。

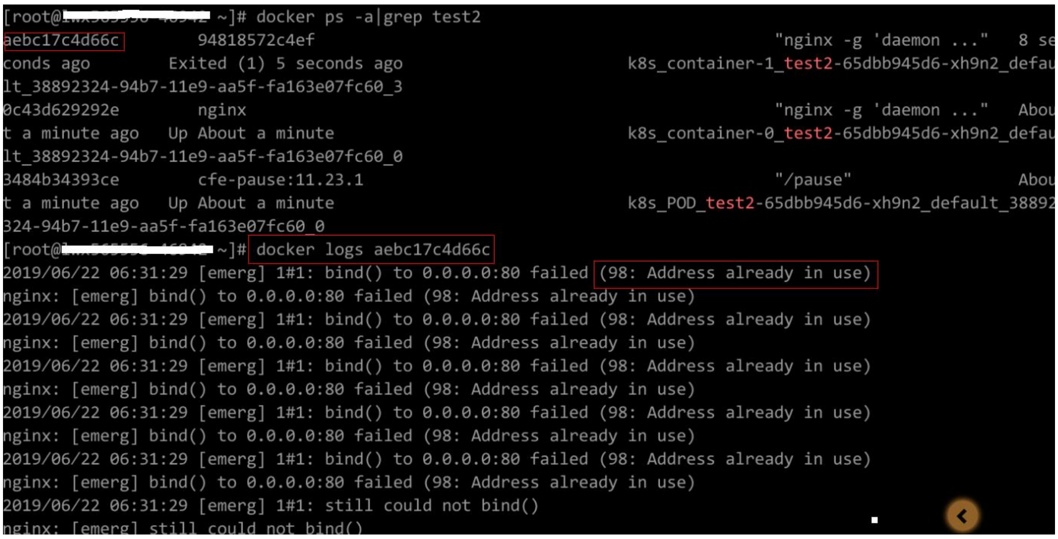
docker ps -a | grep *$podName*

步骤 3 查看退出容器的错误日志。

docker logs *$containerID*

根据日志提示修复工作负载本身的问题。如下图所示，即同一 Pod 中的 container 端口冲突导致容器启动失败。

图6-6 container 冲突导致容器启动失败



----结束

解决方案：

重新创建工作负载，并配置正确的端口，确保端口不冲突。

排查项七：容器启动命令配置有误导致

错误信息如下图所示：

```
[root@[redacted] ~]# docker ps -a|grep test1
2ae258d570c2        94818572c4ef                                                  "/bin/sh -c 'sleep..."   14 s
econds ago       Up 12 seconds                                 k8s_container-0_test1-dbc59fc55-8gr9f_defau
lt_19f0d2a0-94ba-11e9-aa5f-fa163e07fc60_1
492b258c1e89        94818572c4ef                                                  "/bin/sh -c 'sleep..."   Abou
t a minute ago   Exited (1) 14 seconds ago                     k8s_container-0_test1-dbc59fc55-8gr9f_defau
lt_19f0d2a0-94ba-11e9-aa5f-fa163e07fc60_0
2fcd00990111        cfe-pause:11.23.1                                             "/pause"                 Abou
t a minute ago   Up About a minute                            k8s_POD_test1-dbc59fc55-8gr9f_default_19f0d
2a0-94ba-11e9-aa5f-fa163e07fc60_0
[root@[redacted] ~]# docker logs 492b258c1e89
cat: /tmp/test: No such file or directory
```

解决方案：

核查“工作负载详情>更新升级页签>高级配置>启动命令”中的信息，确保启动命令配置正确。

排查项九：用户自身业务 BUG

请检查工作负载启动命令是否正确执行，或工作负载本身 bug 导致容器不断重启。

步骤 1 登录异常工作负载所在的节点。

步骤 2 查看工作负载实例非正常退出的容器 ID。

```
docker ps -a | grep $podName
```

步骤 3 查看退出容器的错误日志。

```
docker logs $containerID
```

注意：这里的 containerID 为已退出的容器的 ID

图6-7 容器启动命令配置不正确

```
[root@dcb-ha-11638 ~]# docker ps -a |grep nginx
cf0352f617f9        3f8a4339aadd                "/bin/bash /tmp/test."   2 minutes ago
      Exited (127) 2 minutes ago                          k8s_container-0_nginx-267
0177225-kt929_test_d6402ef7-4e0f-11e8-b4f7-fa163e74044e_5
c2176ce394a1        cfe-pause:3.7.6             "/pause"                 5 minutes ago
      Up 5 minutes                                        k8s_POD_nginx-2670177225-
kt929_test_d6402ef7-4e0f-11e8-b4f7-fa163e74044e_0
[root@dcb-ha-11638 ~]# docker logs cf035
/bin/bash: /tmp/test.sh: No such file or directory
[root@dcb-ha-11638 ~]#
```

如上图所示，容器配置的启动命令不正确导致容器启动失败。其他错误请根据日志提示修复工作负载本身的 BUG 问题。

----结束

解决方案：

重新创建工作负载，并配置正确的启动命令。

6.1.5 工作负载异常：实例驱逐异常（Evicted）

Eviction 介绍

Eviction，即驱逐的意思，意思是当节点出现异常时，为了保证工作负载的可用性，kubernetes 将有相应的机制驱逐该节点上的 Pod。

目前 kubernetes 中存在两种 eviction 机制，分别由 **kube-controller-manager** 和 **kubelet** 实现。

- **kube-controller-manager 实现的 eviction**

  kube-controller-manager 主要由多个控制器构成，而 eviction 的功能主要由 node controller 这个控制器实现。该 Eviction 会周期性检查所有节点状态，当节点处于 NotReady 状态超过一段时间后，驱逐该节点上所有 pod。

  kube-controller-manager 提供了以下启动参数控制 eviction：

  - **pod-eviction-timeout**：即当节点宕机该时间间隔后，开始 eviction 机制，驱赶宕机节点上的 Pod，默认为 5min。
  - **node-eviction-rate**：驱赶速率，即驱赶 Node 的速率，由令牌桶流控算法实现，默认为 0.1，即每秒驱赶 0.1 个节点，注意这里不是驱赶 Pod 的速率，而是驱赶节点的速率。相当于每隔 10s，清空一个节点。
  - **secondary-node-eviction-rate**：二级驱赶速率，当集群中宕机节点过多时，相应的驱赶速率也降低，默认为 0.01。
  - **unhealthy-zone-threshold**：不健康 zone 阈值，会影响什么时候开启二级驱赶速率，默认为 0.55，即当该 zone 中节点宕机数目超过 55%，而认为该 zone 不健康。
  - **large-cluster-size-threshold**：大集群阈值，当该 zone 的节点多于该阈值时，则认为该 zone 是一个大集群。大集群节点宕机数目超过 55%时，则将驱赶速率降为 0.01，假如是小集群，则将速率直接降为 0。

- **kubelet 的 eviction 机制**

  如果节点处于资源压力，那么 kubelet 就会执行驱逐策略。驱逐会考虑 Pod 的优先级，资源使用和资源申请。当优先级相同时，资源使用/资源申请最大的 Pod 会被首先驱逐。

  kube-controller-manager 的 eviction 机制是粗粒度的，即驱赶一个节点上的所有 pod，而 kubelet 则是细粒度的，它驱赶的是节点上的某些 Pod，驱赶哪些 Pod 与 Pod 的 Qos 机制有关。该 Eviction 会周期性检查本节点内存、磁盘等资源，当资源不足时，按照优先级驱逐部分 pod。

  驱逐阈值分为软驱逐阈值（Soft Eviction Thresholds）和强制驱逐（Hard Eviction Thresholds）两种机制，如下：

  - **软驱逐阈值**：当 node 的内存/磁盘空间达到一定的阈值后，kubelet 不会马上回收资源，如果改善到低于阈值就不进行驱逐，若这段时间一直高于阈值就进行驱逐。
  - **强制驱逐**：强制驱逐机制则简单的多，一旦达到阈值，直接把 pod 从本地驱逐。

  kubelet 提供了以下参数控制 eviction：

- **eviction-soft**：软驱逐阈值设置，具有一系列阈值，比如 memory.available<1.5Gi 时，它不会立即执行 pod eviction，而会等待 eviction-soft-grace-period 时间，假如该时间过后，依然还是达到了 eviction-soft，则触发一次 pod eviction。
- **eviction-soft-grace-period**：默认为 90 秒，当 eviction-soft 时，终止 Pod 的 grace 的时间，即软驱逐宽限期，软驱逐信号与驱逐处理之间的时间差。
- **eviction-max-pod-grace-period**：最大驱逐 pod 宽限期，停止信号与 kill 之间的时间差。
- **eviction-pressure-transition-period**：默认为 5 分钟，驱逐压力过渡时间，超过阈值时，节点会被设置为 memory pressure 或者 disk pressure，然后开启 pod eviction。
- **eviction-minimum-reclaim**：表示每一次 eviction 必须至少回收多少资源。
- **eviction-hard**：强制驱逐设置，也具有一系列的阈值，比如 memory.available<1Gi，即当节点可用内存低于 1Gi 时，会立即触发一次 pod eviction。

问题定位

若节点故障时，实例未被驱逐，请先按照如下方法进行问题定位。

使用如下命令发现很多 pod 的状态为 Evicted：

```
kubectl get pods
```

在节点的 kubelet 日志中会记录 Evicted 相关内容，搜索方法可参考如下命令：

```
cat /var/paas/sys/log/kubernetes/kubelet.log | grep -i Evicted -C3
```

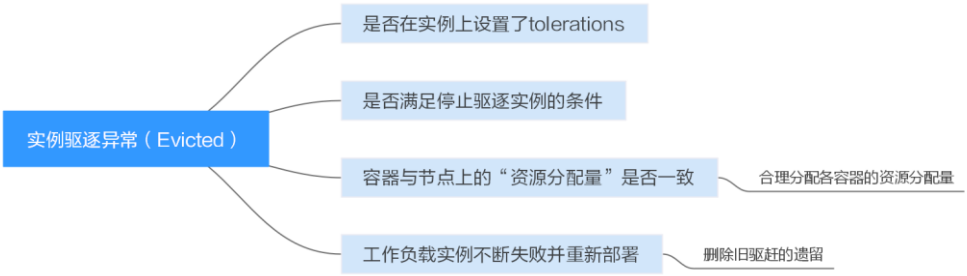
排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频率原因往低频率原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- 排查项一：是否在实例上设置了 tolerations
- 排查项二：是否满足停止驱逐实例的条件
- 排查项三：容器与节点上的“资源分配量”是否一致
- 排查项四：工作负载实例不断失败并重新部署

图6-8 排查思路



排查项一：是否在实例上设置了 tolerations

通过 kubectl 工具或单击对应工作负载后的“更多 > 编辑 YAML”，检查工作负载上是不是打上了 tolerations，具体请参见 https://kubernetes.io/docs/concepts/configuration/taint-and-toleration/。

排查项二：是否满足停止驱逐实例的条件

若属于小规格的集群（集群节点数小于 50 个节点），如果故障的节点大于总节点数的 55%，实例的驱逐将会被暂停。此情况下 k8s 将部署尝试驱逐故障节点的工作负载，具体请参见 https://kubernetes.io/docs/concepts/architecture/nodes/。

排查项三：容器与节点上的“资源分配量”是否一致

容器被驱逐后还会频繁调度到原节点。

问题原因：

节点驱逐容器是根据节点的“资源使用率”进行判断；容器的调度规则是根据节点上的“资源分配量”进行判断。由于判断标准不同，所以可能会出现被驱逐后又再次被调度到原节点的情况。

解决方案：

遇到此类问题时，请合理分配各容器的资源分配量即可解决。

排查项四：工作负载实例不断失败并重新部署

工作负载实例出现不断失败，不断重新部署的情况。

问题分析：

pod 驱逐后，如果新调度到的节点也有驱逐情况，就会再次被驱逐；甚至出现 pod 不断被驱逐的情况。

如果是由 kube-controller-manager 触发的驱逐，会留下一个状态为 Terminating 的 pod；直到容器所在节点状态恢复后，pod 才会自动删除。如果节点已经删除或者其他原因导致的无法恢复，可以使用“强制删除”删除 pod。

如果是由 kubelet 触发的驱逐，会留下一个状态为 Evicted 的 pod，此 pod 只是方便后期定位的记录，可以直接删除。

解决方案：

使用如下命令删除旧驱赶的遗留：

```
kubectl get pods <namespace> | grep Evicted | awk '{print $1}' | xargs kubectl delete pod <namespace>
```

<*namespace*>为命名空间名称，请根据需要指定。

参考

Kubelet does not delete evicted pods

6.1.6 工作负载异常：存储卷无法挂载或挂载超时

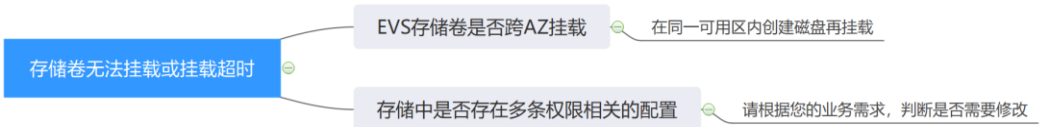
排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频率原因往低频率原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- 排查项一：EVS 存储卷是否跨 AZ 挂载
- 排查项二：存储中是否同时存在多条权限相关的配置
- 排查项三：带云硬盘卷的 Deployment 的副本数大于 1
- 排查项四：EVS 磁盘文件系统损坏

图6-9 排查思路



排查项一：EVS 存储卷是否跨 AZ 挂载

问题描述：

客户在有状态工作负载上挂载 EVS 存储卷，但无法挂载卷并超时。

问题定位：

经查询确认，该节点在**可用区 1**，而要挂载的磁盘在**可用区 2**，导致无法挂载而超时。

解决方案：

在同一可用区内创建磁盘再挂载后即可正常。

排查项二：存储中是否同时存在多条权限相关的配置

如果您挂载的存储中内容太多，同时又配置了以下几条配置，最终会由于逐个修改文件权限，而导致挂载时间过长。

问题定位：

- Securitycontext 字段中是否包含 runAsuser/fsGroup。securityContext 是 kubernetes 中的字段，即安全上下文，它用于定义 Pod 或 Container 的权限和访问控制设置。
- 启动命令中是否包含 ls、chmod、chown 等查询或修改文件权限的操作。

解决建议：

请根据您的业务需求，判断是否需要修改。

排查项三：带云硬盘卷的 Deployment 的副本数大于 1

问题描述：

创建 Pod 失败，并报“添加存储失败”的事件，事件信息如下。

```
Multi-Attach error for volume "pvc-62a7a7d9-9dc8-42a2-8366-0f5ef9db5b60" Volume is already used by pod(s) testttt-7b774658cb-lc98h
```

问题定位：

查看 Deployment 的副本数是否大于 1。

Deployment 中使用 EVS 存储卷时，副本数只能为 1。若用户在后台指定 Deployment 的实例数为 2 以上，此时 CCE 并不会限制 Deployment 的创建。但若这些实例 Pod 被调度到不同的节点，则会有部分 Pod 因为其要使用的 EVS 无法被挂载到节点，导致 Pod 无法启动成功。

解决方案：

使用 EVS 的 Deployment 的副本数指定为 1，或使用其他类型存储卷。

排查项四：EVS 磁盘文件系统损坏

问题描述：

创建 Pod 失败，出现类似信息，磁盘文件系统损坏。

```
MountVolume.MountDevice failed for volume "pvc-08178474-c58c-4820-a828-14437d46ba6f" : rpc error: code = Internal desc = [09060def-afd0-11ec-9664-fa163eef47d0] /dev/sda has file system, but it is detected to be damaged
```

解决方案：

在 EVS 中对磁盘进行备份，然后执行如下命令修复文件系统。

fsck -y {盘符}

6.1.7 工作负载异常：一直处于创建中

问题描述

节点变更之后，节点上的工作负载一直处于创建中。

解决方法

步骤 1 登录 CCE 节点（弹性云服务器）并删除 cpu_manager_state 文件。

删除命令示例如下：

```
rm -rf /mnt/paas/kubernetes/kubelet/cpu_manager_state
```

步骤 2 重启节点或重启 kubelet，重启 kubelet 的方法如下：

```
systemctl restart kubelet
```

此时重新拉起或创建工作负载，已可成功执行。

解决方式链接：CCE 节点变更规格后，为什么无法重新拉起或创建工作负载？

----结束

6.1.8 工作负载异常：结束中，解决 Terminating 状态的 Pod 删不掉的问题

问题描述

在节点处于“不可用”状态时，CCE 会迁移节点上的容器实例，并将节点上运行的 pod 置为“Terminating”状态。

待节点恢复后，处于“Terminating”状态的 pod 会自动删除。

偶现部分 pod（实例）一直处于“Terminating ”状态：

```
#kubectl get pod -n aos
NAME                          READY     STATUS        RESTARTS   AGE
aos-apiserver-5f8f5b5585-s9192    1/1       Terminating    0          3d1h
aos-cmdbserver-789bf5b497-6rwrg   1/1       Running        0          3d1h
aos-controller-545d78bs8d-vm6j9   1/1       Running        3          3d1h
```

通过 kubectl delete pods <podname> -n <namespace> 命令始终无法将其删除：

```
kubectl delete pods aos-apiserver-5f8f5b5585-s9192 -n aos
```

解决方法

无论各种方式生成的 pod，均可以使用如下命令强制删除：

```
kubectl delete pods <pod> --grace-period=0 --force
```

因此对于上面的 pod，我们只要执行如下命令即可删除：

```
kubectl delete pods aos-apiserver-5f8f5b5585-s9192 --grace-period=0 --force
```

6.1.9 工作负载异常：已停止

问题现象

工作负载的状态为“已停止”。

问题原因：

工作负载的 yaml 的中 metadata.enable 字段为 false，导致工作负载被停止，Pod 被删除导致工作负载处于已停止状态，如下图所示：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: test
  namespace: default
  selfLink: /apis/apps/v1/namespaces/default/deployments/test
  uid: b130db9f-9306-11e9-a2a9-fa163eaff9f7
  resourceVersion: '7314771'
  generation: 1
  creationTimestamp: '2019-06-20T02:54:16Z'
  labels:
    appgroup: ''
  annotations:
    deployment.kubernetes.io/revision: '1'
    description: ''
  enable: false
spec:
```

解决方案

将 enable 字段删除或者将 false 修改为 true。

6.1.10 工作负载异常：GPU 节点部署服务报错

问题现象

客户在 CCE 集群的 GPU 节点上部署服务出现如下问题：

1. 容器无法查看显存。
2. 部署了 7 个 GPU 服务，有 2 个是能正常访问的，其他启动时都有报错。
   - 2 个是能正常访问的 CUDA 版本分别是 10.1 和 10.0
   - 其他服务 CUDA 版本也在这 2 个范围内
3. 在 GPU 服务容器中发现一些新增的文件 core.*，在以前的部署中没有出现过。

问题定位

4. gpu-beta 插件的驱动版本较低，客户单独下载驱动安装后正常。
5. 客户工作负载中未声明需要 gpu 资源。

建议方案

节点安装了 gpu-beta 插件后，nvidia-smi 命令行工具在/opt/cloud/cce/nvidia/bin 目录下。如果插件安装后，依然没有这个命令行工具，通常是由于 nvidia 驱动安装失败。请排查 nvidia 驱动是否下载成功。（在/opt/cloud/cce/nvidia 目录下可以看到驱动文件）

如果驱动地址填写错误，需要将插件卸载后重新安装，并配置正确的地址。

说明

nvidia 驱动建议放在 OBS 桶里，并设置为公共读。

相关链接

- GPU 节点使用 nvidia 驱动启动容器排查思路

6.1.11 实例网络空间更新，报 sandbox 相关错，如何处理？

问题现象

实例一直处于创建中，报错 sandbox 相关错，如下所示。

```
Failed create pod sandbox: rpc error: code = Unknown desc = [ failed to setup
sandbox .....
```

解决方案

排查方法如下：

1.13 版本的集群：

说明

此方法仅支持 1.13 版本集群。

1. sandbox 错误一般为节点上容器组件启动有异常，systemctl status canal 查看节点的容器组件，重启组件命令如：systemctl restart canal。
2. 节点上容器组件全部正常，network: failed to find plugin "loopback" in path [/opt/cni/bin]，cni 的文件 loopback 缺失导致，可以从其他局点或者当前局点（最好集群版本配套，可忽略小版本）拷贝一份完整的 loopback 文件，放入/opt/cni/bin/下，然后重启 canal 组件。

1.13 版本之前的集群：

3. sandbox 错误一般为节点上容器组件启动有异常，su paas -c '/var/paas/monit/bin/monit summary' 查看节点的容器组件，重启组件命令如：su paas -c '/var/paas/monit/bin/monit restart canal'。

4. 节点上容器组件全部正常，network: failed to find plugin "loopback" in path [/opt/cni/bin]，cni 的文件 loopback 缺失导致，可以从其他局点或者当前局点（最好集群版本配套，可忽略小版本）拷贝一份完整的 loopback 文件，放入 /opt/cni/bin/下，然后重启 canal 组件。

6.1.12 工作负载异常：实例无法写入数据

Pod 事件

Pod 所在的节点文件系统损坏，新建的 Pod 无法成功在/var/lib/kubelet/device-plugins/.xxxxx 写入数据，Pod 通常会出现以下类似事件：

```
Message: Pod Update Plugin resources failed due to failed to write checkpoint file "kubelet_internal_checkpoint": open /var/lib/kubelet/device-plugins/.xxxxxx: read-only file system, which is unexpected.
```

此类异常 Pod 仅为异常记录，并不实际占用系统资源。

排查步骤

导致文件系统异常的原因有很多，例如物理控制节点的异常开关机。此类异常 Pod 并不影响正常业务，当系统文件未能恢复，出现大量异常 Pod 时，可采取以下步骤进行规避排查：

步骤 1 执行以下命令，将该 Node 标记为不可调度，并将已有 Pod 驱逐到其他节点。

```
kubectl drain <node-name>
```

步骤 2 等待 Pod 调度到其他节点后，排查文件系统异常的原因，并进行恢复或规避。

步骤 3 执行以下命令，取消节点不可调度标记。

```
kubectl uncordon <node-name>
```

----结束

异常 Pod 清理

- 本服务 kubelet 的 GC 回收机制与社区保持一致，在清除 Pod 的 Owner（例如 Deployment）后，异常 Pod 也会随之清理。
- 通过 kubelet 命令，删除有异常记录的 Pod。

6.1.13 挂载文件存储的节点，Pod 创建删除卡死

故障现象

在挂载文件存储（SFS 或 SFS Turbo）的节点上，删除 Pod 卡在“结束中”，创建 Pod 卡在“创建中”。

可能原因

- 后端文件存储被删除，导致无法访问挂载点。
- 节点与文件存储间网络异常，导致无法访问挂载点。

解决方案

步骤 1 登录挂载文件存储的节点，执行如下命令找到文件存储挂载路径。

findmnt

挂载点路径示例：/mnt/paas/kubernetes/kubelet/pods/7b88feaf-71d6-4e6f-8965-f5f0766d9f35/volumes/kubernetes.io~csi/sfs-turbo-ls/mount

步骤 2 执行如下命令尝试进入文件存储文件夹。

cd /mnt/paas/kubernetes/kubelet/pods/7b88feaf-71d6-4e6f-8965-f5f0766d9f35/volumes/kubernetes.io~csi/sfs-turbo-ls/mount

如果不能正确进入，则说明文件存储被删除或文件存储与节点间网络异常。

步骤 3 执行 umount -l 命令解除挂载。

umount -l /mnt/paas/kubernetes/kubelet/pods/7b88feaf-71d6-4e6f-8965-f5f0766d9f35/volumes/kubernetes.io~csi/sfs-turbo-ls/mount

步骤 4 重启 kubelet。

systemctl restart kubelet

----结束

问题根因

该问题常见于文件存储挂载模式为 hard 的场景，在 hard 模式下，所有访问挂载点的进程都会 Hang 住，直到访问成功。使用 soft 模式挂载可以避免该情况。

6.1.14 容器异常退出状态码

当容器启动失败或终止时，K8s 事件中将会打印容器异常退出状态码（Exit Code）来报告容器异常的原因。本文将介绍如何通过事件中打印的 Exit Code 进一步定位容器异常的根本原因。

查看容器异常退出状态码

您可使用 kubectl 连接集群，并通过以下命令查询 Pod 详细状态：

```
kubectl describe pod {pod name}
```

在返回结果中的 Exit Code 字段即为程序上次退出时的状态码，该值不为 0 即表示程序异常退出，可根据退出状态码进一步分析异常原因。

```
Containers:
  container-1:
    Container ID: ...
    Image: ...
    Image ID: ...
    Ports: ...
    Host Ports: ...
    Args: ...
    State: Running
      Started: Sat, 28 Jan 2023 09:06:53 +0000
    Last State: Terminated
      Reason: Error
      Exit Code: 255
      Started: Sat, 28 Jan 2023 09:01:33 +0000
      Finished: Sat, 28 Jan 2023 09:05:11 +0000
    Ready: True
    Restart Count: 1
```

退出状态码说明

容器退出状态码的范围为 0~255 之间：

- 0 表示正常退出。
- 一般由于程序自身原因导致的异常退出，状态码区间在 1~128。在特殊场景下，程序也可以使用 129~255 区间的状态码。
- 由于外界中断导致程序退出时，状态码区间在 129~255。当操作系统给程序发送中断信号时，程序退出时的状态码为操作系统的中断信号值加 128。例如，SIGKILL（强制停止）的中断信号值为 9，那么程序退出状态码则为 9+128 = 137。
- 当指定的状态码不在 0~255 范围内时（例如使用 exit(-1)），将自动转化至 0~255 范围内。

  当指定的状态码为正数，转换公式如下：

  ```
  code % 256
  ```

  当指定的状态码为负数，转换公式如下：

  ```
  256 - (|code| % 256)
  ```

更多退出状态码说明请参见 Exit Codes With Special Meanings。

常见的容器异常退出状态码

表6-5 常见的容器异常退出状态码

容器异常退出状态码	名称	含义
0	正常退出	表示容器正常退出。该状态码不一定表示存在异常，

容器异常退出状态码	名称	含义
		当容器内没有进程时，例如，也会出现退出状态码为 0 的场景。
1	一般程序错误	引起该异常状态的原因较多，大多由于程序自身错误导致，需要进一步通过容器日志定位原因。
125	容器未能运行	发生这种情况的常见原因有如下几种： • 命令中使用了未定义的 flag，例如 docker run --abcd。 • 镜像中用户定义的命令在本机权限不足。 • 容器引擎与宿主机操作系统或硬件不兼容。
126	命令调用错误	镜像中调用的命令无法执行，例如文件权限不足或文件不可执行。
127	找不到文件或目录	无法找到镜像中指定的文件或目录。
137	立即终止（SIGKILL）	表示程序被 SIGKILL 信号终止，常见的原因有如下几种： • Pod 中容器内存达到了其资源限制（resources.limits）。例如，内存溢出（OOM）会导致 cgroup 强制停止该容器。 • 运行容器的节点本身资源不足（OOM），则节点内核会选择停止一些进程来释放内存，可能会导致容器被终止。 • 容器健康检查失败，kubelet 会停止该容器。 • 其他外部进程强制停止容器，例如恶意脚本。
139	分段错误（SIGSEGV）	表示容器收到了来自操作系统的 SIGSEGV 信号，由于容器试图访问无权限的内存位置引起。
143	优雅终止（SIGTERM）	表示容器在主机指示后正确关闭。一般来说，退出码 143 不需要进行故障排除。
255	状态码超出范围	表示容器退出状态码超出范围。例如，可能是设置异常退出使用 exit(-1)导致的，而-1 将会自动转换成 255。 出现该异常时无法判断原因，需要进一步通过容器日志定位原因。

Linux 标准中断信号

您可以使用 **kill -l** 命令查看 Linux 操作系统中信号以及对应的数值。

表6-6 常用的 Linux 标准中断信号

信号（Signal）	状态码（Value）	动作（Action）	描述（Commit）
SIGHUP	1	Term	用户终端连接（正常或非正常）结束时发出
SIGINT	2	Term	程序终止信号，通常由终端发出中断指令，例如键盘输入 Ctrl+C
SIGQUIT	3	Core	和 SIGINT 类似，由终端发出退出指令，通常是键盘输入 Ctrl+\来控制
SIGILL	4	Core	非法指令，通常是因为可执行文件本身出现错误
SIGABRT	6	Core	调用 abort 函数时产生的信号，进程会非正常结束
SIGFPE	8	Core	发生浮点运算错误，出现除数为 0 等其它算术错误时也会产生
SIGKILL	9	Term	终止任何进程
SIGSEGV	11	Core	试图访问无权限的内存位置
SIGPIPE	13	Term	管道断开信号
SIGALRM	14	Term	时钟定时信号
SIGTERM	15	Term	进程结束信号，通常是程序自行正常退出
SIGUSR1	10	Term	用户在应用程序中自行定义的信号
SIGUSR2	12	Term	用户在应用程序中自行定义的信号
SIGCHLD	17	Ign	子进程结束或中断时产生该信号
SIGCONT	18	Cont	让一个暂停（stopped）的进程继续执行
SIGSTOP	19	Stop	暂停进程的执行
SIGTSTP	20	Stop	停止进程的运行
SIGTTIN	21	Stop	后台进程从终端读取输入值

信号（Signal）	状态码（Value）	动作（Action）	描述（Commit）
SIGTTOU	22	Stop	后台进程从终端读取输出值

6.2 容器设置

6.2.1 在什么场景下设置工作负载生命周期中的“停止前处理”？

服务的业务处理时间较长，在升级时，需要先等 Pod 中的业务处理完，才能 kill 该 Pod，以保证业务不中断的场景。

6.2.2 在同一个命名空间内访问指定容器的 FQDN 是什么？

问题背景

客户询问在创建负载时指定部署的容器名称、pod 名称、namespace 名称，在同一个命名空间内访问该容器的 FQDN 是什么？

全限定域名：FQDN，即 Fully Qualified Domain Name，同时带有主机名和域名的名称。（通过符号“.”）

例如：主机名是 bigserver，域名是 mycompany.com，那么 FQDN 就是：bigserver.mycompany.com。

问题建议

方案一：发布服务使用域名发现，需要提前预制好主机名和命名空间，服务发现使用域名的方式，注册的服务的域名为：服务名.命名空间.svc.cluster.local 。这种使用有限制，注册中心部署必须容器化部署。

方案二：容器部署使用主机网络部署，然后亲和到集群的某一个节点，这样可以明确知道容器的服务地址（就是节点的地址），注册的地址为：服务所在节点 IP，这种方案可以满足注册中心利用 VM 部署，缺陷是使用主机网络效率没有容器网络高。

6.2.3 健康检查探针（Liveness、Readiness）偶现检查失败?

健康检查探针偶现检测失败，是由于容器内的业务故障所导致，您需要优先定位自身业务问题。

常见情况有：

- 业务处理时间长，导致返回超时。
- tomcat 建链和等到耗费时间太长（连接数、线程数等），导致返回超时。
- 容器所在节点，磁盘 IO 等性能达到瓶颈，导致业务处理超时。

6.2.4 如何设置容器 umask 值？

问题描述

tailf /dev/null 的方式启动容器，然后手动执行启动脚本的方式得到的目录的权限是 700，而不加 tailf 由 Kubernetes 自行启动的方式得到的目录权限却是 751。

操作步骤

这个问题是因为两种方式设置的 umask 值不一样，所以创建出来的目录权限不相同。

umask 值用于为用户新创建的文件和目录设置缺省权限。如果 umask 的值设置过小，会使群组用户或其他用户的权限过大，给系统带来安全威胁。因此设置所有用户默认的 umask 值为 0077，即用户创建的目录默认权限为 700，文件的默认权限为 600。

可以在启动脚本里面增加如下内容实现创建出来的目录权限为 700：

1. 分别在**/etc/bashrc** 文件和**/etc/profile.d/**目录下的所有文件中加入“umask 0077”。
2. 执行如下命令：

```
echo "umask 0077" >> $FILE
```

说明

FILE 为具体的文件名，例如：echo “umask 0077” >> /etc/bashrc

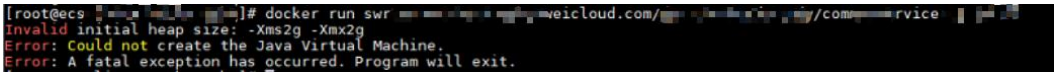
3. 设置**/etc/bashrc** 文件和**/etc/profile.d/**目录下所有文件的属主为：root，群组为：root。
4. 执行如下命令：

```
chown root.root $FILE
```

6.2.5 Dockerfile 中 ENTRYPOINT 指定 JVM 启动堆内存参数后部署容器启动报错？

问题描述

Dockerfile 中 ENTRYPOINT 指定 JVM 启动堆内存参数后部署容器启动报错，报错信息为：invalid initial heap size，如下图：



操作步骤

请检查 ENTRYPOINT 设置，下方的设置是错误的：

```
ENTRYPOINT ["java","-Xms2g -Xmx2g","-jar","xxx.jar"]
```

如下两种办法可以解决该问题：

- （**推荐**）将容器启动命令写在“工作负载 > 更新升级 > 容器设置 > 生命周期 > 启动命令”这里，容器能正常启动。

- 将 ENTRYPOINT 启动命令修改为如下格式：

```
ENTRYPOINT exec java -Xmx2g -Xms2g -jar xxxx.jar
```

6.2.6 CCE 启动实例失败时的重试机制是怎样的？

CCE 是基于原生 Kubernetes 的云容器引擎服务，完全兼容 Kubernetes 社区原生版本，与社区最新版本保持紧密同步，完全兼容 Kubernetes API 和 Kubectl。

在 Kubernetes 中，Pod 的 spec 中包含一个 restartPolicy 字段，其取值包括：Always、OnFailure 和 Never，默认值为：Always。

- Always：当容器失效时，由 kubelet 自动重启该容器。
- OnFailure：当容器终止运行且退出不为 0 时（正常退出），由 kubelet 自动重启该容器。
- Never：不论容器运行状态如何，kubelet 都不会重启该容器。

restartPolicy 适用于 Pod 中的所有容器。

restartPolicy 仅针对同一节点上 kubelet 的容器重启动作。当 Pod 中的容器退出时，kubelet 会按指数回退方式计算重启的延迟（10s、20s、40s...），其最长延迟为 5 分钟。 一旦某容器执行了 10 分钟并且没有出现问题，kubelet 对该容器的重启回退计时器执行重置操作。

每种控制器对 Pod 的重启策略要求如下：

- Replication Controller（RC）和 DaemonSet：必须设置为 Always，需要保证该容器的持续运行。
- Job：OnFailure 或 Never，确保容器执行完成后不再重启。

6.3 监控告警

6.3.1 工作负载的“事件”保存多长时间？

在 1.7.3-r12、1.9.2-r3 及以上版本的集群中，工作负载的“事件”信息保存时间为 1 个小时，1 小时后自动清除数据。

在 1.7.3-r12 之前更老的集群版本中，保存时间为 24 小时。

6.4 调度策略

6.4.1 如何让多个 Pod 均匀部署到各个节点上？

Kubernetes 中 kube-scheduler 组件负责 Pod 的调度，对每一个新创建的 Pod 或者是未被调度的 Pod，kube-scheduler 会选择一个最优的节点去运行这个 Pod。kube-scheduler 给一个 Pod 做调度选择包含过滤和打分两个步骤。过滤阶段会将所有满足 Pod 调度需求的节点选出来，在打分阶段 kube-scheduler 会给每一个可调度节点进行

优先级打分，最后 kube-scheduler 会将 Pod 调度到得分最高的节点上，如果存在多个得分最高的节点，kube-scheduler 会从中随机选取一个。

打分优先级中节点调度均衡（BalancedResourceAllocation）只是其中一项，还有其他打分项会导致分布不均匀。详细的调度说明请参见 Kubernetes 调度器和调度策略。

想要让多个 Pod 尽可能的均匀分布在各个节点上，可以考虑使用工作负载反亲和特性，让 Pod 之间尽量“互斥”，这样就能尽量均匀的分布在各节点上。

示例如下：

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: container-0
          image: nginx:alpine
          resources:
            limits:
              cpu: 250m
              memory: 512Mi
            requests:
              cpu: 250m
              memory: 512Mi
      affinity:
        podAntiAffinity:                # 工作负载反亲和
          preferredDuringSchedulingIgnoredDuringExecution:    # 尽量满足如下条件
            - podAffinityTerm:
                labelSelector:                    # 选择 Pod 的标签，与工作负载本身反亲和
                  matchExpressions:
                    - key: app
                      operator: In
                      values:
                        - nginx
                namespaces:
                  - default
                topologyKey: kubernetes.io/hostname     # 在节点上起作用
      imagePullSecrets:
        - name: default-secret
```

6.4.2 如何设置一个节点上的某个容器不能被驱逐？

问题背景

客户反馈 CCE 集群中一个节点上的两个容器之间互相争资源，导致 kubelet 将其全部驱逐。询问能不能设定策略让其中一个服务一直保留？如何设定？

问题建议

Kubelet 会按照下面的标准对 Pod 的驱逐行为进行评判：

- 根据服务质量：即 BestEffort、Burstable、Guaranteed。
- 根据 Pod 调度请求的被耗尽资源的消耗量。

接下来，Pod 按照下面的顺序进行驱逐（QOS）：

BestEffort -> Burstable -> Guaranteed

- Best-Effort 类型的 pods：系统用完了全部内存时，该类型 pods 会最先被 kill 掉。
- Burstable 类型的 pods：系统用完了全部内存，且没有 Best-Effort container 可以被 kill 时，该类型 pods 会被 kill 掉。
- Guaranteed 类型的 pods：系统用完了全部内存、且没有 Burstable 与 Best-Effort container 可以被 kill，该类型的 pods 会被 kill 掉。

您可以在工作负载 yaml 中添加 QoSClass 字段设置被驱逐优先级，如下图：

说明

- 如果 pod 进程因使用超过预先设定的 limites 而非 Node 资源紧张情况，系统倾向于在其原所在的机器上重启该 container 或本机或其他重新创建一个 pod。
- 如果资源充足，可将 QoS pods 类型均设置为 Guaranteed。用计算资源换业务性能和稳定性，减少排查问题时间和成本。

- 如果想更好的提高资源利用率，业务服务可以设置为 Guaranteed，而其他服务根据重要程度可分别设置为 Burstable 或 Best-Effort，例如 filebeat。

6.4.3 为什么 Pod 在节点不是均匀分布？

Kubernetes 中 kube-scheduler 组件负责 Pod 的调度，对每一个新创建的 Pod 或者是未被调度的 Pod，kube-scheduler 会选择一个最优的节点去运行这个 Pod。kube-scheduler 给一个 Pod 做调度选择包含过滤和打分两个步骤。过滤阶段会将所有满足 Pod 调度需求的节点选出来，在打分阶段 kube-scheduler 会给每一个可调度节点进行优先级打分，最后 kube-scheduler 会将 Pod 调度到得分最高的节点上，如果存在多个得分最高的节点，kube-scheduler 会从中随机选取一个。

打分优先级中节点调度均衡（BalancedResourceAllocation）只是其中一项，还有其他打分项会导致分布不均匀。详细的调度说明请参见 Kubernetes 调度器和调度策略。

6.4.4 如何驱逐节点上的所有 Pod？

您可使用 **kubectl drain** 命令从节点安全地逐出所有 Pod。

说明

默认情况下，**kubectl drain** 命令会保留某些系统级 Pod 不被驱逐，例如 everest-csi-driver。

步骤 1 使用 kubectl 连接集群。

步骤 2 查看集群中的节点。

```
kubectl get node
```

步骤 3 选择一个节点，查看节点上存在的所有 Pod。

```
kubectl get pod --all-namespaces -owide --field-selector
spec.nodeName=192.168.0.160
```

驱逐前该节点上的 Pod 如下：

```
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
IP            NODE          NOMINATED NODE   READINESS GATES
default       nginx-5bcc57c74b-lgcvh                1/1     Running   0          7m25s
10.0.0.140    192.168.0.160   <none>           <none>
kube-system   coredns-6fcd88c4c-97p6s               1/1     Running   0          3h16m
10.0.0.138    192.168.0.160   <none>           <none>
kube-system   everest-csi-controller-56796f47cc-99dtm   1/1     Running   0
3h16m   10.0.0.139    192.168.0.160   <none>           <none>
kube-system   everest-csi-driver-dpfzl              2/2     Running   2          12d
192.168.0.160   192.168.0.160   <none>           <none>
kube-system   icagent-tpfpv                         1/1     Running   1          12d
192.168.0.160   192.168.0.160   <none>           <none>
```

步骤 4 驱逐该节点上的所有 Pod。

```
kubectl drain 192.168.0.160
```

如果节点上存在绑定了本地存储的 Pod 或是一些守护进程集管理的 Pod，将提示“error: unable to drain node "192.168.0.160", aborting command...”。驱逐命令将不会生效，您可在上述命令后面添加如下参数进行强制驱逐：

- --delete-emptydir-data：强制驱逐节点上绑定了本地存储的 Pod，例如 coredns。
- --ignore-daemonsets：忽略节点上的守护进程集 Pod，例如 everest-csi-driver。

示例中节点上存在绑定本地存储的 Pod 和守护进程集 Pod，因此驱逐命令如下：

```
kubectl drain 192.168.0.160 --delete-emptydir-data --ignore-daemonsets
```

步骤 5 驱逐成功后，该节点被自动标记为不可调度，即该节点将会被打上 **node.kubernetes.io/unschedulable = : NoSchedule** 的污点。

驱逐后该节点上的 Pod 如下，节点上仅保留了不可驱逐的系统级 Pod。

```
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE   IP
NODE          NOMINATED NODE   READINESS GATES
kube-system   everest-csi-driver-dpfzl   2/2     Running   2          12d
192.168.0.160   192.168.0.160   <none>          <none>
kube-system   icagent-tpfpv              1/1     Running   1          12d   192.168.0.160
192.168.0.160   <none>          <none>
```

----结束

相关操作

kubectl 的 drain、cordon 和 uncordon 操作：

- drain：从节点安全地逐出所有 Pod，并将该节点标记为不可调度。
- cordon：将节点标记为不可调度，即该节点将会被打上 **node.kubernetes.io/unschedulable = : NoSchedule** 的污点。
- uncordon：将节点标记为可调度。

  更多说明请参考 kubectl 文档。

6.4.5 如何查看 Pod 是否使用 CPU 绑核？

以 4U8G 节点为例，并提前在集群中部署一个 CPU request 为 1，limit 为 2 的工作负载。

步骤 1 登录到节点池中的一个节点，查看/var/lib/kubelet/cpu_manager_state 输出内容。

```
cat /var/lib/kubelet/cpu_manager_state
```

回显如下：

```
{"policyName":"static","defaultCpuSet":"0,2-3","entries":{"c1fcd22d-8a83-4aef-a27a-4c037e482b16":{"container-1":"1"}},"checksum":1500530529}
```

policyName 字段值为 static 代表策略设置成功。

步骤 2 查看容器的 cpuset.preferred_cpus 的 cgroup 设置，输出内容即为优先使用的 CPU 号。

```
cat /sys/fs/cgroup/cpuset/kubepods/pod{pod uid}/{容器 id}/cpuset.cpus
```

驱逐前该节点上的 Pod 如下：

- {pod uid}为 Pod UID，可在已通过 kubectl 连接集群的机器上使用以下命令获取：

```
kubectl get po {pod name} -n {namespace} -ojsonpath='{.metadata.uid}{"\n"}'
```

命令中的{pod name}和{namespace}是 Pod 名称及其所在的命名空间。

- {容器 id}需要是完整的容器 ID，可在容器运行的节点上通过以下命令获取：

docker 节点池：

```
docker ps --no-trunc | grep {pod name} | grep -v cce-pause | awk '{print $1}'
```

containerd 节点池：

```
crictl ps --no-trunc | grep {容器名称} | grep -v cce-pause | awk '{print $1}'
```

完整示例如下：

```
cat /sys/fs/cgroup/cpuset/kubepods/podc1fcd22d-8a83-4aef-a27a-
4c037e482b16/5cb15f55f429e4496172bef05994477caa96e0ca468563208695c1ad5cc141e0/cpuse
t.cpus
```

回显如下，表示绑定 1 号 CPU。

```
1
```

6.4.6 节点关机后 Pod 不重新调度

问题现象

节点关机后，节点上的 Pod 仍然显示 running 状态。通过 kubectl describe pod <pod-name>命令查询 Pod 最新事件为：

```
Warning NodeNotReady 17s node-controller Node is not ready
```

问题原因

节点关机后，系统会自动给节点添加污点，比如：

- node.kubernetes.io/unreachable:NoExecute
- node.cloudprovider.kubernetes.io/shutdown:NoSchedule
- node.kubernetes.io/unreachable:NoSchedule
- node.kubernetes.io/not-ready:NoExecute

当 Pod 对这些污点存在容忍策略时，Pod 不会进行重新调度，因此需要检查 Pod 对污点的容忍策略。

解决方案

通过查询 Pod 或者工作负载的 yaml，查看容忍策略。一般情况下，工作负载的容忍度设置由以下字段组成：

```
tolerations:
- key: "key1"
  operator: "Equal"
  value: "value1"
  effect: "NoSchedule"
```

或者：

```
tolerations:
- key: "key1"
  operator: "Exists"
  effect: "NoSchedule"
```

当上述容忍度的设置填写错误时，可能会出现调度问题。例如以下的容忍策略：

```
tolerations:
- operator: "Exists"
```

上述例子中只填写了 operator 参数为 Exists（此时容忍度不能指定 value 参数）。

- 当一个容忍度的 operator 参数为 Exists 但 key 为空时， 表示这个容忍度与任意的 key、value 和 effect 都匹配，即这个容忍度能容忍任何污点。
- 如果 effect 为空但键名 key 已填写，则表示与所有键名 key 的效果相匹配。

关于 Kubernetes 容忍度的详细说明，请参见污点和容忍度。

因此，需要修改工作负载的 yaml，还原 tolerations 为默认配置如下：

```
tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
```

上述默认的容忍度表示 Pod 可以在拥有以上污点的节点上运行 300s，然后会被驱逐。

6.5 其他

6.5.1 定时任务停止一段时间后，为何无法重新启动？

定时任务在运行过程中，如果被暂停，再次被开启时，会根据最后一次的成功时间跟当前的时间计算时间差，然后与定时的周期*100 作对比，如果时间差大于单次周期时长*100，后期的定时任务就不会被触发，详情请参考 CronJob 限制。

例如，假设一个 CronJob 被设置为从 08:30:00 开始每隔 1 分钟创建一个新的 Job，且 startingDeadlineSeconds 字段未被设置。如果 CronJob 控制器从 08:29:00 到 10:21:00 终止运行，则该 Job 将不会启动，因为从 08:29:00 到 10:21:00 超过了 100 分钟，即错过的调度次数超过了 100（示例中一个调度周期为 1 分钟）。

但如果设置了 startingDeadlineSeconds 字段，则控制器会统计从 startingDeadlineSeconds 设置的值到现在的时间，计算期间错过了多少次 Job。 例如，如果 startingDeadlineSeconds 是 200，则控制器会统计在过去 200 秒中错过了多少次 Job。此时如果 CronJob 控制器同样在 08:29:00 到 10:21:00 时间段终止运行，则 Job 仍将从 10:22:00 开始，因为最近 200 秒中仅错过了 3 个调度（示例中一个调度周期为 1 分钟）。

解决方法

如果想要解决这个问题，可以在定时任务的 CronJob 中配置参数：startingDeadlineSeconds。该参数只能使用 kubectl 命令，或者通过 API 接口进行创建或修改。

YAML 示例如下：

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: hello
spec:
  startingDeadlineSeconds: 200
  schedule: "* * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: hello
            image: busybox:1.28
            imagePullPolicy: IfNotPresent
            command:
            - /bin/sh
            - -c
            - date; echo Hello
          restartPolicy: OnFailure
```

当然，如果重新创建 CronJob，也可以临时规避这个限制。

6.5.2 创建有状态负载时，实例间发现服务是指什么？

云容器引擎的实例间发现服务，在原生 Kubernetes 中称之为 Headless Service。Headless Service 也是一种 Service，但是会在 YAML 中定义 spec.clusterIP: None，也就是不需要 Cluster IP 的 Service。

Headless Service 和普通 Service 的区别

- 普通 Service：

  一个 Service 可能对应多个 EndPoint（Pod），client 访问的是 Cluster IP，通过 iptables 或 IPVS 规则转到 Real Server，从而达到负载均衡的效果。例如：Service 有 2 个 EndPoint，但是 DNS 查询时只会返回 Service 的地址，具体 client 访问的是哪个 Real Server，是由 iptables 或 IPVS 规则来决定的，客户端无法自行选择访问指定的 EndPoint。

- Headless Service：

  访问 Headless Service 时，DNS 查询会如实的返回每个真实的 EndPoint（Pod 的 IP 地址）。对应到每一个 EndPoints，即每一个 Pod，都会有对应的 DNS 域名；这样 Pod 之间就可以互相访问，达到实例间发现和访问的效果。

Headless Service 使用场景

当某个工作负载的多个 Pod 之间没有任何区别时，可以使用普通 Service，利用集群 kube-proxy 实现 Service 的负载均衡，例如常见的无状态应用 Nginx。

但是某些应用场景下，工作负载的各个实例间存在不同的角色区别，比如 Redis 集群，每个 Redis 实例都是不同的，它们之间存在主从关系，并且需要相互通信。这种情况下，使用普通 Service 无法通过 Cluster IP 来保证访问到某个指定的实例，因此需要设置 Headless Service 直接访问 Pod 的真实 IP 地址，实现 Pod 间互相访问。

Headless Service 一般结合 StatefulSet 来部署有状态的应用，比如 Redis 集群、MySQL 集群等。

6.5.3 CCE 容器拉取私有镜像时报错“Auth is empty”

问题描述

在 CCE 的控制台界面中为已经创建的工作负载更换镜像，选择我上传的镜像，容器在拉取镜像时报错“Auth is empty, only accept X-Auth-Token or Authorization”。

```
Failed to pull image "IP地址:端口号/magicdoom/tidb-operator:latest": rpc error: code
= Unknown desc = Error response from daemon: Get https://IP地址:端口号
/v2/magicdoom/tidb-operator/manifests/latest: error parsing HTTP 400 response body:
json: cannot unmarshal number into Go struct field Error.code of type
errcode.ErrorCode: "{\"errors\":[{\"code\":400,\"message\":\"Auth is empty, only
accept X-Auth-Token or Authorization.\"}]}"
```

解答

您可以通过 CCE 控制台界面选择私有镜像创建应用，此时 CCE 会自动带上该 secret，升级时不会出现该问题。

您通过 API 创建应用时，在 deployment 中带入该 secret 也可以在升级时避免该问题。

```
imagePullSecrets:
- name: default-secret
```

6.5.4 为什么 Pod 调度不到某个节点上？

步骤 1 请排查节点和 docker 是否正常，排查方法请参见排查项七：内部组件是否正常。

步骤 2 如果节点和 docker 正常，而 pod 调度不到节点上，请确认 pod 是否做了亲和，排查方法请参见排查项三：检查工作负载的亲和性配置。

步骤 3 如果节点上的资源不足，导致节点调度不上，请扩容或者新增节点。

----结束

6.5.5 CCE 集群中工作负载镜像的拉取策略？

容器在启动运行前，需要镜像。镜像的存储位置可能会在本地，也可能会在远程镜像仓库中。

Kubernetes 配置文件中的 imagePullPolicy 属性是用于描述镜像的拉取策略的，如下：

- Always：总是拉取镜像。

```
imagePullPolicy: Always
```

- IfNotPresent：本地有则使用本地镜像，不拉取。

```
imagePullPolicy: IfNotPresent
```

- Never：只使用本地镜像，从不拉取，即使本地没有。

```
imagePullPolicy: Never
```

说明如下：

1. 如果设置为 Always ，则每次容器启动或者重启时，都会从远程仓库拉取镜像。
   如果省略 imagePullPolicy，策略默认为 Always。
2. 如果设置为 IfNotPreset，有下面两种情况：
   a. 当本地不存在所需的镜像时，会从远程仓库中拉取。
   b. 如果我们需要的镜像和本地镜像内容相同，只不过重新打了 tag。此 tag 镜像本地不存在，而远程仓库存在此 tag 镜像。这种情况下，Kubernetes 并不会拉取新的镜像。

6.5.6 鲲鹏集群 Docker 容器挂载点被卸载

故障现象

鲲鹏集群 Docker 容器挂载点被卸载。

问题根因

鲲鹏集群节点为 EulerOS 2.8 系统时，如果在 Docker 服务文件中配置了 MountFlags=shared 字段，会因为 systemd 特性的原因导致容器挂载点被卸载。

解决方法

修改 Docker 服务文件，删除 MountFlags=shared 字段，重启 Docker。

步骤 1 登录节点。

步骤 2 执行如下命令，删除配置文件中 MountFlags=shared 字段，然后保存。

vi /usr/lib/systemd/system/docker.service

```
[Service]
MountFlags=shared
Type=notify
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/etc/sysconfig/docker-network
Environment=GOTRACEBACK=crash
```

步骤 3 重启 Docker。

systemctl restart docker

----结束

6.5.7 下载镜像缺少层如何解决

故障现象

在使用 containerd 容器引擎场景下，拉取镜像到节点时，概率性缺少镜像层，导致工作负载容器创建失败。

问题根因

docker v1.10 之前支持 mediaType 为 application/octet-stream 的 layer，而 containerd 不支持 application/octet-stream，导致没有拉取。

解决方法

有如下两种方式可解决该问题。

- 使用高版本 Docker（>= docker v1.11）重新打包镜像。
- 手动下载镜像
  c. 登录节点。
  d. 执行如下命令手动下载镜像。
  ctr -n k8s.io images pull --user u:p images
  e. 使用新下载的镜像重新创建工作负载。

6.5.8 容器内的文件权限和用户都是问号

问题现象

节点操作系统为 CentOS 7.6 或 EulerOS 2.5 时，如果使用“Debian GNU/Linux 11 (bullseye)”内核为基础镜像的容器，会出现容器内的文件权限和用户异常。

```
[root@                ]# docker run  -it debian:11 bash
root@a6b8fa7fcdea:/# ls -al
ls: cannot access 'dev': Operation not permitted
ls: cannot access 'root': Operation not permitted
ls: cannot access 'run': Operation not permitted
ls: cannot access 'lib': Operation not permitted
ls: cannot access 'mnt': Operation not permitted
ls: cannot access '.': Operation not permitted
ls: cannot access 'tmp': Operation not permitted
ls: cannot access 'proc': Operation not permitted
ls: cannot access 'bin': Operation not permitted
ls: cannot access 'srv': Operation not permitted
ls: cannot access 'sys': Operation not permitted
ls: cannot access 'var': Operation not permitted
ls: cannot access 'etc': Operation not permitted
ls: cannot access 'media': Operation not permitted
ls: cannot access 'usr': Operation not permitted
ls: cannot access 'sbin': Operation not permitted
ls: cannot access 'home': Operation not permitted
ls: cannot access 'boot': Operation not permitted
ls: cannot access 'lib64': Operation not permitted
ls: cannot access '..': Operation not permitted
ls: cannot access 'opt': Operation not permitted
ls: cannot access '.dockerenv': Operation not permitted
total 0
d????????? ? ? ? ?            ? .
d????????? ? ? ? ?            ? ..
-????????? ? ? ? ?            ? .dockerenv
d????????? ? ? ? ?            ? bin
d????????? ? ? ? ?            ? boot
d????????? ? ? ? ?            ? dev
d????????? ? ? ? ?            ? etc
d????????? ? ? ? ?            ? home
d????????? ? ? ? ?            ? lib
d????????? ? ? ? ?            ? lib64
d????????? ? ? ? ?            ? media
d????????? ? ? ? ?            ? mnt
d????????? ? ? ? ?            ? opt
d????????? ? ? ? ?            ? proc
d????????? ? ? ? ?            ? root
d????????? ? ? ? ?            ? run
d????????? ? ? ? ?            ? sbin
```

问题影响

容器内文件权限及用户异常。

解决方案

CCE 提供以下两种解决方案，您可根据实际情况选取：

- 建议业务容器的基础镜像使用 Debian 9 或者 Debian 10。
  建议节点操作系统使用 EulerOS 2.9 或者 Ubuntu18.04。

7 网络管理

7.1 网络规划

7.1.1 集群与虚拟私有云、子网的关系是怎样的？

“虚拟私有云”类似家庭生活中路由器管理 192.168.0.0/16 的私有局域网，是为用户在云上构建的一个私有网络，是弹性云服务器、负载均衡、中间件等工作的基本网络环境。根据实际业务需要可以设置不同规模的网络，一般可为 10.0.0.0/8~24，172.16.0.0/12~24，192.168.0.0/16~24，其中最大的网络 10.0.0.0/8 的 A 类地址网络。

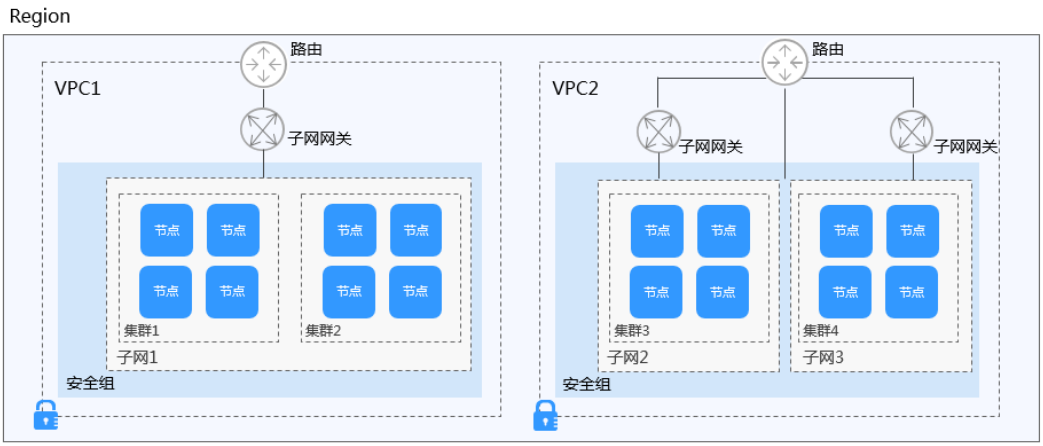
子网是虚拟私有云中的一个子集，可以将虚拟私有云划分为一个个子网，每个子网之间可以通过安全组控制其之间能否互通，保证子网之间可以相互隔离，用户可以将不同业务部署在不同的子网内。

集群是同一个 VPC 中一个或多个弹性云服务器（又称：节点）通过相关技术组合而成的计算机群体，为容器运行提供了计算资源池。

如图 7-1，同一个 region 下可以有多个虚拟私有云（图中以 VPC 表示）。虚拟私有云由一个个子网组成，子网与子网之间的网络交互通过子网网关完成，而集群就是建立在某个子网中。因此，存在以下三种场景：

- 不同集群可以创建在不同的虚拟私有云中。
- 不同集群可以创建在同一个子网中。
- 不同集群可以创建在不同的子网中。

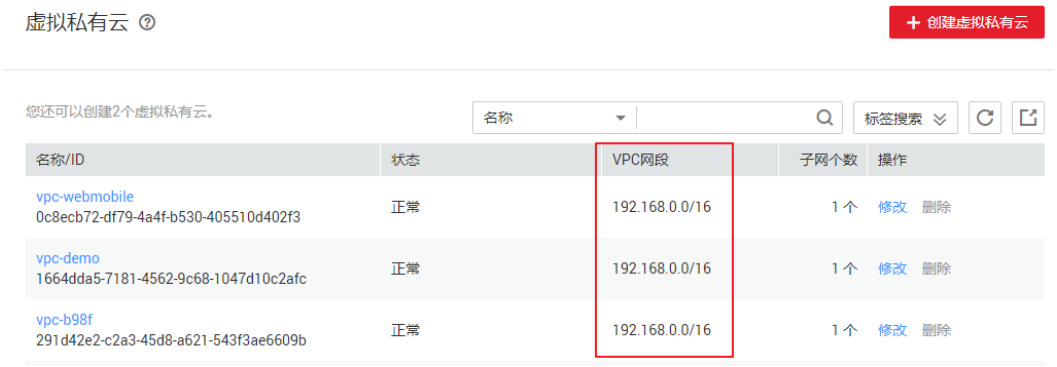
图7-1 集群与 VPC、Subnet 的关系



7.1.2 如何查看虚拟私有云 VPC 的网段？

在“虚拟私有云”页面，可查看虚拟私有云的“名称/ID”和“VPC 网段”。用户可以调整已创建的 VPC 或通过重新创建 VPC 调整网段。

图7-2 查看 VPC 网段



7.1.3 如何设置 CCE 集群中的 VPC 网段和子网网段？

VPC 中的子网网段一旦创建，便无法更改。创建虚拟私有云时，请预留一定的 VPC 网段和子网网段资源，避免后续无法扩容。

子网网段可在“创建虚拟私有云”页面的“子网配置 > 子网网段”中进行设置。在设置选项下可查看到“可用 IP 数”。

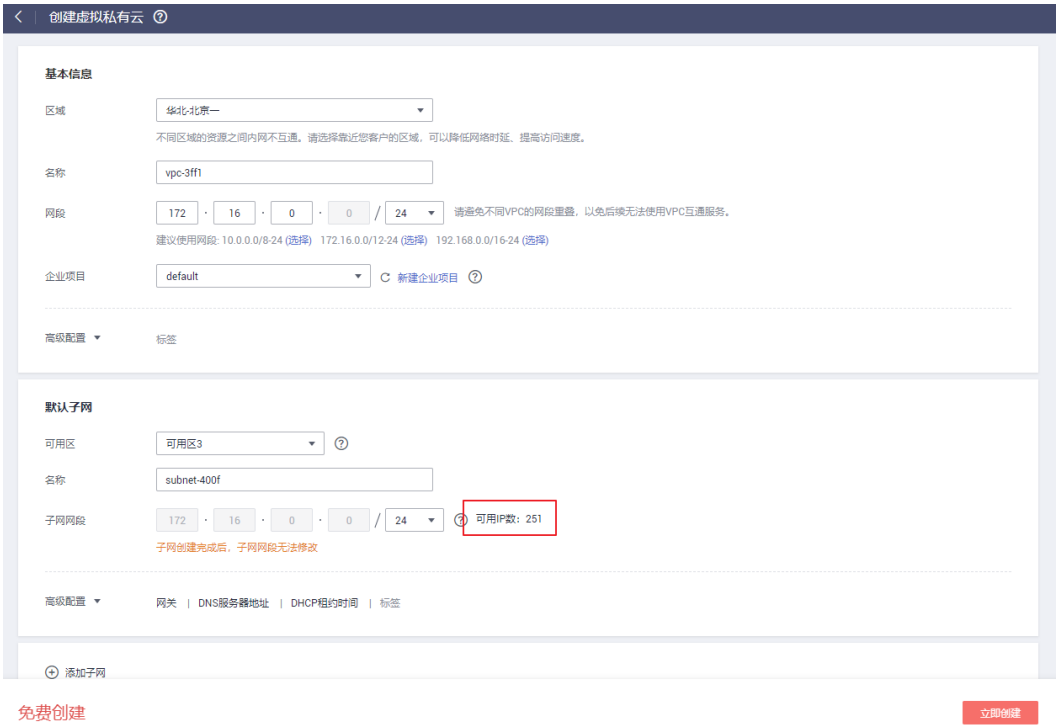
容器网段掩码如果设置不合适，会导致集群实际可用的节点较少。

例如：

- 节点规模为 1000，子网可以选择 192.168.0.0/20，支持约 4090 个节点。

- VPC 选择 192.168.0.0/16，子网选择 192.168.0.0/25，则可支持 122 个节点。若创建 200 个节点集群，则实际只能添加 122 个节点（包括控制节点）。

图7-3 查看“可用 IP 数”



7.1.4 如何设置 CCE 集群中的容器网段？

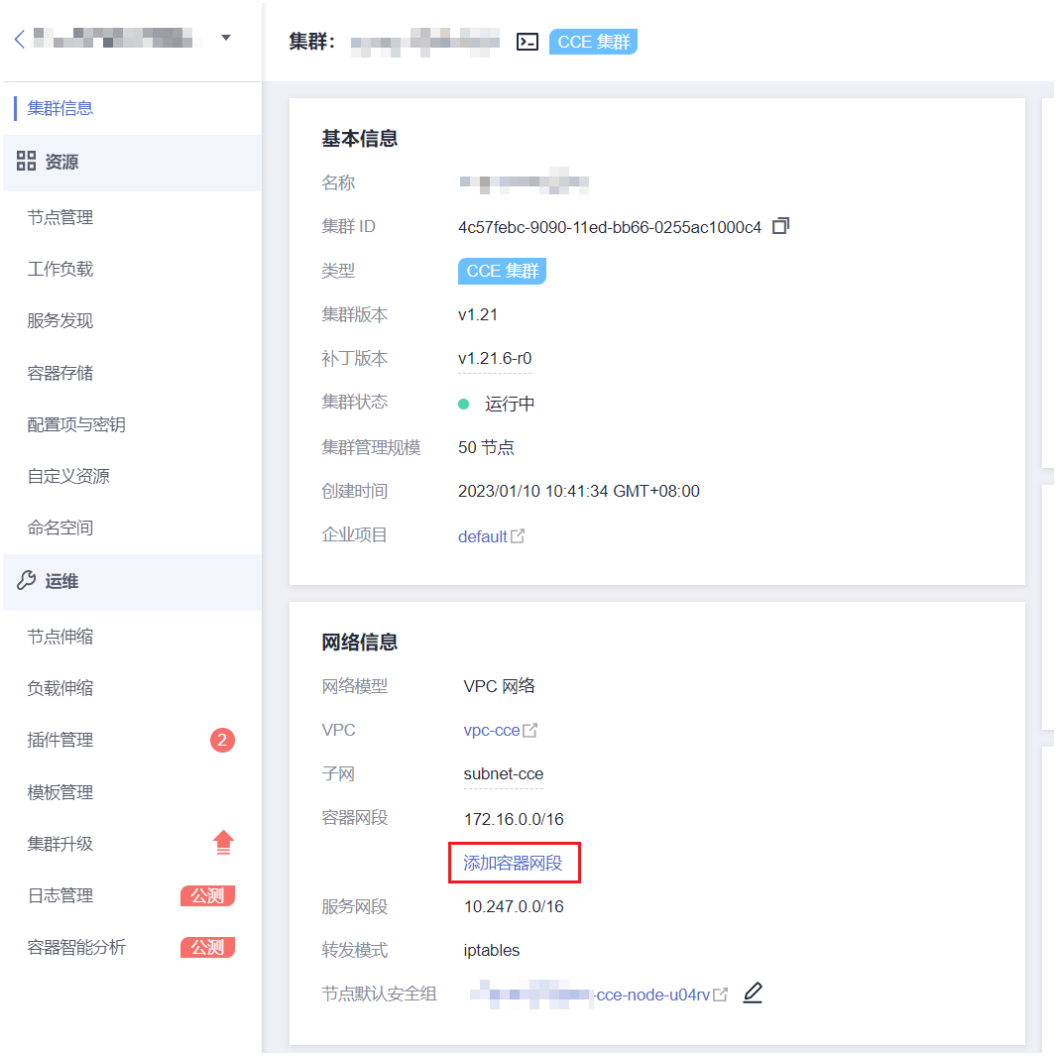
进入 CCE 控制台，在创建集群时进行“容器网段”设置。

当前可供选择的容器网段为 10.0.0.0/8~18，172.16.0.0/16~18，192.168.0.0/16~18。

集群创建完成后，如需添加容器网段，可前往集群信息页面，单击“添加容器网段”进行添加。

须知

- 容器网段添加后无法删除，请谨慎操作。
- 服务网段默认为 10.247.0.0/16，容器网段不能选择此网段。

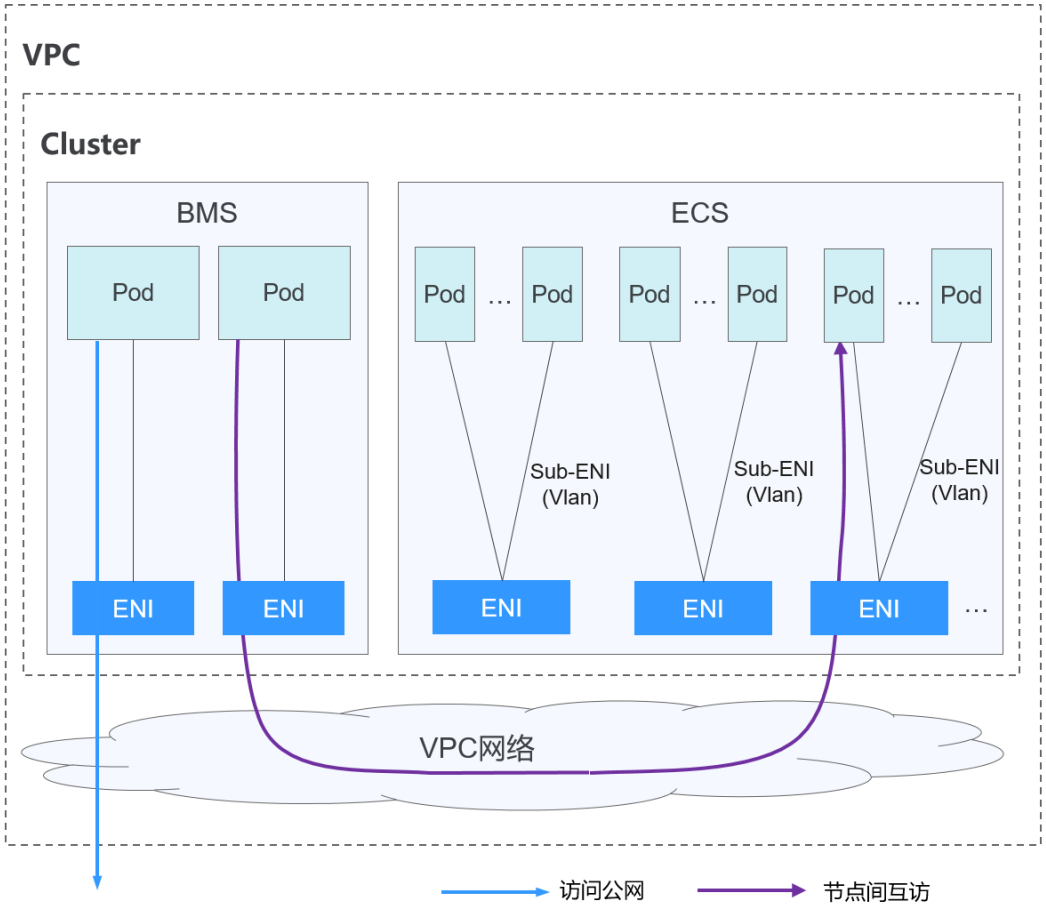


7.1.5 什么是云原生网络 2.0 网络模式，适用于什么场景？

云原生网络 2.0 是什么

云原生网络 2.0 是新一代容器网络模型，深度整合了虚拟私有云 VPC 的原生弹性网卡（Elastic Network Interface，简称 ENI）能力，采用 VPC 网段分配容器地址，支持 ELB 直通容器，享有高性能。

图7-4 云原生网络 2.0



约束与限制

仅 CCE Turbo 集群支持使用云原生网络 2.0。

适用场景

- 性能要求高，需要使用 VPC 其他网络能力的场景：由于云原生网络 2.0 直接使用的 VPC 网络，性能与 VPC 网络的性能几乎一致，所以适用于对带宽、时延要求极高的业务场景，比如：线上直播、电商秒杀等。
- 大规模组网：云原生网络 2.0 当前最大可支持 2000 个 ECS 节点，10 万个容器。

7.1.6 什么是弹性网卡？

弹性网卡即虚拟网卡，您可以通过创建并配置弹性网卡，并将其附加到您的云服务器实例（包括弹性云服务器和裸金属服务器）上，实现灵活、高可用的网络方案配置。

弹性网卡类型

- 主弹性网卡：在创建云服务器实例时，随实例默认创建的弹性网卡称作主弹性网卡。主弹性网卡无法与实例进行解绑。

- 扩展弹性网卡：您可以创建扩展弹性网卡，将其附加到云服务器实例上，您也可以将其从实例上进行解绑。每台实例可附加的扩展弹性网卡数量由实例规格决定。

应用场景

- 灵活迁移

  通过将弹性网卡从云服务器实例解绑后再绑定到另外一台服务器实例，保留已绑定私网 IP、弹性公网 IP 和安全组策略，无需重新配置关联关系，将故障实例上的业务流量快速迁移到备用实例，实现服务快速恢复。
- 业务分离管理

  可以为服务器实例配置多个分属于同一 VPC 内不同子网的弹性网卡，特定网卡分别承载云服务器实例的内网、外网、管理网流量。针对子网可独立设置访问安全控制策略与路由策略，弹性网卡也可配置独立安全组策略，从而实现网络隔离与业务流量分离。

使用限制

- 云服务器实例与扩展弹性网卡必须在同一 VPC，可以分属于不同安全组。
- 主弹性网卡不能解绑服务器。
- 云服务器可附加的扩展弹性网卡数量由云服务器实例规格决定。

7.1.7 集群安全组规则配置

CCE 作为通用的容器平台，安全组规则的设置适用于通用场景。集群在创建时将会自动为 Master 节点和 Node 节点分别创建一个安全组，其中 Master 节点的安全组名称是：**{集群名}-cce-control-{随机 ID}**；Node 节点的安全组名称是：**{集群名}-cce-node-{随机 ID}**。使用 CCE Turbo 集群时会额外创建一个 ENI 的安全组，名为**{集群名}-cce-eni-{随机 ID}**。

用户可根据安全需求，登录 CCE 控制台，单击服务列表中的“网络 > 虚拟私有云 VPC”，在网络控制台单击“访问控制 > 安全组”，找到 CCE 集群对应的安全组规则进行修改和加固。

如集群在创建时需要指定节点安全组，请参考集群自动创建的 Node 节点安全组规则放通指定端口，以保证集群中的正常网络通信。

须知

安全组规则的修改和删除可能会影响集群的正常运行，请谨慎操作。如需修改安全组规则，请尽量避免对 CCE 运行依赖的端口规则进行修改。

Node 节点安全组规则

入方向

集群自动创建 Node 节点的安全组名称为**{集群名}-cce-node-{随机 ID}**，默认入方向规则如下图所示，**源地址属于本安全组的需全部放通**，其余端口说明见下表。

图7-5 VPC 网络模型 Node 节点默认安全组

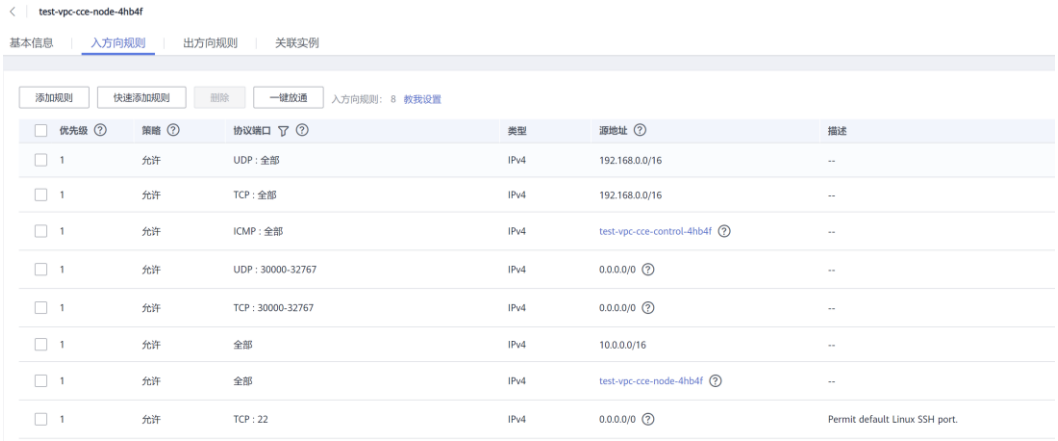


表7-1 VPC 网络模型 Node 节点安全组默认端口说明

端口	默认源地址	说明	是否可修改	修改建议
UDP：全部 TCP：全部	VPC 网段	Node 节点之间互访、Node 节点与 Master 节点互访。	不建议修改	不涉及
ICMP：全部	Master 节点网段	Master 节点访问 Node 节点。	不建议修改	不涉及
TCP：30000-32767 UDP：30000-32767	所有 IP 地址	集群 NodePort 服务默认访问端口范围。	可修改	端口需对 VPC 网段、容器网段和 ELB 的网段放通。
全部	容器网段	节点与容器互访。	不建议修改	不涉及
全部	Node 节点网段	Node 节点之间互访。	不建议修改	不涉及
TCP：22	所有 IP 地址	允许 SSH 远程连接 Linux 弹性云服务器。	**建议修改**	不涉及

图7-6 容器隧道网络模型 Node 节点默认安全组

表7-2 容器隧道网络模型 Node 节点安全组默认端口说明

端口	默认源地址	说明	是否可修改	修改建议
UDP：4789	所有 IP 地址	容器间网络互访。	不建议修改	不涉及
TCP：10250	master 节点网段	master 的主动访问 node 的 kubelet（如执行 kubectl exec {pod}）。	不建议修改	不涉及
TCP：30000-32767 UDP：30000-32767	所有 IP 地址	集群 NodePort 服务默认访问端口范围。	可修改	端口需对 VPC 网段、容器网段和 ELB 的网段放通。
TCP：22	所有 IP 地址	允许 SSH 远程连接 Linux 弹性云服务器。	**建议修改**	不涉及
全部	本安全组和 VPC 网段	源地址属于本安全组和 VPC 网段的需全部放通。	不建议修改	不涉及

出方向

对于**出方向规则**，CCE 创建的安全组默认全部放通，通常情况下不建议修改。如需加固出方向规则，请注意如下端口需要放通。

表7-3 Node 节点安全组出方向规则最小范围

端口	放通地址段	说明

端口	放通地址段	说明
UDP：53	子网的 dns 服务器	用于域名解析。
UDP：4789（仅容器隧道网络模型的集群需要）	所有 IP 地址	容器间网络互访。
TCP：5443	Master 节点网段	master 的 kube-apiserver 的监听端口。
TCP：5444	VPC 网段、容器网段	kube-apiserver 服务端口，提供 k8s 资源的生命周期管理。
TCP：6443	Master 节点网段	-
TCP：8445	VPC 网段	Node 节点存储插件访问 Master 节点。
TCP：9443	VPC 网段	Node 节点网络插件访问 Master 节点。

Master 节点安全组规则

Master 节点的安全组名称是为**{集群名}-cce-control-{随机 ID}**，默认入方向规则如下图所示，**源地址属于本安全组的需全部放通**，其余端口说明见下表。

图7-7 Master 节点安全组规则

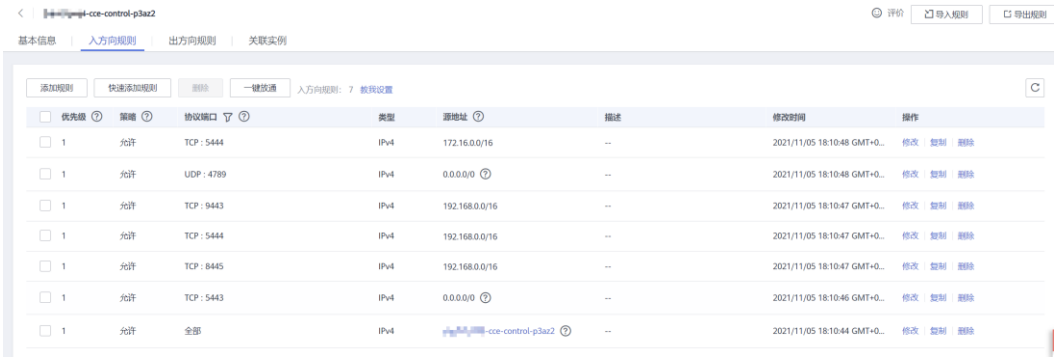


表7-4 Master 节点安全组默认端口说明

端口	默认源地址	说明	是否支持修改	修改建议
TCP：5444	VPC 网段、容器网段	kube-apiserver 服务端口，提供 k8s 资源的生命周期管理。	不建议修改	不涉及

端口	默认源地址	说明	是否支持修改	修改建议
UDP：4789（仅容器隧道网络模型的集群需要）	所有 IP 地址	容器间网络互访。	不建议修改	不涉及
TCP：9443	VPC 网段	Node 节点网络插件访问 Master 节点。	不建议修改	不涉及
TCP：5443	所有 IP 地址	master 的 kube-apiserver 的监听端口。	可修改	端口需保留对 VPC 网段、容器网段和托管网格控制面网段放通。
TCP：8445	VPC 网段	Node 节点存储插件访问 Master 节点。	不建议修改	不涉及
全部	本安全组和 VPC 网段	源地址属于本安全组和 VPC 网段的需全部放通。	不建议修改	不涉及

说明

5443 端口默认对所有网段放通，如果您需要对安全组做加固，请保留 5443 端口对 198.19.128.0/20、198.19.160.0/20、198.19.176.0/20 网段放通，以使用 CloudShell 功能。因为 CloudShell 基于 VPCEP 实现，当 5443 端口未放通 198.19.128.0/20 网段时将无法访问集群。

7.1.8 如何设置 IPv6 服务网段

问题背景

当您需要创建一个 IPv4/IPv6 双栈的 CCE Turbo 集群时，需要设置 IPv6 服务网段，该网段默认值为 fc00::/112，包含了 65536 个 IPv6 服务地址。如果您需要自定义服务网段，您可参考本文进行设置。

IPv6 介绍

IPv6 地址

IPv6 地址采用 128 位二进制表示，是 IPv4 地址长度的 4 倍。因此 IPv4 地址的十进制格式不再适用，IPv6 采用了十六进制来表示，以 16 位为一组，每组以冒号“:”隔开，可以分为 8 组，每组以 4 位十六进制方式（不区分大小写）表示，共计 32 个十六进制数。

IPv6 地址存在多种省略写法：

- 0 位省略：如果每个冒号分组中存在以 0 开头的，则可以将 0 位省略，多个 0 连续时可省略多个。例如以下 IPv6 地址均是等价的。

- ff01:0d28:03ee:0000:0000:0000:0000:0c23
- ff01:d28:3ee:0000:0000:0000:0000:c23
- ff01:d28:3ee:0:0:0:0:c23

- 双冒号省略：如果以十六进制表示的 IPv6 地址中间依然存在很多个全为 0 的分组，可以把连续全为 0 的分组压缩成双冒号"::"。但为保证唯一性，这种压缩方式只能使用一次，即一个 IPv6 地址中只能出现一次双冒号"::"。

例如：

双冒号省略前	双冒号省略前
ff01:d28:3ee:0:0:0:0:c23	ff01:d28:3ee::c23
0:0:0:0:0:0:0:1	::1
0:0:0:0:0:0:0:0	::

IPv6 地址段

IPv6 地址段通常采用 CIDR（无类别域间路由选择）表示法，通常用斜杠（/）后跟一个数字表示，即格式为“IPv6 地址/前缀长度”。此处前缀长度与 IPv4 地址段的掩码作用类似，用数字来表示网络部分所占用的二进制位数，可将 IPv6 地址分为网络地址和主机地址两部分。而前缀长度指定了网络部分占用的位数，剩余位数则是主机地址部分，可以更加方便和灵活地表示不同的地址段。

例如，fc00:d28::/32 表示一个前缀长度为 32 位的 IPv6 地址段，则在该网段中分配地址时，前 32 位（以二进制计算，此处即为 fc00:d28）为网络地址，后 96 位则为可用的主机地址。

IPv6 服务网段使用约束

在设置集群服务网段时，需要首先考虑以下有如下使用约束：

- IPv6 服务网段必须属于 fc00::/8 网段内。

  该地址属于本地唯一地址（ULA）网段。ULA 拥有固定前缀：fc00::/7，其中包括 fc00::/8 和 fd00::/8 两个范围，类似于 IPv4 的专用网络地址 10.0.0.0/8、172.16.0.0/12 和 192.168.0.0/16，相当于私有 IP 网段，仅能够在本地网络使用。
- 前缀范围 112-120，您可以通过调整前缀数值，调整地址个数，地址数最多有 65536 个。

IPv6 服务网段示例

根据约束，本文中提供一个包含 8192 个地址的 IPv6 网段设置示例，供您参考。

1. 根据地址数需求设定前缀长度，且前缀范围为 112-120。

   本例中，需要 8192 个地址数，8192 个地址需要 13 位二进制数表示，因此在 IPv6 中前缀长度为 128-13=115。

2. 设置 IPv6 网络地址，且网络地址必须属于 fc00::/8 网段内。

本例中，确定前缀长度为 115，由于网络地址必须属于 fc00::/8 网段内，因此前 8 位二进制数是固定的。可修改的网络地址范围是第 9 位至第 115 位，第 116 位至第 128 位则属于主机地址。

将 IPv6 地址写成二进制形式，则根据以上约束，前缀长度为 115 时以下加粗部分是不可修改的。

二进制： **1111 1100** **** **** ... ***0 0000 0000 0000**/115

| | | | | | | |

十六进制： f c x x ... y 0 0 0/115

其中 x 为任意十六进制数，y 可选范围为 0、2、4、6、8、a、c、e。

Node 节点安全组规则

入方向

集群自动创建 Node 节点的安全组名称为**{集群名}-cce-node-{随机 ID}**，默认入方向规

7.2 网络异常

7.2.1 工作负载网络异常时，如何定位排查？

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频率原因往低频率原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

- 排查项一：容器+容器端口
- 排查项二：节点 IP+节点端口
- 排查项三：负载均衡 IP+端口
- 排查项四：NAT 网关+端口
- 排查项五：检查容器所在节点安全组是否放通

排查项一：容器+容器端口

在 CCE 控制台界面或者使用 kubectl 命令查找 pod 的 IP，然后登录到集群内的节点或容器中，使用 curl 命令等方法手动调用接口，查看结果是否符合预期。

如果容器 IP+端口不能访问，建议登录到业务容器内使用“127.0.0.1+端口”进行排查。

常见问题：

1. 容器端口配置错误（容器内未监听访问端口）。
2. URL 不存在（容器内无相关路径）。
3. 服务异常（容器内的业务 BUG）。

排查项二：节点 IP+节点端口

只有发布为节点访问（NodePort）或负载均衡（LoadBalancer ）的服务才能通过节点 IP+节点端口进行访问。

- **节点访问（NodePort）类型：**

  节点的访问端口就是节点对外发布的端口。

- **负载均衡（LoadBalancer ）类型：**

  负载均衡的节点端口通过“编辑 YAML”可以查看。

如下图所示：

nodePort: 30637 为节点对外暴露的端口。**targetPort: 80** 为 Pod 对外暴露的端口。**port: 123** 为服务对外暴露的端口，负载均衡类型的服务同时使用该端口配置 ELB 的监听器。

```
spec:
  ports:
    - name: cce-service-0
      protocol: TCP
      port: 123
      targetPort: 80
      nodePort: 30637
```

找到节点端口（nodePort）后，使用容器所在节点的 IP 地址+端口进行访问，并查看结果是否符合预期。

常见问题：

1. 节点的入方向对业务端口未放通。
2. 节点配置了自定义路由，并且配置错误。
3. pod 的 label 与 service 的 label 不匹配（kubectl 或 API 创建）。

排查项三：负载均衡 IP+端口

如果使用负载均衡 IP+端口不能访问，但节点 IP+端口可以访问。

请排查：

- 相关端口或 URL 的后端服务器组是否符合预期。
- 节点上的安全组是否对 ELB 暴露了相关的协议或端口。
- 四层 ELB 的健康检查是否开启（未开启的话，请开启）。
- 七层 ELB 的访问方式中使用的证书是否过期。

常见问题：

1. 发布四层 ELB 时，如果客户在界面未开启健康检查，ELB 可能会将流量转发到异常的节点。
2. UDP 协议的访问，需要放通节点的 ICMP 协议。
3. pod 的 label 与 service 的 label 不匹配（kubectl 或 API 创建）。

排查项四：NAT 网关+端口

配置在 NAT 后端的服务器，通常不配置 EIP，不然可能会出现网络丢包等异常。

排查项五：检查容器所在节点安全组是否放通

用户可单击服务列表中的“网络 > 虚拟私有云 VPC”，在网络控制台单击“访问控制 > 安全组”，找到 CCE 集群对应的安全组规则进行修改和加固。

- CCE 集群：

  Node 节点的安全组名称是：**{集群名}-cce-node-{随机字符}**。

请排查：

- 从集群外访问集群内负载时，来访者的 IP 地址、端口、协议需在集群安全组的入方向规则中开放。
- 集群内的工作负载访问外部时，访问的地址、端口、协议需在集群安全组的出方向规则中开放。

更多安全组配置信息请参见集群安全组规则配置。

7.2.2 集群内部无法使用 ELB 地址访问负载

问题现象

在集群内部（节点上或容器中），使用 ELB 地址无法访问。

问题原因

当 LoadBalancer Service 设置了服务亲和为节点级别，即 externalTrafficPolicy 取值为 Local 时，在使用中可能会碰到从集群内部（节点上或容器中）使用 ELB 地址访问不通的情况，回显类似如下内容：

```
upstream connect error or disconnect/reset before headers. reset reason: connection failure
```

这是由于 Kubernetes 在创建 LoadBalancer Service 时，kube-proxy 会把 ELB 的访问地址作为 External-IP 添加到 iptables 或 IPVS 中。如果客户端从集群内部发起访问 ELB 地址的请求，该地址会被认为是服务的 External-IP，被 kube-proxy 直接转发，而不再经过集群外部的 ELB。

当 externalTrafficPolicy 的取值为 Local 时，在不同容器网络模型和服务转发模式下，情况会有所不同，详情如下：

Server	Client	容器隧道集群	VPC 集群（IPVS）	容器隧道集群	VPC 集群（iptables）

		（IPVS）		（iptables）	
节点访问类型 Service	同节点	OK，Pod 容器所在节点通，其他不通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通
	跨节点	OK，Pod 容器所在节点通，其他不通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通，访问本节点 IP+访问端口通；其他不通	OK，访问 Pod 容器所在节点通，访问本节点 IP+访问端口通；其他不通
	同节点容器	OK，Pod 容器所在节点通，其他不通	OK，访问 Pod 容器所在节点不通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点不通
	跨节点容器	OK，Pod 容器所在节点通，其他不通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通	OK，访问 Pod 容器所在节点通
独享型负载均衡类型 Service	同节点	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通
	同节点容器	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通
nginx-ingress 插件的 service 为 Local 级别独享型	同节点	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通
	同节点容器	公网通，私网不通	公网通，私网不通	公网通，私网不通	公网通，私网不通

解决办法

解决这个问题通常有如下办法：

- （**推荐**）在集群内部访问使用 Service 的 ClusterIP 或服务域名访问。
- 将 Service 的 externalTrafficPolicy 设置为 Cluster，即集群级别服务亲和。不过需要注意这会影响源地址保持。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-
```

```
bandwidth","bandwidth_chargemode":"traffic","bandwidth_size":5,"bandwidth_share
type":"PER","eip_type":"5_bgp","name":"james"}'
  labels:
    app: nginx
  name: nginx
spec:
  externalTrafficPolicy: Cluster
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

- 使用 Service 的 pass-through 特性，使用 ELB 地址访问时绕过 kube-proxy，先访问 ELB，进过 ELB 再访问到负载。

说明

- 独享型负载均衡配置 pass-through 后，在工作负载同节点和同节点容器内无法通过 Service 访问。
- 1.15 及以下老版本集群暂不支持该能力。
- IPVS 网络模式下，对接同一个 ELB 的 Service 需保持 pass-through 设置情况一致。

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    kubernetes.io/elb.pass-through: "true"
    kubernetes.io/elb.class: union
    kubernetes.io/elb.autocreate: '{"type":"public","bandwidth_name":"cce-
bandwidth","bandwidth_chargemode":"traffic","bandwidth_size":5,"bandwidth_share
type":"PER","eip_type":"5_bgp","name":"james"}'
  labels:
    app: nginx
  name: nginx
spec:
  externalTrafficPolicy: Local
  ports:
  - name: service0
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  type: LoadBalancer
```

7.2.3 集群外部访问 Ingress 异常

Ingress 基于七层的 HTTP 和 HTTPS 协议进行转发，是集群流量的入口，可以通过域名和路径对访问做到更细粒度的划分。集群在添加 Ingress 后，可能会出现无法正常访问的情况，本文提供添加 Ingress 失败或无法正常访问的通用排查思路，帮助您找到问题所在。在进行 Ingress 问题排查前，请您阅读以下几点须知，并进行自检。

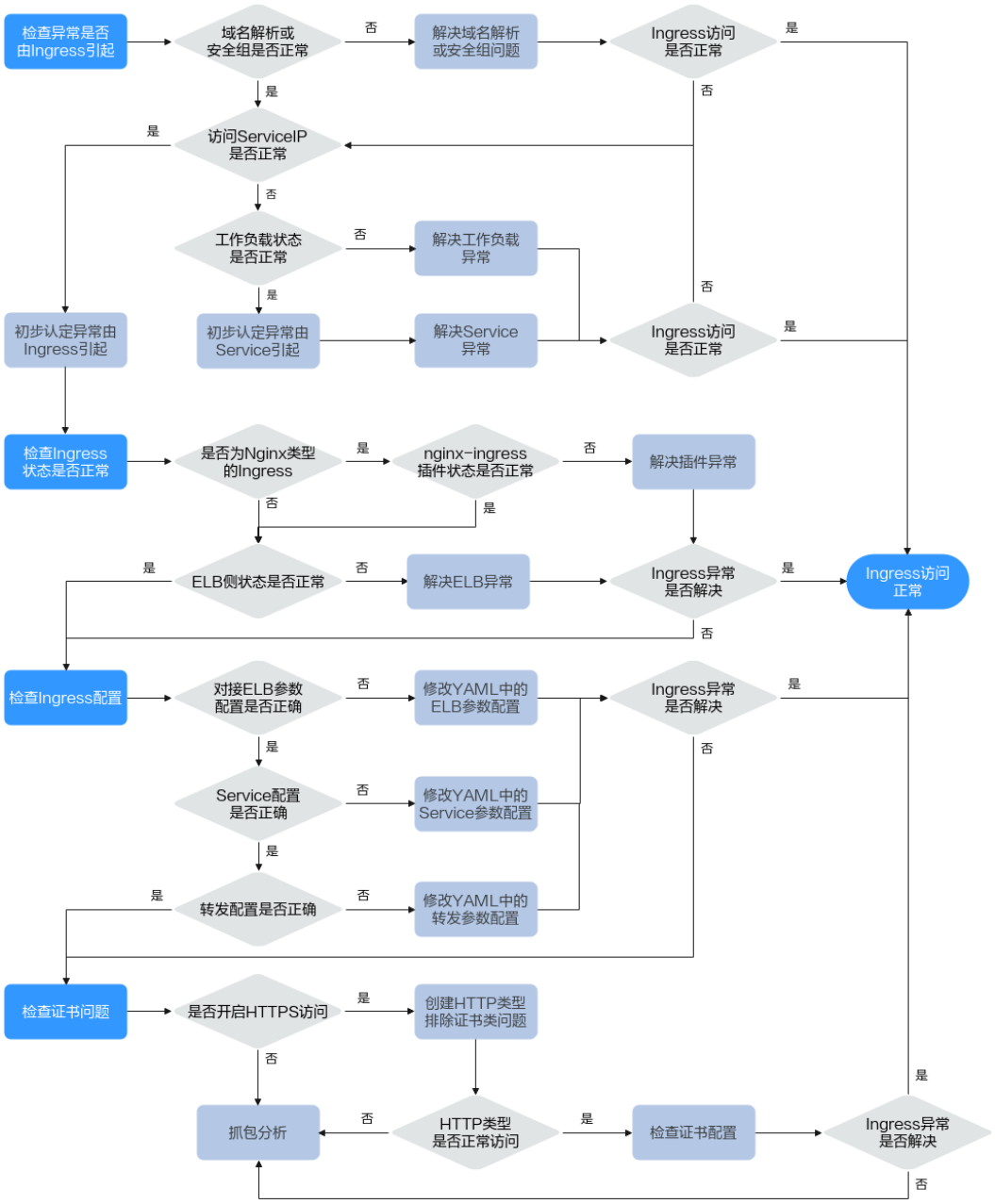
须知

- 如创建 Ingress 过程中指定了 host 地址，将无法通过 IP 访问。
- 请检查集群的节点安全组，确认 30000-32767 范围内的业务端口在入方向对所有网段放开。

排查思路

本文提供了 Ingress 外部访问异常排查全景图，如下图所示，帮助您由浅入深地排查 Ingress 访问异常问题。

图7-8 Ingress 访问异常排查全景图



1. **检查异常是否由 Ingress 引起**。

   首先需要确认问题是否由 Ingress 导致的，因此需要确保外部域名解析正常、安全组规则正确，且 Ingress 对应的 Service 和工作负载工作正常。

2. **检查 Ingress 状态是否正常**。

   在 Service 和工作负载都正常的情况下，需要保证 Ingress 依赖的 ELB 状态正常。如果是 Nginx 型的 Ingress，还需要保证 nginx-ingress 插件的状态是正常的。

3. **检查 Ingress 配置是否正确**。

   如果以上排查结果都正常，说明可能是 Ingress 的配置出现问题。

- 检查对接 ELB 参数是否填写正确。
- 检查 Service 参数是否填写正确。
- 检查转发配置的参数是否填写正确。

4. **检查证书问题**。

   如果 Ingress 开启了 HTTPS 访问，还需要排除证书配置错误的问题。您可使用相同 ELB 创建一个 HTTP 协议的 Ingress 访问，如 HTTP 协议下访问正常，则说明 HTTPS 协议证书可能存在问题。

5. 如果以上排查均无效果，请进行抓包分析，或提交工单寻求帮助。

检查异常是否由 Ingress 引起

您需要确认访问异常是否由 Ingress 引起，当存在域名解析异常、安全组规则错误、Service 异常或工作负载本身异常时，都可能会引起 Ingress 访问不通。

以下排查顺序遵从由外到内的规则：

步骤 1 检查域名解析或安全组规则是否正常，排除集群外部可能存在的问题。

1. 执行以下命令检查域名在权威 DNS 的解析是否生效。

   ```
   nslookup -qt=类型 域名 权威 DNS 地址
   ```

2. 检查集群节点安全组规则，确认 30000-32767 范围内的业务端口在入方向对所有网段放开。如需对安全组进行加固，详情请参见集群安全组规则配置。

优先级	策略	协议端口	类型	源地址
1	允许	UDP : 30000-32767	IPv4	0.0.0.0/0
1	允许	TCP : 30000-32767	IPv4	0.0.0.0/0

步骤 2 检查 Service 是否可以正常访问容器内业务，检查集群内部可能存在的问题。

您可通过在集群中新建 Pod 并通过 ClusterIP 访问 Service 的方式进行检查，如您的服务为 NodePort 类型，也可通过 EIP:Port 使用互联网访问服务来验证。

1. 通过 kubectl 连接集群，查询集群内服务。

   ```
   # kubectl get svc
   NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
   kubernetes   ClusterIP   10.247.0.1       <none>        443/TCP        34m
   nginx        ClusterIP   10.247.138.227   <none>        80/TCP         30m
   ```

2. 创建一个 Pod 并登录到容器内。

   ```
   kubectl run -i --tty --image nginx:alpine test --rm /bin/sh
   ```

3. 使用 curl 命令访问 Service 的 ClusterIP:Port，检验集群内服务是否可访问。

   ```
   curl 10.247.138.227:80
   ```

   如 Service 可正常访问，则说明后端工作负载状态正常，初步可认定异常由 Ingress 引起，请参照检查 Ingress 状态是否正常继续排查。

   如 Service 访问异常，则需继续排查工作负载状态，确定异常原因。

步骤 3 检查工作负载状态是否正常。

如工作负载正常，但 Service 无法正常访问，则说明异常可能是由于 Service 引起。请检查 Service 的配置，例如容器端口是否正确填写为容器内业务开放端口。

如工作负载正常，但访问结果不符合预期，请排查容器内运行的业务代码。

----结束

检查 Ingress 状态是否正常

CCE 支持两种类型的 Ingress，其中 Nginx 类型的 Ingress Controller 由社区开源的插件提供，需要在集群中安装插件自行运维；而 ELB 型的 Ingress Controller 运行在 master 节点上。

步骤 1 如果您使用 Nginx 类型的 Ingress，需要在集群中安装 nginx-ingress 插件。如果您使用 ELB 型的 Ingress，则无需检查此步骤。

前往“插件管理 > 已安装插件”查看 nginx-ingress 插件的状态为“运行中”。请保证集群下节点资源充足，若资源不足，插件实例将无法调度。

步骤 2 前往 ELB 控制台检查 ELB 的状态。

- ELB 型 Ingress

  访问端口可自定义，请检查 ELB 侧创建的监听器和后端服务器组未被删除或修改。

  建议您在创建 ELB 型 Ingress 时通过控制台选择自动创建 ELB，并且不要对自动创建的 ELB 进行修改，能够有效避免 ELB 侧导致的 Ingress 异常。

- Nginx 型 Ingress

  访问端口固定为 80 和 443，不支持自定义端口。安装 nginx-ingress 插件将同时占用 80 和 443 端口，切勿删除，否则只能通过重装插件解决。

您也可以通过错误码来简单判断故障是否由于 ELB 引起，如下图页面所示，说明该异常大概率由 ELB 侧故障引起，需要重点排查。

----结束

检查 Ingress 配置是否正确

如果以上排查项都正常，需要考虑是否由于参数设置引起异常。由于使用 kubectl 创建时需要填写的参数较多易出错，建议您使用控制台创建，根据可视界面按需设置参数，自动过滤不符合要求的负载均衡及 Service，能够有效避免出现关键参数格式错误或缺失的问题。

请您根据以下思路进行逐一排查 Ingress 配置：

- **检查对接 ELB 参数是否正确**

由于 ELB 通过 annotations 字段下的参数进行定义，但是 K8s 在创建资源时并不会对 annotations 字段参数进行校验，如果出现关键参数错误或缺失，Ingress 资源也可被创建，但无法正常访问。

以下是出现频率较高的问题，供您参考：

- 对接的目标 ELB 未与集群处于**同一 VPC** 下。
- 添加 ELB 型 Ingress 时对接已有 ELB，annotations 中关键字段 **kubernetes.io/elb.id**、**kubernetes.io/elb.ip**、**kubernetes.io/ingress.class**、**kubernetes.io/elb.port** 缺失。
- 添加 Nginx 型 Ingress 时，**未安装 nginx-ingress 插件**导致无 ELB 连接。
- 添加 Nginx 型 Ingress 时，annotations 中关键字段 **kubernetes.io/ingress.class**、**kubernetes.io/elb.port** 缺失。
- 添加 Nginx 型 Ingress 时，**kubernetes.io/elb.port** 参数不支持自定义端口，使用 HTTP 协议固定为 80，HTTPS 协议固定为 443。

- **检查 Service 配置是否正确**
  - 检查 Ingress 对接的 Service 类型是否正确，Ingress 支持的 Service 请参见下表。

表7-5 Ingress 支持的 Service 类型

Ingress 类型	访问类型	集群内访问（ClusterIP）	节点访问（NodePort）
ELB 型 Ingress	负载均衡路由	不支持	支持
	ENI 负载均衡路由	支持	不支持
Nginx 型 Ingress	负载均衡路由	支持	支持
	ENI 负载均衡路由	支持	不支持

  - 检查 Service 的访问端口号是否正确，此处 Service 的访问端口号（port 字段）需区别于容器端口号（targetPort 字段）。
- **检查转发配置的参数是否填写正确**
  - 添加的 URL 转发路径要求后端应用内存在相同的路径，否则转发无法生效。

    例如，Nginx 应用默认的 Web 访问路径为“/usr/share/nginx/html”，在为 Ingress 转发策略添加“/test”路径时，需要应用的 Web 访问路径下也包含相同路径，即“/usr/share/nginx/html/test”，否则将返回 404。

说明

使用 Nginx 型的 Ingress Controller 时，您可通过在 annotations 字段添加 rewrite 注释进行重定向，将业务内不存在的 path 路径进行重写，避免访问路径不存在的错误，详情请参见 Rewrite。

  - 创建 Ingress 时指定了域名（host），将无法通过 IP 访问。

检查证书问题

CCE 的 Ingress 密钥证书类型为 IngressTLS 或 kubernetes.io/tls，若证书类型不正确，创建的 Ingress 将无法在 ELB 侧建立监听器，导致 Ingress 访问异常。

步骤 1 首先去除 YAML 中关于 HTTPS 的参数，尝试创建 HTTP 类型的 Ingress 是否可正常访问。

如 HTTP 访问正常，则考虑 HTTPS 密钥证书是否存在问题。

步骤 2 排除密钥类型错误。检查密钥类型是否为 **IngressTLS 或 kubernetes.io/tls 类型**。

```
# kubectl get secret
NAME                TYPE                                DATA   AGE
ingress             IngressTLS                          2      36m
```

步骤 3 创建测试证书，排除证书问题。

```
# openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key -out tls.crt
-subj "/CN={YOUR_HOST}/O={YOUR_HOST}"
```

步骤 4 使用测试的私钥证书 tls.key 和 tls.crt 创建正确类型的密钥，并重新创建 HTTPS 类型的 Ingress 测试是否可以正常访问。本文以创建 IngressTLS 类型密钥为例。

通过 kubectl 方式创建时，创建 IngressTLS 类型密钥的示例如下：

```
kind: Secret
apiVersion: v1
type: IngressTLS
metadata:
  name: ingress
  namespace: default
data:
  tls.crt: LS0tLS1CRU*****FURS0tLS0t
  tls.key: LS0tLS1CRU*****VZLS0tLS0=
```

说明

此处 tls.crt 和 tls.key 为示例，请获取真实密钥进行替换。tls.crt 和 tls.key 的值为 Base64 加密后的内容。

----结束

7.2.4 CCE 中域名解析失败

问题现象

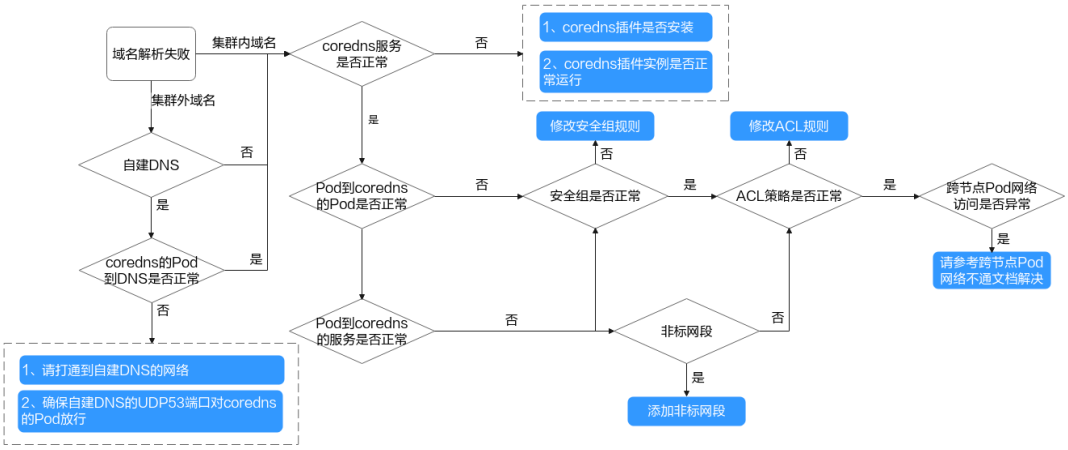
CCE 集群中域名解析失败。

排查思路

以下排查思路根据原因的出现概率进行排序，建议您从高频率原因往低频率原因排查，从而帮助您快速找到问题的原因。

如果解决完某个可能原因仍未解决问题，请继续排查其他可能原因。

图7-9 域名解析失败排查思路



当遇到域名解析失败的问题时，首先需要判断是集群内域名还是集群外域名解析失败。

- 集群内域名：

1. 确认 coredns 插件是否安装，coredns 服务是否正常运行。
2. 其他 Pod 到 coredns 的 Pod 网络是否正常，其他 Ood 到 coredns 的服务是否网络正常，如网络不正常：
   a) 安全组是否正常
   b) ACL 是否正常
   c) 是否跨界点 Pod 网络是否正常，如果跨节点 Pod 网络不通则需要确认以下问题是否存在：
      i. 修改了节点内核
      ii. 安全组和 ACL 策略未放通
      iii. VPC 路由表是否正常
      iv. 节点上的 iptables 规则是否正常
      v. 内核其他参数
      vi. 非标网段
   d) VPC 子网是否属于标准的私有网段 IP（如非标网段，请在 YangtseConfiguration 中添加）

- 集群外域名：

1. 确定是否为自建 DNS（容器如果未走 coredns 或者节点 DNS 非本 region 云解析的地址均属自建）
2. coredns 到自建 DNS 网络是否正常，工作负载到自建 DNS 的网络是否正常，如不正常：

a) 请打通到自建 DNS 网络

b) 请确保 DNS 的 UDP53 端口放行，需要对 Pod 网段安全组&ACL 放通如下策略：

i. 节点网段到节点网段

ii. 节点网段到容器网段

iii. 容器网段到节点网段

iv. 容器网段到容器网段

7.2.5 为什么访问部署的应用时浏览器返回 404 错误码？

CCE 服务本身在浏览器中访问应用时不会返回任何的错误码，请优先排查自身业务。

404 Not Found

如果 404 的返回如下图所示，说明这个返回码是 ELB 返回的，说明 ELB 找不到相关的转发策略。请排查相关的转发规则等。

图7-10 404:ALB

如果 404 的返回如下图所示，说明这个返回码是由 nginx（客户业务）返回，请排查客户自身业务问题。

图7-11 404:nginx/1.**.*

7.2.6 为什么容器无法连接互联网？

当容器无法连接互联网时，请排查容器所在节点能否连接互联网。

排查项一：节点能否连接互联网

步骤 1 登录 ECS 控制台。

步骤 2 查看节点对应的弹性云服务器是否已绑定弹性 IP 或者配置 NAT 网关。

如下图，若弹性 IP 一栏有 IP 地址，表示已绑定弹性 IP；若没有，请为弹性云服务器绑定弹性 IP。

图7-12 节点是否已绑定弹性 IP

----结束

排查项二：节点是否配置网络 ACL

步骤 1 登录 VPC 控制台。

步骤 2 单击左侧导航栏的“访问控制 > 网络 ACL”。

步骤 3 排查节点所在集群的子网是否配置了网络 ACL，并限制了外部访问。

----结束

7.2.7 VPC 的子网无法删除，怎么办？

VPC 的子网无法删除可能是因为您在 CCE 的集群中使用了该 VPC 的子网，因此需要在 CCE 界面删除相应的集群后，再删除 VPC 的子网。

须知

- 删除集群会将集群内的节点以及运行的工作负载和服务都销毁，请谨慎操作。
- 不建议在 ECS 界面删除 CCE 集群中的节点。

7.2.8 如何修复出现故障的容器网卡？

容器的网卡出现故障，会导致容器不断重启，且该容器无法对外提供服务。可通过如下步骤修复出现故障的容器网卡：

操作步骤

步骤 1 执行如下命令，删除故障容器的 Pod。

kubectl delete pod {podName} -n {podNamespace}

其中：

- **{podName}**：替换为实际故障容器所属 pod 名称。
- **{podNamespace}**：替换为实际 Pod 所在的 namespace 名称。

步骤 2 删除故障容器的 Pod 之后系统自动为容器重建 Pod，从而修复容器网卡。

----结束

7.2.9 节点无法连接互联网（公网），如何排查定位？

当节点无法连接互联网时，请参照如下方法排查。

排查项一：节点是否绑定弹性 IP

登录 ECS 控制台，查看节点对应的弹性云服务器是否已绑定弹性 IP。

如下图，若弹性 IP 一栏有 IP 地址，表示已绑定弹性 IP。若没有，请为弹性云服务器绑定弹性 IP。

图7-13 节点是否已绑定弹性 IP

排查项二：节点是否配置网络 ACL

登录 VPC 控制台，单击左侧导航栏的“访问控制 > 网络 ACL”。排查节点所在集群的子网是否配置了网络 ACL，并限制了外部访问。

7.2.10 如何解决 VPC 网段与容器网络冲突的问题？

在集群创建页面，若“容器网段”配置与“VPC 网段”冲突，界面会提示“该网段与 VPC 网段有冲突，请重新选择”，重新调整“容器网段”即可。

图7-14 网段冲突提示

7.2.11 ELB 四层健康检查导致 java 报错：Connection reset by peer

完整错误信息

```
java.io.IOException: Connection reset by peer
at sun.nio.ch.FileDispatcherImpl.read0(Native Method)
at sun.nio.ch.SocketDispatcher.read(SocketDispatcher.java:39)
at sun.nio.ch.IOUtil.readIntoNativeBuffer(IOUtil.java:223)
at sun.nio.ch.IOUtil.read(IOUtil.java:197)
at sun.nio.ch.SocketChannelImpl.read(SocketChannelImpl.java:380)
at
com.wanyu.smarthome.gateway.EquipmentSocketServer.handleReadEx(EquipmentSocketServe
r.java:245)
at
com.wanyu.smarthome.gateway.EquipmentSocketServer.run(EquipmentSocketServer.java:11
5)
```

分析结果

使用 Java NIO 建立 Socket 服务端，当客户端意外关闭的情况，不是发送指定指令通知服务器退出，就会产生此错误。

TCP 健康检查的机制

1. ELB 节点根据健康检查配置，向后端服务器（IP+健康检查端口）发送 TCP SYN 报文。
2. 后端服务器收到请求报文后，如果相应的端口已经被正常监听，则会返回 SYN+ACK 报文。

3. 如果在超时时间内没有收到后端服务器的 SYN+ACK 报文，则判定健康检查失败，然后发送 RST 报文给后端服务器中断 TCP 连接。
4. 如果在超时时间内收到了 SYN+ACK 报文，则发送 ACK 给后端服务器，判定健康检查成功，并发送 RST 报文给后端服务器中断 TCP 连接。

注意

正常的 TCP 三次握手后，会进行数据传输，但是在健康检查时会发送 RST 中断建立的 TCP 连接。该实现方式可能会导致后端服务器中的应用认为 TCP 连接异常退出，并打印错误信息，如“Connection reset by peer”。

这种错误如偶现，属于合理范围内。

7.2.12 Service 事件：Have no node to bind，如何排查？

步骤 1 登录 CCE 控制台，进入集群，在左侧导航栏选择“服务发现”。

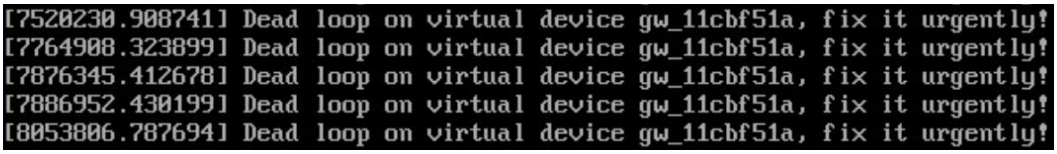
步骤 2 在 service 列表里确认此服务是否有关联的工作负载，或关联的工作负载的相关实例是否正常。

----结束

7.2.13 为什么登录虚拟机 VNC 界面会间歇性出现 Dead loop on virtual device gw_11cbf51a, fix it urgently？

问题现象

VPC 网络模式的集群，登录虚拟机出现 Dead loop on virtual device gw_11cbf51a, fix it urgently，如图：



原因定位

VPC 网络模式的集群采用了 linux 开源社区的 ipvlan 模块实现容器网络通信，这一日志打印与 ipvlan 的设计实现有关，ipvlan L2E 模式它优先进行二层转发，再进行三层转发。

场景还原：

假定有业务 Pod A，它持续对外提供服务，不断被同节点访问收发报文，通过本机 k8s service 经过容器 gw 接口进行访问，或者同属本节点的 Pod 间直接互相访问。在升级、缩容，或者其他原因导致的退出场景，容器 A 已停止运行，对应的网络资源被回收。此时同节点的报文仍持续尝试在往容器 A 的 IP 发送报文。内核中的 ipvlan 模块首先尝试根据目的 IP 来二层转发这些报文，但是由于 Pod A 已无法找到该 IP 所属的网卡，ipvlan 模块判断它有可能是个外部报文，因此尝试进行三层转发，根据路由规则匹配上了 gw 口，因此 gw 口又收到此报文，再经由 ipvlan 模块转发，如此循环。内核中

的 dev_queue_xmit 函数检测到重复进入发包过程达 10 次，报文被丢弃同时打印该日志。

发起访问端的在报文丢失后一般会进行几次退避重试，因此在这种场景下会连续打印几次条日志，直到发起访问端的容器内 ARP 老化或业务自身终止访问。

跨节点容器间通信，由于目的 IP 及源 IP 不属于同个节点级专属子网（注意此子网与 VPC 子网概念不同），报文不会重复走到此业务流程因此，不会触发此问题。

同集群不同节点间的 Pod 通过 Cluster 模式的 NodePort 来访问除外，它会被 SNAT 成被访问端容器 gw 接口的 IP，因此也有可能触发此日志打印。

问题影响

被访问端容器正常运行时不会有影响。容器被销毁时，有一定影响，但影响较小，重复进入发包过程 10 次，然后被丢包，这个过程在内核中处理十分迅速，对性能影响可以忽略。

对于 ARP 老化或业务自身不再重试，或新容器会被拉起，容器 service 服务报文经过 kubeproxy 重定向到新业务。

开源现状

目前开源社区 ipvlan L2E 模式仍存在此问题，已向开源社区反馈，待确认更优方案。

解决方法

打印 Dead loop 问题本身无需解决。

但我们推荐服务 Pod 使用优雅退出，在业务真正终止服务前，优先将 Pod 设置为删除中的状态，待业务处理完毕请求后再退出。

7.2.14 集群节点使用 networkpolicy 概率性出现 panic 问题

问题场景

集群版本：v1.15.6-r1 版本

集群类型：CCE 集群

网络模式：容器隧道网络模式

节点操作系统：CentOS 7.6

上述集群的用户配置使用 networkpolicy 后，由于节点上 canal-agent 网络组件与 CentOS 7.6 内核存在不兼容，概率性导致 CentOS 7.6 的节点 panic。

受影响范围

以下三条为必要条件，若有一条不满足，则您不会受到此问题的影响：

- v1.15.6-r1 容器隧道网络模式集群。

- canal-agent 组件版本为 1.0.RC10.1230.B005 或更低版本的 CentOS 7.6 节点（简单的判断方法为 2021 年 2 月 23 日及之前创建的节点）。
- 计划使用或已配置使用 networkpolicy 规则。

排查方法

快速排查方法（适用于节点为按需计费类型）

若您的节点为按需计费类型，可从 cce-console 上查看节点创建时间，对创建于 2021 年 2 月 24 日及之后的新建 CentOS 7.6 节点已无该问题。

准确排查方法（通用）

若您的集群版本为 v1.15.6-r1，网络模式为容器隧道网络，节点操作系统为 CentOS 7.6，而 canal-agent 组件版本为 1.0.RC10.1230.B005.sp1 或更高版本，则无此问题；若低于此版本（例如 1.0.RC10.1230.B005、1.0.RC10.1230.B003、1.0.RC10.1230.B002），则建议您重置/删除对应节点后再使用 networkpolicy。

执行以下步骤查询节点网络组件版本：

步骤 1 准备可执行 kubectl 的节点。

步骤 2 执行如下命令查询存量 CentOS 节点列表：

```
for node item in $(kubectl get nodes --no-headers | awk '{print $1}') ; do kubectl get node ${node_item} -o yaml | grep CentOS >/dev/null; if [[ "$?" == "0" ]];then echo "${node_item} is CentOS node";fi;done
```

回显如图：

```
10.0.50.187 is CentOS node
10.0.50.220 is CentOS node
10.0.50.43 is CentOS node
```

步骤 3 假定用户 CentOS 节点的 ip 为 10.0.50.187 (请替换为实际 IP)，执行下述命令查看 canal-agent 版本：

```
kubectl get packageversions.version.cce.io 10.0.50.187 -o yaml | grep -A 1 canal-agent
```

回显如图：

```
  - name: canal-agent
    version: 1.0.RC10.1230.B005.sp1
```

----结束

解决办法

如果您希望继续使用该节点资源，建议重置所属集群中的 CentOS 7.6 节点，以升级节点上网络组件到最新版本。如果您希望删除该隐患节点后重新购买亦可。

7.2.15 节点远程登录界面(VNC)打印较多 source ip_type 日志问题

问题场景

集群版本：v1.15.6-r1 版本

集群类型：CCE 集群

网络模式：VPC 网络

节点操作系统：CentOS 7.6

上述节点的容器进行容器间通信时，由于容器网络组件在 VNC 界面打印较多 source ip_type 或者 not ipvlan but in own host 日志，导致影响节点 VNC 界面使用体验及高负载场景下的容器网络性能。现象如下：

```
[ 3840.916433] ==========source ip_type 2, ipv4 10.0.0.128, mac fa:16:3e:57:f2:8f
[ 3840.916527] ==========source ip_type 2, ipv4 10.0.0.129, mac fa:16:3e:57:f2:8f
[ 3840.916736] ==========source ip_type 2, ipv4 10.0.0.129, mac fa:16:3e:57:f2:8f
```

```
[16739.000551] =====not ipvlan but in own host, mac_src=fa:16:3e:34:23:93 mac_dst=ff:ff:ff:ff:ff:ff
[16740.000968] =====not ipvlan but in own host, mac_src=fa:16:3e:34:23:93 mac_dst=ff:ff:ff:ff:ff:ff
```

排查方法

1、快速排查方法

适用于节点为按需计费类型，若您的节点为该类型，可从 cce-console 上查看节点创建时间，2021 年 2 月 24 日及之后创建的 CentOS 7.6 节点无该问题。

2、准确排查方法（通用）

您可以执行下述步骤排查节点是否受此问题影响：

步骤 1 以 root 用户登录 CCE 集群节点。

步骤 2 执行下述命令排查是否为隐患节点：

```
ETH0_IP=$(ip addr show eth0 | grep "inet " | head -n 1 | awk '{print $2}' | awk -F
'/' '{print $1}') ;arping -w 0.2 -c 1 -I gw_11cbf51a 1.1.1.1 >/dev/null 2>&1 ;
echo ;dmesg -T | grep -E "==not ipvlan but in own host|==source ip_type"
1>/dev/null 2>&1 ; if [[ "$?" == "0" ]];then echo "WARNING, node ${ETH0_IP} is
affected"; else echo "node ${ETH0_IP} works well"; fi;
```

说明

参考命令中 1.1.1.1 为示例 IP，该 IP 仅用于触发发送 ARP 报文。可直接使用，也可替换为任意合法 IP。

步骤 3 若回显为如下则表示节点存在隐患（10.2.0.35 为示例节点 eth0 网卡 IP，具体以实际回显为准）：

```
WARNING, node 10.2.0.35 is affected
```

若回显如下则表示节点无此问题：

```
node 10.2.0.35 works well
```

----结束

解决办法

如果您希望继续使用该节点资源，建议重置所属集群中的 CentOS 7.6 节点，以升级节点上网络组件到最新版本。如果您希望删除该隐患节点后重新购买亦可。

7.3 安全加固

7.3.1 集群节点如何不暴露到公网？

- 如果不需要访问集群节点的 22 端口，可在安全组规则中禁用 22 端口的访问。
- 如非必须，集群节点不建议绑定 EIP。

如有远程登录集群节点的需求，推荐使用云堡垒机服务作为中转连接集群节点。

7.4 网络指导

7.4.1 CCE 如何与其他服务进行内网通信？

与 CCE 进行内网通信的常见云服务有：RDS、DMS、Kafka、RabbitMQ、VPN、ModelArts 等，有如下两种场景：

- 在同一个 VPC 网络下，CCE 节点可以与此 VPC 下的所有服务进行互通。CCE 的容器与其他服务通信时，需要关注对端是否开启了容器网段的入方向的安全组规则（此限制只针对于 CCE 的集群为 VPC 网络模式）。
- 如果不在同一个 VPC，可以使用“对等连接”或 VPN 的方式打通两个 VPC，不过需要注意的是两个 VPC 的网段及容器网段不能重复。并且，对端 VPC 或小网需要配置回程路由（此限制只针对于 CCE 的集群为 VPC 网络模式）。

须知

- 此逻辑针对于天翼云所有服务均有效。
- “容器隧道网络”的集群，天然支持各服务间内网通信，不需要另外配置。
- “VPC 网络”模型的集群需要注意的事项：
  - 对端看到的来源 IP 为容器 IP。

1. 容器在 VPC 内的节点上互通，是通过 CCE 添加的自定义路由规则实现的。
2. CCE 中的容器访问其他服务时，需要关注**对端（目的端）是否开启了容器网段入方向的安全组规则**或**防火墙**。
3. 如果使用 vpn 或“对等连接”等方式打通了小网通信，需要在中间路上和目的地，都需要**在对等连接添加容器网段的路由**。

7.4.2 使用 CCE 设置工作负载访问方式时，端口如何填写？

CCE 支持工作负载的内部互访和被互联网访问两种方式。

在设置工作负载访问方式时，一般需要填写两个端口号，分别为“容器端口”和“访问端口”。

“容器端口”的含义是一致的，指容器中工作负载启动监听的端口。端口根据每个业务的不同而不同，一般在容器镜像中已指定。

“访问端口”，需根据选择访问类型的不同来填写。当前访问类型分为如下内容。

- 内部访问：包括“集群虚拟 IP”和“节点私有 IP”两种方式。

表7-6 内部访问类型说明

内部访问类型	说明	访问端口如何填写
集群虚拟 IP	用于“工作负载之间的互访”，例如某个后端工作负载需要和前端工作负载互访，就需要选择该类型来实现内部互访。 集群虚拟 IP，即 Cluster IP，是在工作负载设置此访问类型后，就会自动分配一个可用的 Cluster IP。	访问端口：指该容器工作负载发布为服务后，所设定的服务端口号，请填写 1-65535 之间的整数值。在内部工作负载互访时，将通过“Cluster IP:访问端口”来访问。
节点私有 IP	可以通过“节点 IP:节点端口”的形式访问工作负载，若该节点绑定了弹性 IP，则外网就可以访问该工作负载。	访问端口：指容器映射到节点上的端口。配置完成后，系统会在用户所在项目的所有节点上打开一个真实的端口号。访问工作负载时可以通过“节点 IP:访问端口”来访问工作负载。 如无特殊需求，选择“自动生成”即可，系统会自动分配访问端口号。若选择“指定端口”，请填写 30000-32767 之间的整数，且确保集群内该值唯一。

- 外部访问：包括“弹性 IP”和“负载均衡”两种方式。

表7-7 外部访问类型说明

外部访问类型	说明	端口如何填写
弹性 IP	为节点绑定弹性 IP，访问工作负载时，通过“节点弹性 IP:节点端口”的形式访问工作负载。工作负载可被公网访问。	访问端口：指容器映射到节点上的端口。配置完成后，系统会在用户所在项目的所有节点上打开一个真实的端口号。访问工作负载时可以通过“节点IP:访问端口”来访问工作负载。 如无特殊需求，选择“自动生成”即可，系统会自动分配访问端口号。若选择“指定端口”，请填写 30000-32767 之间的整数，且确保集群内该值唯一。
负载均衡	负载均衡通过将访问流量自动分发到多台节点，扩展工作负载系统对外的服务能力，实现更高水平的工作负载程序容错性能。 您需要提前创建负载均衡实例，并在 CCE 访问类型中选择负载均衡实例。	访问端口：代表负载均衡上注册的对外端口，外部访问使用 ELB 的 VIP+服务端口。

7.4.3 Ingress 中的 property 字段如何实现与社区 client-go 兼容？

使用场景

社区 Ingress 结构体中没有 property 属性，导致用户使用 client-go 调用创建 ingress 的 api 接口时，创建的 Ingress 中没有 property 属性。为了与社区的 client-go 兼容，CCE 提供了如下解决方案。

解决方案

在使用 client-go 创建 Ingress 实例时，在 annotation 中做如下声明：

```
kubernetes.io/ingress.property:
'[{"host":"test.com","path":"/test","matchmode":"STARTS_WITH"},{"host":"test.com","
path":"/dw","matchmode":"EQUAL_TO"}]'
```

匹配规则：用户调用 CCE 的 Kubernetes 接口创建 Ingress 时，会试图匹配 Ingress rules 中的 host 和 path 两个字段，如果和 annotation 中的 host 和 path 一致，则会在这条 path 下面注入 propery 属性，示例如下：

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: test
  namespace: default
  resourceVersion: '2904229'
  generation: 1
```

```
  labels:
    isExternal: 'true'
    zone: data
  annotations:
    kubernetes.io/ingress.class: cce
    kubernetes.io/ingress.property:
'[{"host":"test.com","path":"/test","matchmode":"STARTS_WITH"},{"Path":"/dw","Match
Mode":"EQUAL_TO"}]'
spec:
  rules:
    - host: test.com
      http:
       paths:
         - path: /ss
           backend:
             serviceName: zlh-test
             servicePort: 80
         - path: /dw
           backend:
             serviceName: zlh-test
             servicePort: 80
```

转换后的格式如下：

```
kind: Ingress
apiVersion: extensions/v1beta1
metadata:
  name: test
  namespace: default
  resourceVersion: '2904229'
  generation: 1
  labels:
    isExternal: 'true'
    zone: data
  annotations:
    kubernetes.io/ingress.class: cce
    kubernetes.io/ingress.property:
'[{"host":"test.com","path":"/ss","matchmode":"STARTS_WITH"},{"host":"","path":"/dw
","matchmode":"EQUAL_TO"}]'
spec:
  rules:
    - host: test.com
      http:
       paths:
         - path: /ss
           backend:
             serviceName: zlh-test
             servicePort: 80
           property:
             ingress.beta.kubernetes.io/url-match-mode: STARTS_WITH
         - path: /dw
           backend:
             serviceName: zlh-test
             servicePort: 80
```

表7-8 关键参数说明

参数	参数类型	描述
host	String	域名配置。 若不配置，会自动匹配 path。
path	String	匹配路径。
ingress.beta.kubernetes.io/url-match-mode	String	路由匹配策略，取值如下： • REGEX：正则匹配 • STARTS_WITH：前缀匹配 • EQUAL_TO：精确匹配

7.5 其他

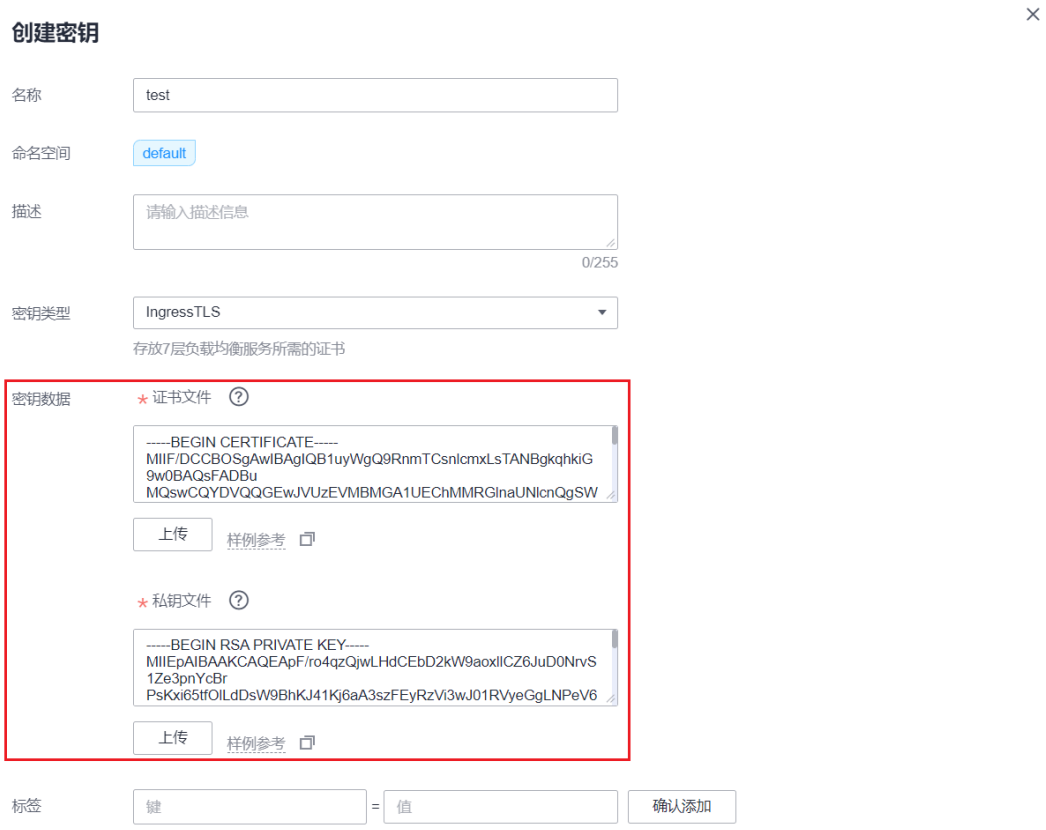
7.5.1 如何获取 TLS 密钥证书？

场景

当您的 Ingress 需要使用 HTTPS 协议时，创建 Ingress 时必须配置 IngressTLS 或 kubernetes.io/tls 类型的密钥。

以创建 IngressTLS 密钥证书为例：

图7-15 创建密钥



这里上传的证书文件和私钥文件必须是配套的，不然会出现无效的情况。

解决方法

一般情况下，您需要从证书提供商处获取有效的合法证书。如果您需要在测试环境下使用，您可以自建证书和私钥，方法如下：

说明

自建的证书通常只适用于测试场景，使用时界面会提示证书不合法，影响正常访问，建议您选择手动上传合法证书，以便通过浏览器校验，保证连接的安全性。

1. 自己生成 tls.key。

```
openssl genrsa -out tls.key 2048
```

将在当前目录生成一个 tls.key 的私钥。

2. 用此私钥去签发生成自己的证书。

```
openssl req -new -x509 -key tls.key -out tls.crt -subj
/C=CN/ST=Beijing/O=Devops/CN=example.com -days 3650
```

生成的私钥格式必须为：

```
----BEGIN RSA PRIVATE KEY-----
……..
-----END RSA PRIVATE KEY-----
```

生成的证书格式必须为：

```
-----BEGIN CERTIFICATE-----
……
-----END CERTIFICATE-----
```

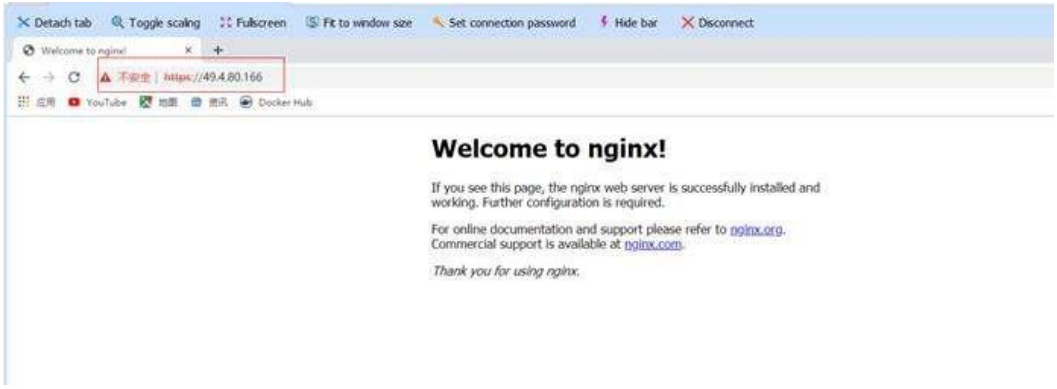
3. 导入证书。

   新建 TLS 密钥时，对应位置导入证书及私钥文件即可。

验证

通过浏览器访问 Ingress 地址可以正常访问，但因为是自己签发的证书和密钥，所以这里 CA 不认可，显示不安全。

图7-16 验证结果



7.5.2 CCE 集群的节点是否支持绑定多网卡？

支持使用多网卡。

但不建议用户手动给 CCE 集群的节点绑定多网卡，绑定多网卡后配置的规则有可能会影响到 CCE Turbo 集群的访问。

7.5.3 服务发布到 ELB，ELB 的后端为何会被自动删除？

问题描述：

服务发布到 ELB，工作负载已正常，但服务的 pod 端口未及时发布出来，ELB 里的后端会被自动删除。

问题解答：

1. 创建 ELB 时候，如果 ELB 监控检查失败，后端服务器组会删除，而且后续服务正常以后也不会添加。如果是更新已有的 SVC 时则不会删除。
2. 添加删除节点的时候，由于集群状态的改变，可能会引起集群内的 Node 访问方式的改变，为保证服务正常运行，所以 ELB 会进行一次刷新操作，这个过程类似于更新 ELB。

修复建议：

优化应用，加快应用的启动速度。

7.5.4 为什么更换命名空间后无法创建 ingress？

问题描述

在 default 命名空间下可以正常创建 ingress，但在其他命名空间（如:ns）下创建时不能创建成功。

原因分析

创建弹性负载均衡 ELB 后，使用 default 命名空间创建 80 端口的 http 监听，在 CCE 中只允许在本命名空间下创建同一端口的其它 ingress（实际转发策略可根据域名、service 来区分）；所以出现客户侧在其它命名空间无法创建相同端口的 ingress 的情况（会提示端口冲突）。

解决方法

可以使用 yaml 创建，端口冲突只有前台限制，后台未限制。

7.5.5 服务加入 Istio 后，如何获取客户端真实源 IP？

问题现象

服务启用 Istio 后，访问日志获取到的客户端源 IP 全部变为 127.0.0.1，如何获取到真实的源 IP？

解决方案

本文以 ELB 类型 Service 加入服务网格的 nginx 应用为例，其他类型的 Service 方法类似，详细步骤如下：

步骤 1 创建 ELB 类型的 Service（节点级别）。

步骤 2 加入 Istio 服务列表。

步骤 3 添加 Istio 服务网关并将 service 下 istio-system 命名空间下自动生成的 Service 改为节点级别。

步骤 4 在容器内进行 nginx 配置：

配置路径为：/etc/nginx/conf.d/default.conf

```
set_real_ip_from 100.0.0.0/8;
set_real_ip_from 127.0.0.1;
real_ip_header X-Forwarded-For;
real_ip_recursive on;
```

```
server {
    listen       80;
    server_name  localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    set_real_ip_from 100.0.0.0/8;
    set_real_ip_from 127.0.0.1;
    real_ip_header X-Forwarded-For;
    real_ip_recursive on;
```

步骤 5 集群所有节点加载 TOA 插件，这样就可以在容器中获取到源 IP 了。

----结束

说明

- 请求从客户端发出后，经过 ELB 转换了一次，然后到 istio 的时候会再转换一次，所以 set_real_ip_from 需要写两次转换的，然后就可以拿到真实 IP 了。
- 当前后端服务为 http 协议时验证正常，服务端为 https 协议目前没有验证。

7.5.6 如何批量修改集群 node 节点安全组？

约束与限制

一个安全组关联的实例数量建议不超过 1000 个，否则可能引起安全组性能下降。

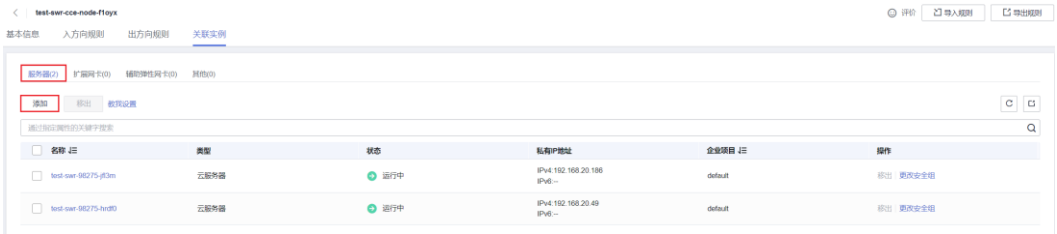
操作步骤

步骤 1 登录 VPC 控制台，并在左上角选择区域和项目。

步骤 2 在左侧导航树选择“访问控制 > 安全组”。

步骤 3 在安全组界面，单击操作列的“管理实例”。

步骤 4 在“服务器”页签，并单击“添加”。



步骤 5 勾选需要加入安全组的服务器，单击“确定”。您也可以通过服务器的名称、ID、私有 IP 地址、状态、企业项目或标签进行筛选。

通过修改左下角的单页最大显示条数，您可至多一次性添加 20 台服务器至安全组中。

说明

加入新的安全组后，节点仍保留原安全组。如需移除，请单击原安全组的“管理实例”按钮，并勾选其中的节点服务器进行移除。


添加服务器
安全组
选择服务器
云服务器
裸金属服务器
通过指定属性的关键字搜索
名称
ID
私有IP地址
状态
企业项目
标签
状态
私有IP地址
IPv6地址
企业项目
关机
192.168.0.70
--
default
关机
192.168.0.85
--
default
关机
192.168.48.116
--
default
关机
192.168.0.160
--
default
运行中
192.168.0.38
--
default
5
总条数：12
1
2
3
已选服务器
您还没选择服务器。

8 存储管理

8.1 CCE 支持的存储在持久化和多节点挂载方面的区别是怎样的？

容器存储是为容器工作负载提供存储的组件，支持多种类型的存储，同一个工作负载（pod)可以使用任意数量的存储。

当前云容器引擎 CCE 支持本地磁盘存储、云硬盘存储卷、文件存储卷、对象存储卷和极速文件存储卷。

各类存储的区别和对比如下：

表8-1 各类存储的区别和对比

存储类型	持久化存储	伴随容器自动迁移	多节点挂载
本地磁盘存储	支持	不支持	不支持
云磁盘存储卷（EVS）	支持	支持	不支持
对象存储卷（OBS）	支持	支持	支持，可由多个节点或工作负载共享
文件存储卷（SFS）	支持	支持	支持，可由多个节点或工作负载共享
极速文件存储卷（SFS Turbo）	支持	支持	支持，可由多个节点或工作负载共享

CCE 存储类型选择

创建工作负载时，可以使用以下类型的存储。建议将工作负载 pod 数据存储在云存储上。若存储在本地磁盘上，节点异常无法恢复时，本地磁盘中的数据也将无法恢复。

- 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 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（不填写源路径）适用于容器的临时存储。配置项（ConfigMap）是一种用于存储工作负载所需配置信息的资源类型，内容由用户决定。密钥（Secret）是一种用于存储工作负载所需要认证信息、密钥的敏感信息等的资源类型，内容由用户决定。
- 云硬盘存储卷：CCE 支持将 EVS 创建的云硬盘挂载到容器的某一路径下。当容器迁移时，挂载的云硬盘将一同迁移。这种存储方式适用于需要永久化保存的数据。
- 文件存储卷： CCE 支持创建 SFS 存储卷并挂载到容器的某一路径下，也可以使用底层 SFS 服务创建的文件存储卷，SFS 存储卷适用于多读多写的持久化存储，适用于多种工作负载场景，包括媒体处理、内容管理、大数据分析和分析工作负载程序等场景。
- 对象存储卷：CCE 支持创建 OBS 对象存储卷并挂载到容器的某一路径下，对象存储适用于云工作负载、数据分析、内容分析和热点对象等场景。
- 极速文件存储卷：CCE 支持创建 SFS Turbo 极速文件存储卷并挂载到容器的某一路径下，极速文件存储具有按需申请，快速供给，弹性扩展，方便灵活等特点，适用于 DevOps、容器微服务、企业办公等应用场景。

8.2 添加节点时可以不要 100G 数据盘吗？

不可以，100G 数据盘是必须要的。

新建节点会给节点绑定一个 100G 的 docker 专用数据盘。CCE 数据盘默认使用 LVM（Logical Volume Manager）进行磁盘管理，开启后您可以通过空间分配调整数据盘中不同资源的空间占比。

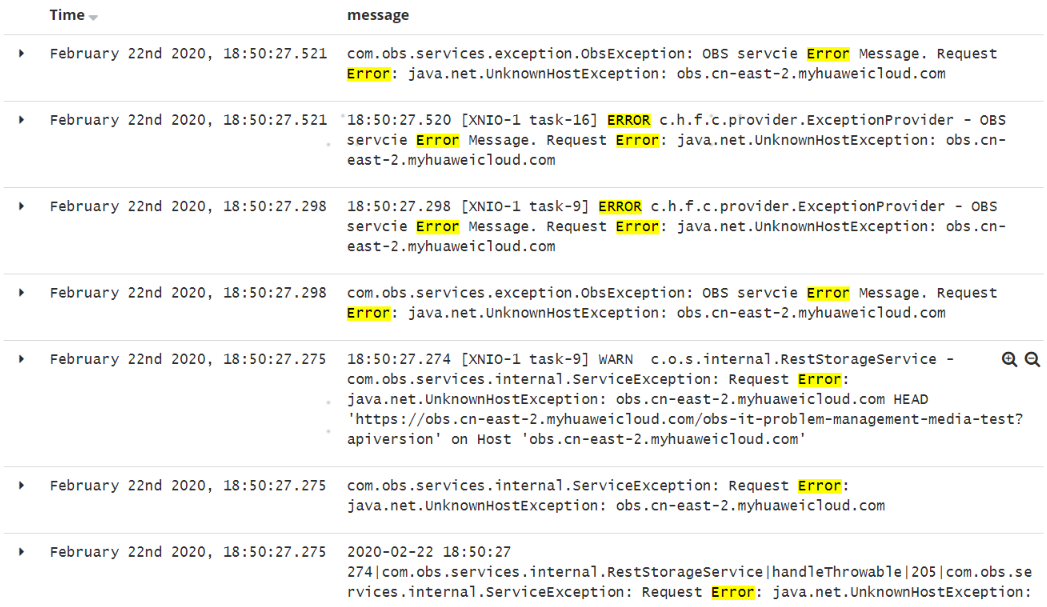
若数据盘卸载或损坏，会导致 docker 服务异常，最终导致节点不可用。

8.3 CCE 集群使用 EVS 做持久卷，在卷被删除或者过期后是否可以恢复？

云硬盘 EVS 存储需要人工配置备份策略。如果卷被删除或者释放，可以使用云硬盘备份恢复数据。

8.4 公网访问 CCE 部署的服务并上传 OBS，为何报错找不到 host？

线下机器访问 CCE 部署的服务并上传 OBS，报错找不到 host，报错截图如下：



Time	message
February 22nd 2020, 18:50:27.521	com.obs.services.exception.ObsException: OBS servcie Error Message. Request Error: java.net.UnknownHostException: obs.cn-east-2.myhuaweicloud.com
February 22nd 2020, 18:50:27.521	18:50:27.520 [XNIO-1 task-16] ERROR c.h.f.c.provider.ExceptionProvider - OBS servcie Error Message. Request Error: java.net.UnknownHostException: obs.cn-east-2.myhuaweicloud.com
February 22nd 2020, 18:50:27.298	18:50:27.298 [XNIO-1 task-9] ERROR c.h.f.c.provider.ExceptionProvider - OBS servcie Error Message. Request Error: java.net.UnknownHostException: obs.cn-east-2.myhuaweicloud.com
February 22nd 2020, 18:50:27.298	com.obs.services.exception.ObsException: OBS servcie Error Message. Request Error: java.net.UnknownHostException: obs.cn-east-2.myhuaweicloud.com
February 22nd 2020, 18:50:27.275	18:50:27.274 [XNIO-1 task-9] WARN c.o.s.internal.RestStorageService - com.obs.services.internal.ServiceException: Request Error: java.net.UnknownHostException: obs.cn-east-2.myhuaweicloud.com HEAD 'https://obs.cn-east-2.myhuaweicloud.com/obs-it-problem-management-media-test?apiversion' on Host 'obs.cn-east-2.myhuaweicloud.com'
February 22nd 2020, 18:50:27.275	com.obs.services.internal.ServiceException: Request Error: java.net.UnknownHostException: obs.cn-east-2.myhuaweicloud.com
February 22nd 2020, 18:50:27.275	2020-02-22 18:50:27 274|com.obs.services.internal.RestStorageService|handleThrowable|205|com.obs.services.internal.ServiceException: Request Error: java.net.UnknownHostException:

问题定位

服务收到 http 请求之后，向 OBS 传输文件，这些报文都会经过 Proxy。

传输文件总量很大的话，会消耗很多资源，目前 proxy 分配内存 128M，在压测场景下，损耗非常大，最终导致请求失败。

目前压测所有流量都经过 Proxy，业务量大就要加大分配资源。

解决方法

1. 传文件涉及大量报文拷贝，会占用内存，建议把 Proxy 内存根据实际场景调高后再进行访问和上传。
2. 可以考虑把该服务从网格内移除出去，因为这里的 Proxy 只是转发了一下包，并没有做其他事情，如果是通过 Ingress Gateway 走进来的话，这个服务的灰度发布功能是不受影响的。

8.5 弹性文件存储 SFS 最多可以挂载多少台节点（ECS）？

弹性文件存储卷为集群共享资源，可以挂载的 ECS 节点数量目前暂不受限。

8.6 Pod 接口 ExtendPathMode: PodUID 如何与社区 client-go 兼容？

使用场景

社区 Pod 结构体中没有 ExtendPathMode，用户使用 client-go 调用创建 pod 或 deployment 的 API 接口时，创建的 pod 中没有 ExtendPathMode。为了与社区的 client-go 兼容，CCE 提供了如下解决方案。

解决方案

须知

- 创建 pod 时，在 pod 的 annotation 中需增加 **kubernetes.io/extend-path-mode。**
- 创建 deployment 时，需要在 template 中的 annotation 增加 **kubernetes.io/extend-path-mode。**

如下为创建 pod 的 yaml 示例，在 annotation 中添加 **kubernetes.io/extend-path-mode** 关键字后，完全匹配到 containername，name，mountpath 三个字段，则会在 volumeMount 中增加对应的 **extendpathmode**：

```
apiVersion: v1
kind: Pod
metadata:
  name: test-8b59d5884-96vdz
  generateName: test-8b59d5884-
  namespace: default
  selfLink: /api/v1/namespaces/default/pods/test-8b59d5884-96vdz
  labels:
    app: test
    pod-template-hash: 8b59d5884
  annotations:
    kubernetes.io/extend-path-mode: '[{"containername":"container-0","name":"vol-
156738843032165499","mountpath":"/tmp","extendpathmode":"PodUID"}]'
    metrics.alpha.kubernetes.io/custom-endpoints:
'[{"api":"","path":"","port":"","names":""}]'
  ownerReferences:
    - apiVersion: apps/v1
      kind: ReplicaSet
      name: test-8b59d5884
      uid: 2633020b-cd23-11e9-8f83-fa163e592534
      controller: true
      blockOwnerDeletion: true
spec:
  volumes:
    - name: vol-156738843032165499
      hostPath:
        path: /tmp
```

```
      type: ''
  - name: default-token-4s959
    secret:
      secretName: default-token-4s959
      defaultMode: 420
containers:
  - name: container-0
    image: 'nginx:latest'
    env:
      - name: PAAS_APP_NAME
        value: test
      - name: PAAS_NAMESPACE
        value: default
      - name: PAAS_PROJECT_ID
        value: b6315dd3d0ff4be5b31a963256794989
    resources:
      limits:
        cpu: 250m
        memory: 512Mi
      requests:
        cpu: 250m
        memory: 512Mi
    volumeMounts:
      - name: vol-156738843032165499
        mountPath: /tmp
        extendPathMode: PodUID
      - name: default-token-4s959
        readOnly: true
        mountPath: /var/run/secrets/kubernetes.io/serviceaccount
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    imagePullPolicy: Always
restartPolicy: Always
terminationGracePeriodSeconds: 30
dnsPolicy: ClusterFirst
serviceAccountName: default
serviceAccount: default
nodeName: 192.168.0.24
securityContext: {}
imagePullSecrets:
  - name: default-secret
  - name: default-secret
affinity: {}
schedulerName: default-scheduler
tolerations:
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    tolerationSeconds: 300
priority: 0
dnsConfig:
```

```
  options:
    - name: timeout
      value: ''
    - name: ndots
      value: '5'
    - name: single-request-reopen
enableServiceLinks: true
```

表8-2 关键参数说明

参数	参数类型	描述
containername	String	容器名称。
name	String	volume 的名称。
mountpath	String	挂载路径
extendpathmode	String	将在已创建的“卷目录/子目录”中增加一个三级目录，便于更方便获取单个 Pod 输出的文件。 支持如下五种类型。 • None：不配置拓展路径。 • PodUID：Pod 的 ID。 • PodName：Pod 的名称。 • PodUID/ContainerName：Pod 的 ID/容器名称。 • PodName/ContainerName：Pod 名称/容器名称。

8.7 创建存储卷失败

现象描述

创建 PV 或 PVC 失败，在事件中看到如下信息。

```
{"message": "Your account is suspended and resources can not be used.", "code": 403}
```

问题根因

事件信息表示帐号被停用或没有权限，请检查帐号状态是否正常；如帐号正常请查看该用户的命名空间权限。

8.8 CCE 容器云存储 PVC 能否感知底层存储故障？

CCE PVC 按照社区逻辑实现，PVC 本身的定义是存储声明，与底层存储解耦，不负责感知底层存储细节，因此没有感知底层存储故障的能力。

云监控服务 CES 具备查看云服务监控指标的能力：云监控服务基于云服务自身的服务属性，已经内置了详细全面的监控指标。当用户在云平台上开通云服务后，系统会根据服务类型自动关联该服务的监控指标，帮助用户实时掌握云服务的各项性能指标，精确掌握云服务的运行情况。

建议有存储故障感知诉求的用户配套云监控服务 CES 的云服务监控能力使用，实现对底层存储的监控和告警通知。

8.9 无法使用 kubectl 命令删除 PV 或 PVC

现象描述

无法使用 kubectl delete 命令直接删除已有的 PV 或 PVC，删除后会一直处于 Terminating 状态。

问题根因

Kubernetes 为了防止误删除 PV 和 PVC 导致数据丢失，存在数据保护机制，无法使用 delete 命令直接删除。

解决方案

执行以下命令，先解除保护机制，再删除 PV 或 PVC。

如果已经使用 kubectl delete 命令删除 PV 或 PVC，会一直处在 Terminating 状态，在执行下面 patch 命令后会直接删除，无需重复执行 kubectl delete 命令。

PV：

```
kubectl patch pv <pv-name> -p '{"metadata":{"finalizers":null}}'
kubectl delete pv <pv-name>
```

PVC：

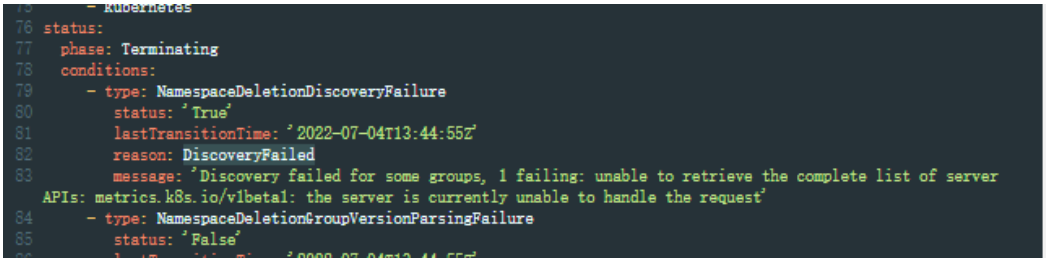
```
kubectl patch pvc <pvc-name> -p '{"metadata":{"finalizers":null}}'
kubectl delete pvc <pvc-name>
```

9 命名空间

9.1 命名空间因 APIService 对象访问失败无法删除

问题现象

删除命名空间时，命名空间一直处“删除中”状态，无法删除。查看命名空间 yaml 配置，status 中有报错“DiscoveryFailed”，示例如下：



上图中报错信息为：Discovery failed for some groups, 1 failing: unable to retrieve the complete list of server APIs: metrics.k8s.io/v1beta1: the server is currently unable to handle the request

表示当前删除命名空间动作阻塞在 kube-apiserver 访问 metrics.k8s.io/v1beta1 接口的 APIService 资源对象。

问题根因

当集群中存在 APIService 对象时，删除命名空间会先访问 APIService 对象，若 APIService 资源无法正常访问，会阻塞命名空间删除。除用户创建的 APIService 对象资源外，CCE 集群部分插件也会自动创建 APIService 资源，如 metrics-server, prometheus 插件。

说明

APIService 使用介绍请参考：https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/apiserver-aggregation/

解决方法

可以采用如下两种方法解决：

- 修复报错信息中的 APIService 对象，使其能够正常访问，如果是插件中的 APIService，请确保插件的 Pod 正常运行
- 删除报错信息中的 APIService 对象，如果是插件中的 APIService，可从页面卸载该插件

10 模板插件

10.1 集群安装 nginx-ingress 插件失败，一直处于创建中？

问题背景

客户已经购买并搭建了 CCE 集群，希望在公网上可以访问到 CCE 上部署的应用服务，目前最高效的方式是在 ingress 资源上注册该应用的 Service 路径，从而满足要求。

但客户安装 ingress 插件后，插件状态一直显示“创建中”，nginx-ingress-contoller 的 pod 一直处于 pending 状态。

解决方案

nginx 限制的内存资源不足导致无法启动，取消限制后正常。

场景模拟

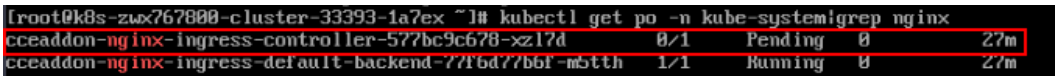
步骤 1 新集群 3 个节点，规格 6cpu，12G 内存，每个节点 2U4G。

步骤 2 单击 nginx-ingress 插件安装，选择规格 2 核 2G。

步骤 3 nginx-ingress deployment 安装成功，但是 nginx-ingress-controller 安装失败。

图10-1 一直处于创建中

图10-2 安装失败



步骤 4 错误显示资源不足。

```
status:
  phase: Pending
  conditions:
    - type: PodScheduled
      status: 'False'
      lastProbeTime: '2020-02-13T01:20:57Z'
      lastTransitionTime: '2020-02-12T08:50:21Z'
      reason: Unschedulable
      message: '0/3 nodes are available: 3 Insufficient cpu, 3 Insufficient memory.'
  qosClass: Guaranteed
```

步骤 5 添加节点资源为 4U8G 后，nginx-ingress 安装正常。

----结束

问题原因

最初建立的集群中各节点的基本配置为 2U4G，且各节点上有 kubelet，kube-proxy 及 docker 等相关程序占用系统资源，导致节点可用资源低于 2000m，无法满足 nginx-ingress 插件的要求，从而无法安装。

建议方案

请重新购买节点，节点要求（>=4U8G）。

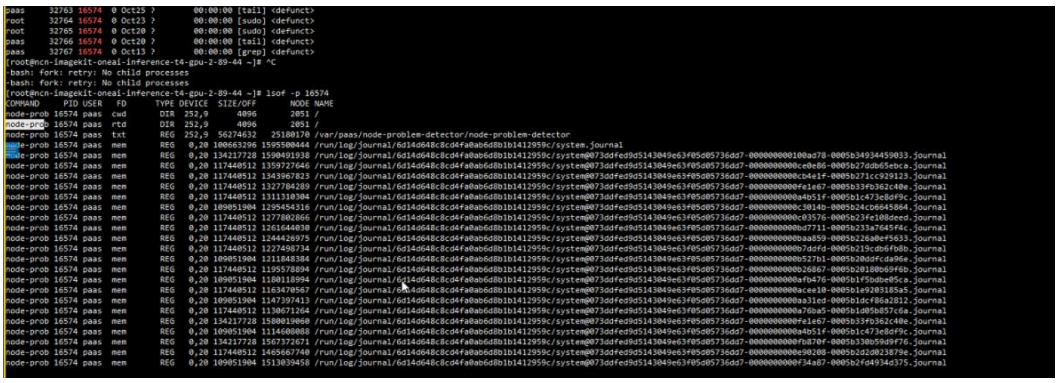
10.2 NPD 插件版本过低导致进程资源残留问题

问题描述

在节点负载压力比较大的场景下，可能存在 NPD 进程资源残留的问题。

问题现象

登录到 CCE 集群的 ECS 节点，查询存在大量 npd 进程。



解决方案

升级 npd 插件至最新版本。

步骤 1 登录 CCE 控制台，进入集群，在左侧导航栏中选择“插件管理”，在“已安装插件”下，单击 **npd** 下的“升级”。

说明

如果 npd 插件版本已经为 1.13.6 及以上版本，则不需要进行升级操作。

步骤 2 在基本信息页面选择插件版本（例如 1.13.6），单击“下一步”。

步骤 3 npd 插件暂未开放可配置参数，直接单击“升级”即可升级 npd 插件。

----结束

10.3 模板格式不正确，无法删除模板实例？

问题现象

若上传的模板中包含不正确或者不兼容的资源，会导致安装模板失败，类似下图：

此时模板实例无法正常工作。如果您尝试在界面上删除，可能会出现 deletion failed 的报错，模板实例仍在列表中：

解决方法

您可以使用 kubectl 命令删除残留的模板实例。

安装模板时，模板中的一些资源可能已经创建成功，因此首先要手动删除这些资源。确保残留的资源删除后，需删除模板实例。

若为 helm v2 的实例，在 kube-system 命名空间下查询模板实例对应的配置项（ConfigMap），例如：

```
[paas@192-168-0-40 ~]$ kubectl -n kube-system get cm
NAME                                 DATA   AGE
9a37566a.cce.io                      0      25d
aosredis.v1                          1      55s
cceaddon-coredns.v1                  1      25d
cceaddon-everest.v1                  1      17d
cceaddon-metrics-server.v1           1      25d
cceaddon-npd-custom-config           0      25d
cceaddon-npd.v1                      1      25d
cceaddon-prometheus.v1               1      25h
cluster-autoscaler-status            1      8d
cluster-versions                     1      25d
coredns                              1      25d
extension-apiserver-authentication   6      25d
ingress-controller-leader-nginx      0      22d
[paas@192-168-0-40 ~]$
```

删除该配置项，此时模板实例即删除成功：

```
[paas@192-168-0-40 ~]$ kubectl -n kube-system delete cm aosredis.v1
configmap "aosredis.v1" deleted
[paas@192-168-0-40 ~]$
```

若为 helm v3 的实例，在实例所在命名空间下查询模板实例对应的秘钥（Secret），例如：

```
[root@cce-1717-vpc-node2 ~]# kubectl -n default get secret
NAME                           TYPE                                  DATA   AGE
default-secret                 kubernetes.io/dockerconfigjson        1      21h
default-token-978pv            kubernetes.io/service-account-token   3      21h
paas.elb                       cfe/secure-opaque                     3      21h
sh.helm.release.v1.test-nginx.v1   helm.sh/release.v1                1      139m
[root@cce-1717-vpc-node2 ~]#
```

删除该秘钥，此时模板实例即删除成功：

```
[root@cce-1717-vpc-node2 ~]# kubectl -n default delete secret sh.helm.release.v1.test-nginx.v1
secret "sh.helm.release.v1.test-nginx.v1" deleted
[root@cce-1717-vpc-node2 ~]#
```

注：若用户通过前端 console 操作，在获取实例、更新实例等操作中 CCE 会自动尝试转换原 v2 模板实例到 v3 模板实例。在秘钥中存储 release 信息，原配置项中 release 信息不会删除。建议用户在配置项和秘钥中均查询并删除该实例。

10.4 CCE 是否支持 nginx-ingress？

nginx-ingress 简介

nginx-ingress 是比较热门的 ingress-controller，作为反向代理将外部流量导入到集群内部，将 Kubernetes 内部的 Service 暴露给外部，在 Ingress 对象中通过域名匹配 Service，这样就可以直接通过域名访问到集群内部的服务。

nginx-ingress 由 ingress-controller 和 nginx 组件组成：

- ingress-controller：负责监听 Kubernetes 的 Ingress 对象，更新 nginx 配置。

  说明

  Ingress 的具体说明，请参见 https://kubernetes.io/docs/concepts/services-networking/ingress/。

- nginx：负责请求负载均衡，支持 7 层转发能力。

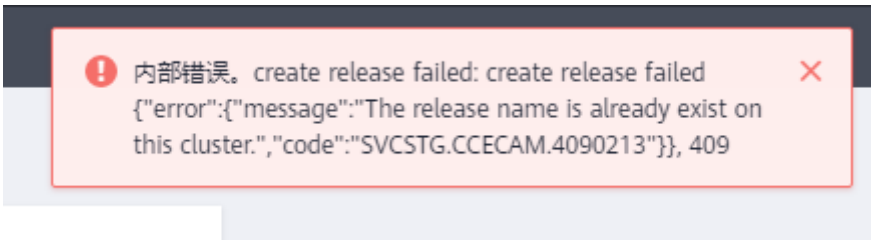
安装方法

目前 CCE 在插件市场中已提供 nginx-ingress 插件的一键式安装，也可以在安装界面展开参数进行设置。

10.5 插件安装失败，提示 The release name is already exist 处理

问题现象

当安装插件失败，返回 **The release name is already exist** 错误。



问题原因

当安装插件返回 **The release name is already exist** 错误时，表示 kuberneters 集群中有残留该插件 release 记录，一般由于集群 etcd 做过备份恢复或者该插件之前安装删除异常导致。

解决方案

通过 kubectl 对接集群，手动清理该插件 release 对应的 secret 及 configmap。以下以清理 autoscaler 插件 release 为示例。

步骤 1 配置 kubectl 对接集群后，执行以下命令查看插件相关的 release 的 secret 列表。

kubectl get secret -nkube-system |grep cceaddon

```
[root@cce-123-vpc-node2 ~]# kubectl get secret -nkube-system |grep cceaddon
sh.helm.release.v1.cceaddon-autoscaler.v1       helm.sh/release.v1                  1      61s
sh.helm.release.v1.cceaddon-autoscaler.v2       helm.sh/release.v1                  1      47s
sh.helm.release.v1.cceaddon-coredns.v1          helm.sh/release.v1                  1      6h2m
sh.helm.release.v1.cceaddon-everest.v1          helm.sh/release.v1                  1      6h2m
[root@cce-123-vpc-node2 ~]#
```

插件 release 的 secret 名称为"**sh.helm.release.v1.cceaddon-{插件名称}.v***"格式，可能有多个版本，删除时多个版本同时删除。

步骤 2 执行删除 release secret 命令。

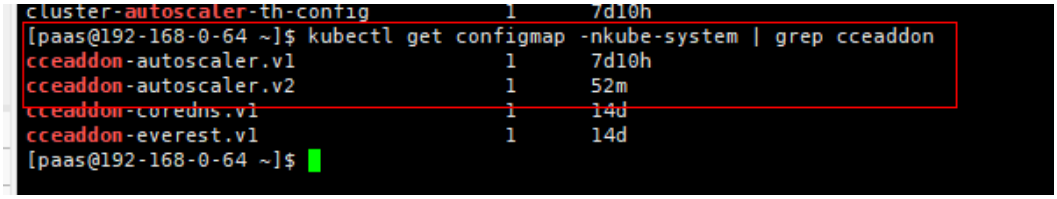
如删除上图中的 autoscaler 插件对应的 release secret

kubectl delete secret sh.helm.release.v1.cceaddon-autoscaler.v1 sh.helm.release.v1.cceaddon-autoscaler.v2 -nkube-system

```
[root@cce-123-vpc-node2 ~]# kubectl delete secret sh.helm.release.v1.cceaddon-autoscaler.v1 sh.helm.release.v1.cceaddon-autoscaler.v2 -nkube-system
secret "sh.helm.release.v1.cceaddon-autoscaler.v1" deleted
secret "sh.helm.release.v1.cceaddon-autoscaler.v2" deleted
[root@cce-123-vpc-node2 ~]#
```

步骤 3 若该插件为 helm v2 时创建，cce 会在查看插件列表及插件详情等操作中自动将 configmap 中的 v2 release 转换至 secret 中的 v3 release，原 configmap 中的 v2 release 不会删除。可执行以下命令查看插件相关的 release 的 configmap 列表。

kubectl get configmap -nkube-system | grep cceaddon

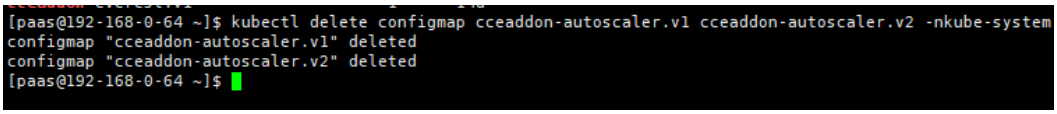


插件 release 的 configmap 名称为"**cceaddon-{插件名称}.v***"格式，可能有多个版本，删除时多个版本同时删除。

步骤 4 执行删除 release configmap 命令。

如删除上图中的 autoscaler 插件对应的 release configmap

kubectl delete configmap cceaddon-autoscaler.v1 cceaddon-autoscaler.v2 -nkube-system



⚠ 注意

删除 kube-system 下资源属高风险操作，请确保命令正确后再执行，以免出现误删资源。

步骤 5 在 CCE 控制台安装插件，然后再卸载保证之前的残留的插件资源清理干净，卸载完成后再进行第二次安装插件，安装成功即可。

说明

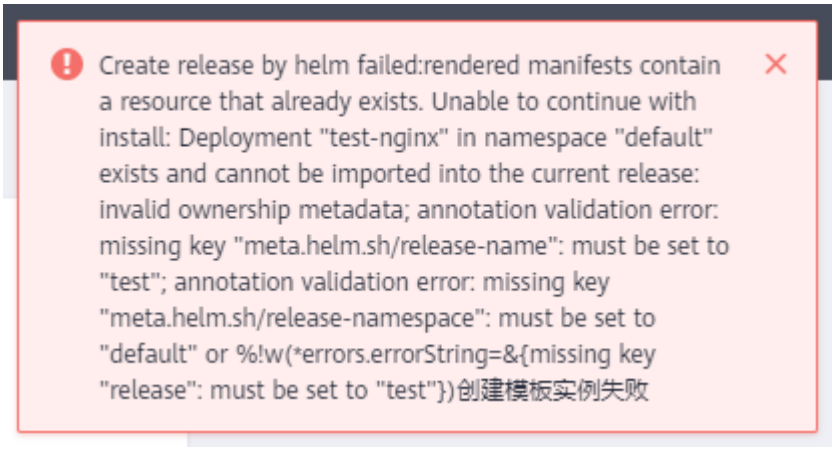
第一次安装插件是可能因之前的插件残留资源而导致安装后插件状态异常，属正常现象，这时在控制台卸载插件能保证这些残留资源清理干净，再次安装插件能正常运行。

----结束

10.6 创建或升级实例失败，提示 rendered manifests contain a resource that already exists

问题现象

创建或升级实例失败，提示“Create release by helm failed:rendered manifests contain a resource that already exists。Unable to continue with install: ..., label validation error:missing key \"app.kubernetes.io/managed-by\":must be set to\"Helm\" ... 创建模板实例失败”。



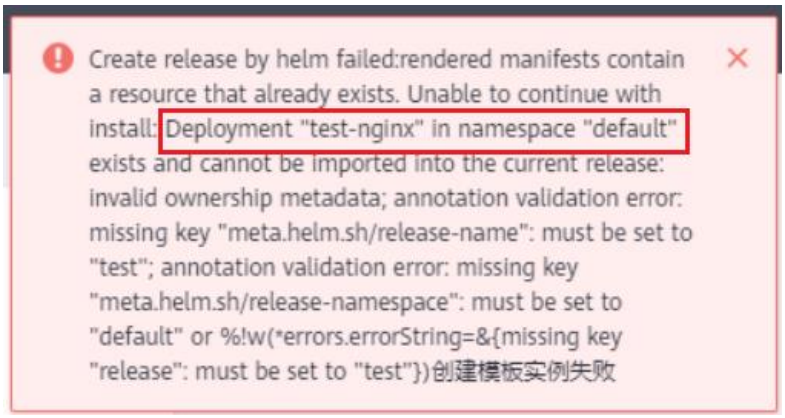
问题原因

当出现以上报错内容，说明该资源不为通过 helm v3 创建。若环境存在同名资源且无 helm v3 的归属标记“app.kubernetes.io/managed-by: Helm”时，则会提示资源冲突。

解决方案

删除相关报错资源，重新通过 helm 创建。

步骤 1 查看报错内容，确认产生冲突的资源。请您关注“Unable to continue with install:”后的信息，例如以下报错为 default 命名空间中的 test-nginx 工作负载出现冲突。



步骤 2 前往集群控制台或执行以下 kubectl 命令删除集群中的 test-nginx 工作负载。此处仅为示例，请根据实际报错信息进行删除。

```
kubectl delete deploy test-nginx -n default
```

步骤 3 解决资源冲突后，尝试重新安装模板。

----结束

11 API&kubectl

11.1 用户访问 CCE 集群的方式有哪些？

当前有两种访问集群的方式：

- CTAPI 方式

  CTAPI 采用 ak/sk 方式鉴权，需要天翼云帐号 ak/sk 信息进行认证后访问。
- 集群 API 方式

  集群 API 需要使用集群证书认证访问。

11.2 通过 API 或 kubectl 操作 CCE 集群，创建的资源是否能在控制台展示？

在 CCE 控制台，暂时不支持显示的 kubernetes 资源有：**DaemonSet**、**ReplicationController**、**ReplicaSets**、**Endpoints** 等。

若需要查询这些资源，请通过 kubectl 命令进行查询。

此外，Deployment、Statefulset、Service 和 Pod 资源需满足以下条件，才能在控制台显示：

- **Deployment 和 Statefulset：**标签中必须至少有一个标签是以"app"为 key 的。
- **Pod：**只有创建了无状态工作负载（Deployment）和有状态工作负载（StatefulSet）后，对应 Pod 实例才会在工作负载详情页的“实例列表”页签中显示。
- **Service：**Service 当前在无状态工作负载（Deployment）和有状态工作负载（StatefulSet）详情页的“访问方式”页签中显示。

  此处的显示需要 Service 与工作负载有一定的关联：

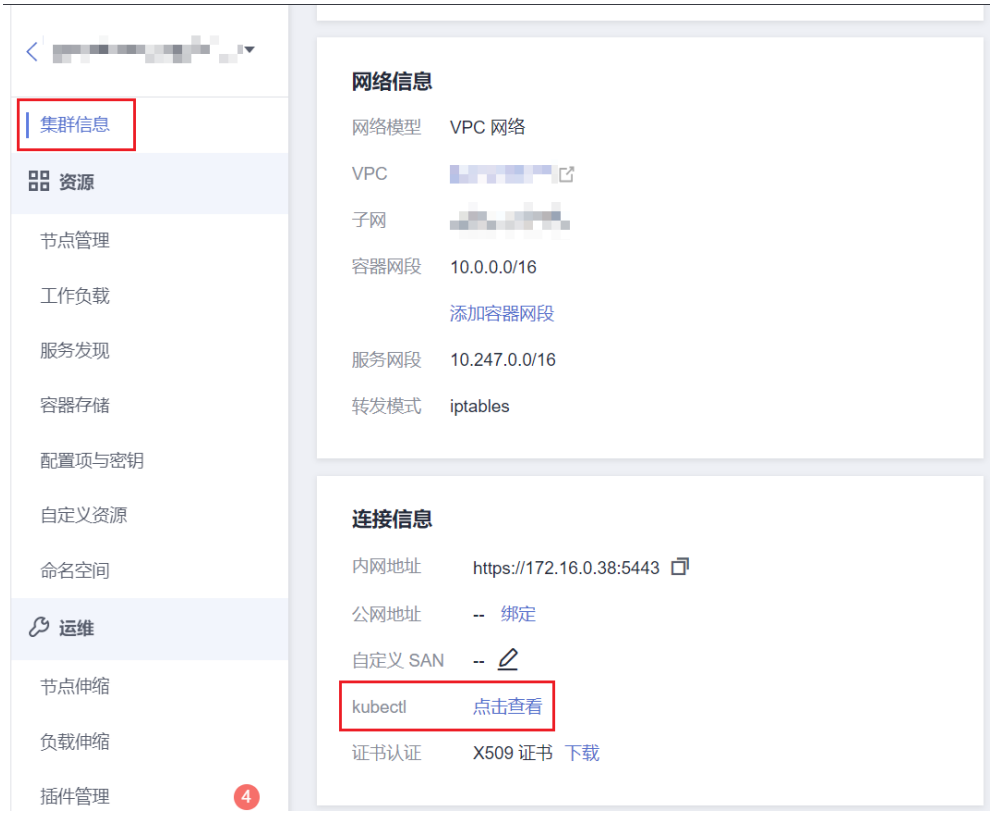
  a. 工作负载中的一个标签必须是以"app"为 key。

  b. Service 的标签和工作负载的标签保持一致。

11.3 通过 kubectl 连接集群时，其配置文件 config 如何下载？

步骤 1　登录 CCE 控制台，单击需要连接的集群名称，进入“集群信息”页面。

步骤 2　在“连接信息”版块中查看 kubectl 的连接方式。



步骤 3　在弹出的窗口中可以下载 kubectl 配置文件 kubeconfig.json。

图11-1 下载 kubeconfig.json

----结束

11.4 kubectl top node 命令为何报错

故障现象

执行 kubectl top node 命令报错 Error from server (ServiceUnavailable): the server is currently unable to handle the request (get nodes.metrics.k8s.io)

可能原因

执行 kubectl 时出现 Error from server (ServiceUnavailable)时，表示未能连接到集群，需要检查 kubectl 到集群 Master 节点的网络是否能够连通。

解决方法

- 如果是在集群外部执行 kubectl，请检查集群是否绑定公网 IP，如已绑定，请重新下载 kubeconfig 文件配置，然后重新执行 kubectl 命令。
- 如果是在集群内节点上执行 kubectl，请检查节点的安全组，是否放通 Node 节点与 Master 节点 TCP/UDP 互访，安全组的详细说明请参见集群安全组规则配置

11.5 kubectl 使用报错：Error from server (Forbidden)

故障现象

使用 kubectl 在创建或查询 Kubernetes 资源时，显示如下内容。

```
# kubectl get deploy Error from server (Forbidden): deployments.apps is forbidden: User "0c97ac3cb280f4d91fa7c0096739e1f8" cannot list resource "deployments" in API group "apps" in the namespace "default"
```

问题根因

使用户没有操作该 Kubernetes 资源的权限。

解决方法

使给该用户授权 Kubernetes 权限，具体方法如下。

步骤 1 登录 CCE 控制台，在左侧导航栏中选择“权限管理”。

步骤 2 在右边下拉列表中选择要添加权限的集群。

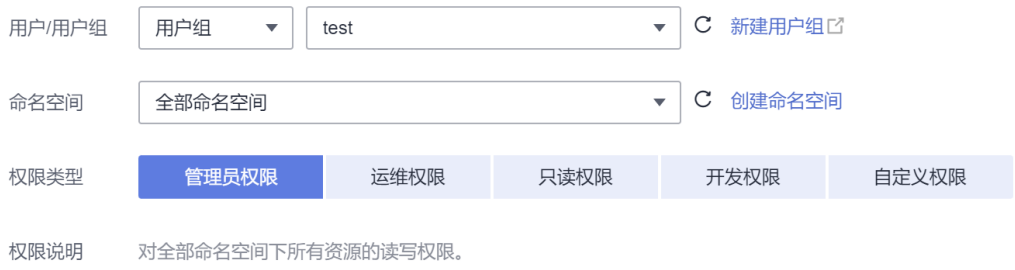
步骤 3 在右上角单击“添加权限”，进入添加授权页面。

步骤 4 在添加权限页面，确认集群名称，选择该集群下要授权使用的命名空间，例如选择“全部命名空间”，选择要授权的用户或用户组，再选择具体权限。

说明

对于没有 IAM 权限的用户，给其他用户和用户组配置权限时，无法选择用户和用户组，此时支持填写用户 ID 或用户组 ID 进行配置。

图11-2 配置命名空间权限



其中自定义权限可以根据需要自定义，选择自定义权限后，在自定义权限一行右侧单击新建自定义权限，在弹出的窗口中填写名称并选择规则。创建完成后，在添加权限的自定义权限下拉框中可以选择。

图11-3 自定义权限

步骤 5　单击“确定”。

12 域名 DNS

12.1 域名解析失败，如何定位处理？

排查项一：检查是否已安装 CoreDNS 插件

步骤 1 登录 CCE 控制台，进入集群。

步骤 2 在左侧导航栏中选择“插件管理”，在“已安装插件”中确认异常的集群是否已安装 CoreDNS 插件。

步骤 3 如果未安装，请安装。详情请参见为什么 CCE 集群的容器无法通过 DNS 解析？

----结束

排查项二：检查 CoreDNS 实例是否已到达性能瓶颈

CoreDNS 所能提供的域名解析 QPS 与 CPU 消耗成正相关，如遇 QPS 较高的场景，需要根据 QPS 的量级调整 CoreDNS 实例规格。集群超过 100 节点时，推荐使用 NodeLocal DNSCache 提升 DNS 性能，详情请参见用户指南 > 网络管理 > DNS。

步骤 1 登录 CCE 控制台，进入集群。

步骤 2 在左侧导航栏中选择“插件管理”，在“已安装插件”中找到集群对应的 CoreDNS 插件，确认插件状态为“运行中”。

步骤 3 单击 CoreDNS 插件名称，查看插件实例列表。

步骤 4 单击 CoreDNS 实例的“监控”按钮，查看实例 CPU、内存使用率。

如实例已达性能瓶颈，则需调整 CoreDNS 插件规格。

1. 在“已安装插件”下，单击 CoreDNS 插件的“编辑”按钮，进入插件详情页。
2. 在“规格配置”下配置 CoreDNS 参数规格。您可以根据业务需求选择 CoreDNS 所能提供的域名解析 QPS。
3. 您也可以选择自定义 QPS，通过选择不同的实例数、CPU 配额和内存配额，来定制集群的 CoreDNS 参数规格。

4. 单击“确定”，完成配置下发。

----结束

排查项三：解析外部域名很慢或超时

如果域名解析失败率低于 1/10000，请参考解析外部域名很慢或超时，如何优化配置？进行参数优化，或在业务中增加重试。

排查项四：概率性出现 UnknownHostException

集群中的业务请求到外部域名服务器时发生域名解析错误，概率性出现 UnknownHostException。UnknownHostException 是一个常见的异常，发生该异常时优先检查域名是否存在问题或键入错误。

您可根据以下步骤进行排查：

步骤 1 仔细检查主机名是否正确，检查域名的拼写并删除多余的空格。

步骤 2 检查 DNS 设置。在运行应用程序之前，通过 **ping hostname** 命令确保 DNS 服务器已启动并正在运行。如果主机名是新的，则需要等待一段时间才能访问 DNS 服务器。

步骤 3 检查 CoreDNS 实例的 CPU、内存使用率监控，确认是否已到达性能瓶颈，具体操作步骤请参见排查项二：检查 CoreDNS 实例是否已到达性能瓶颈。

步骤 4 检查 CoreDNS 是否有发生限流，如果触发限流可能出现部分请求处理时间延长，需要调整 CoreDNS 插件规格。

登录 CoreDNS Pod 所在节点，查看以下文件内容：

```
cat /sys/fs/cgroup/cpu/kubepods/pod{pod uid}/{coredns 容器 id}/cpu.stat
```

- *{pod uid}*为 CoreDNS 的 Pod UID，可通过以下命令获取：

  ```
  kubectl get po {pod name} -nkube-system -ojsonpath='{.metadata.uid}{"\n"}'
  ```

  以上命令中的*{pod name}*需要是在当前节点上运行的 CoreDNS Pod 名称。

- *{coredns 容器 id}*需要是完整的容器 ID，可通过以下命令获取：

  ```
  docker ps --no-trunc | grep k8s_coredns | awk '{print $1}'
  ```

完整的命令示例如下：

```
cat /sys/fs/cgroup/cpu/kubepods/pod27f58662-3979-448e-8f57-09b62bd24ea6/6aa98c323f43d689ac47190bc84cf4fadd23bd8dd25307f773df25003ef0eef0/cpu.stat
```

请关注以下指标：

- nr_throttled：被限流次数。
- throttled_time：被限流的总时间长度（纳秒）。

----结束

如果检查后无上述问题，可采用下方优化策略。

优化策略：

1. 修改 CoreDNS 的缓存时间
2. 配置存根域
3. 修改 ndots

说明

- **增加 coredns 的缓存时间**：有利于同一个域名的第 N 次解析，较少级联 DNS 的请求数量。
- **配置存根域**：有利于减少 DNS 请求链路。

修改方式：

1. 修改 CoreDNS 缓存时间及配置存根域
   修改方法请参见为 CoreDNS 配置存根域。
   修改完成后重启 CoreDNS。
2. 修改 ndots
   修改方法请参见解析外部域名很慢或超时，如何优化配置？。
   示例：

```
    dnsConfig:
      options:
        - name: timeout
          value: '2'
        - name: ndots
          value: '5'
        - name: single-request-reopen
```

   建议值修改成：2

12.2 为什么 CCE 集群的容器无法通过 DNS 解析？

问题描述

某客户在 DNS 服务中做内网解析，将自有的域名绑定到 DNS 服务中的内网域名中，并绑定到特定的 VPC 中，发现本 VPC 内的节点（ECS）可以正常解析内网域名的记录，而 VPC 内的容器则无法解析。

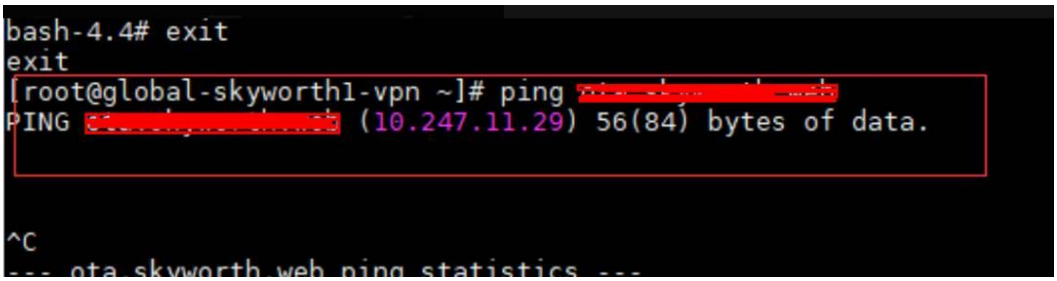
适用场景

VPC 内的容器无法进行正常 DNS 解析的情况。

解决方案

由于本案例涉及的是内网域名解析，按照内网域名解析的规则，需要确保 VPC 内的子网 DNS 设置成的云上 DNS，具体以内网 DNS 服务的控制台界面提示为准。

本案例的子网中已经完成该设置，其中用户可以在该 VPC 子网内的节点（ECS）进行域名解析，也说明已完成该设置，如下图：



但是在容器内进行解析却提示 bad address 无法解析域名返回地址，如下图：

登录 CCE 控制台查看该集群的插件安装情况。

如果已安装插件列表中没有 coredns 插件，可能是用户卸载了该插件等原因导致。

安装 coredns 插件，并添加相应的域名及对应的 DNS 服务地址，即可进行域名解析。

12.3 为什么修改子网 DNS 配置后，无法解析租户区域名？

问题描述

用户集群子网 DNS 配置，增加了 DNS 服务器配置，如 114.114.114.114，该域名无法解析租户区域名。

根因分析

CCE 会将用户的子网 DNS 信息配置到 node 节点上，coredns 插件中也是使用该配置信息，因此会导致用户在节点容器内解析域名会偶发失败的状况。

解决方案

建议您通过修改 coredns 插件的存根域更新用户集群子网 DNS 配置，修改方法请参见用户指南 > 网络管理 > DNS。

12.4 解析外部域名很慢或超时，如何优化配置？

工作负载的容器内的 resolv.conf 文件，示例如下：

```
root@test-5dffdddf95-vpt4m:/# cat /etc/resolv.conf
nameserver 10.247.3.10
search istio.svc.cluster.local svc.cluster.local cluster.local
options ndots:5 single-request-reopen timeout:2
```

其中：

- **nameserver**：DNS 服务器的 IP 地址，此处为 coredns 的 ClusterIP。
- **search**：域名的搜索列表，此处为 Kubernetes 的常用后缀。
- **ndots**：“.”的个数小于它的域名，会优先使用 search 进行解析。
- **timeout**：超时时间。
- **single-request-reopen**：发送 A 类型请求和 AAAA 类型请求使用不同的源端口。

在界面创建工作负载时，以上几项配置默认都会创建，具体参数如下：

```
dnsConfig:
  options:
    - name: timeout
      value: '2'
    - name: ndots
      value: '5'
    - name: single-request-reopen
```

以上参数可以根据业务需要进行优化或修改。

场景一：解析外部域名慢

优化方案：

1. 如果此工作负载不需要访问集群内的 k8s 服务，可以参考如何设置容器内的 DNS 策略？。
2. 如果此工作服务访问其他的 k8s 服务时，使用的域名中“.”的个数小于 2，可以将 ndots 参数设置为 2。

场景二：解析外部域名超时

优化方案：

1. 通常业务内的超时时间要大于 timeout * attempts 的时间。
2. 如果解析此域名通常要超过 2s，可以将 timeout 改大。

12.5 如何设置容器内的 DNS 策略？

CCE 支持通过 dnsPolicy 标记每个 Pod 配置不同的 DNS 策略：

- **None**：表示空的 DNS 设置，这种方式一般用于想要自定义 DNS 配置的场景，而且，往往需要和 dnsConfig 配合一起使用达到自定义 DNS 的目的。

- **Default：**有人说 Default 的方式，是使用宿主机的方式，这种说法并不准确。

  这种方式其实是让 kubelet 来决定使用何种 DNS 策略。而 kubelet 默认的方式，就是使用宿主机的 /etc/resolv.conf，但是 kubelet 是可以灵活来配置使用什么文件来进行 DNS 策略的，我们完全可以使用 kubelet 的参数：–resolv-conf=/etc/resolv.conf 来决定您的 DNS 解析文件地址。

- **ClusterFirst：**这种方式表示 Pod 内的 DNS 使用集群中配置的 DNS 服务，简单来说，就是使用 Kubernetes 中 kubedns 或 coredns 服务进行域名解析。如果解析不成功，才会使用宿主机的 DNS 配置进行解析。

如果未明确指定 dnsPolicy，则默认使用“ClusterFirst”：

- 如果将 dnsPolicy 设置为“Default”，则名称解析配置将从运行 pod 的工作节点继承。
- 如果将 dnsPolicy 设置为“ClusterFirst”，则 DNS 查询将发送到 kube-dns 服务。

  对于以配置的集群域后缀为根的域的查询将由 kube-dns 服务应答。所有其他查询（例如，www.kubernetes.io）将被转发到从节点继承的上游名称服务器。在此功能之前，通常通过使用自定义解析程序替换上游 DNS 来引入存根域。但是，这导致自定义解析程序本身成为 DNS 解析的关键路径，其中可伸缩性和可用性问题可能导致集群丢失 DNS 功能。此特性允许用户在不接管整个解析路径的情况下引入自定义解析。

如果某个工作负载不需要使用集群内的 coredns，可以使用 kubectl 命令或 API 将此策略设置为 dnsPolicy: Default。

13 权限

13.1 能否只配置命名空间权限，不配置集群管理权限？

命名空间权限和集群管理权限是相互独立又相互补充的两个权限体系：

- 命名空间权限：作用于集群内部，用于管理集群资源操作（如创建工作负载等）。
- 集群管理（IAM）权限：云服务层面的权限，用于管理 CCE 集群与周边资源（如 VPC、ELB、ECS 等）的操作。

对于 IAM Admin 用户组的管理员用户来说，可以为 IAM 子用户授予集群管理权限（如 CCE Administrator、CCE FullAccess 等），也可以在 CCE 控制台授予某个集群的命名空间权限。但由于 CCE 控制台界面权限是由 IAM 系统策略进行判断，如果 IAM 子用户未配置集群管理（IAM）权限，该子用户将无法进入 CCE 控制台。

如果您无需使用 CCE 控制台，只使用 kubectl 命令操作集群中的资源，则不受集群管理（IAM）权限的影响，您只需要获取具有命名空间权限的配置文件（kubeconfig），详情请参考如果不配置集群管理权限，是否可以使用 kubectl 命令呢？。集群配置文件在传递过程中可能存在泄露风险，应在实际使用中注意。

13.2 如果不配置集群管理权限的情况下，是否可以使用 API 呢？

CCE 提供的 API 可以分为云服务接口和集群接口：

- 云服务接口：支持操作云服务层面的基础设施（如创建节点），也可以调用集群层面的资源（如创建工作负载）。

  使用云服务接口时，必须配置集群管理（IAM）权限。
- 集群接口：直接通过 Kubernetes 原生 API Server 来调用集群层面的资源（如创建工作负载），但不支持操作云服务层面的基础设施（如创建节点）。

  使用集群接口时，无需配置集群管理（IAM）权限，仅需在调用集群接口时带上集群证书。但是，集群证书需要有集群管理（IAM）权限的用户进行下载，在证书传递过程中可能存在泄露风险，应在实际使用中注意。

13.3 如果不配置集群管理权限，是否可以使用 kubectl 命令呢？

使用 kubectl 命令无需经过 IAM 认证，因此理论上不配置集群管理（IAM）权限是可以使用 kubectl 命令的。但前提是需要获取具有命名空间权限的 kubectl 配置文件（kubeconfig），以下场景认证文件传递过程中均存在安全泄露风险，应在实际使用中注意。

- 场景一

  如果某 IAM 子用户先配置了集群管理权限和命名空间权限，然后在界面下载 kubeconfig 认证文件。后面再删除集群管理权限（保留命名空间权限），依然可以使用 kubectl 来操作 Kubernetes 集群。因此如需彻底删除用户权限，必须同时删除该用户的集群管理权限和命名空间权限。

- 场景二

  如果某 IAM 用户拥有一定范围的集群管理权限和命名空间权限，然后在界面下载 kubeconfig 认证文件。此时 CCE 根据用户信息的权限判断 kubectl 有权限访问哪些 Kubernetes 资源，即哪个用户获取的 kubeconfig 文件，kubeconfig 中就拥有哪个用户的认证信息，任何人都可以通过这个 kubeconfig 文件访问集群。

14 参考知识

14.1 如何扩容容器的存储空间？

使用场景

容器默认大小为 10G，当容器中产生数据较多时，容易导致容器存储空间不足，可以通过此方法来扩容。

解决方案

步骤 1 登录 CCE 控制台，单击集群列表中的集群名称。

步骤 2 在左侧导航栏中选择“节点管理”。

步骤 3 选择集群中的节点，单击操作列中的“更多 > 重置节点”。

须知

重置节点操作可能导致与节点有绑定关系的资源（本地存储，指定调度节点的负载等）无法正常使用。请谨慎操作，避免对运行中的业务造成影响。

步骤 4 在确认页面中单击“是”。

步骤 5 重新配置节点参数。

如需对容器存储空间进行调整，请重点关注以下配置。

存储配置：单击数据盘后方的“展开高级设置”可进行如下设置：

- 自定义容器引擎空间大小：容器引擎占用的存储空间，默认为数据盘空间的90%，用于存放容器引擎 (Docker/Containerd) 工作目录、容器镜像的数据和镜像元数据。
- 自定义容器 Pod 空间大小：CCE 支持对每个工作负载下的容器组 Pod 占用的磁盘空间设置上限（包含容器镜像占用的空间）。合理的配置可避免容器组无节制使用磁盘空间导致业务异常。建议此值不超过容器引擎空间的 80%。

说明

- 自定义容器 Pod 存储空间的能力与节点操作系统与容器存储 Rootfs 有关，设置规则如下：
- 容器存储 Rootfs 使用 DeviceMapper 时，节点支持自定义容器 Pod 空间设置（basesize），单个容器存储空间大小默认为 10GiB，可以配置为其他值。
- 容器存储 Rootfs 使用 OverlayFS 时，大部分节点不支持自定义容器 Pod 空间设置（basesize），默认为不限制，即单个容器存储空间大小默认为容器引擎空间。

  仅 1.19.16 版本、1.21.3 版本、1.23.3 版本及之后版本集群中的 EulerOS 2.9 系统节点支持自定义容器 Pod 空间设置（basesize），可以配置为其他值。
- 使用 EulerOS 2.9 的 docker basesize 设置时，若容器配置 CAP_SYS_RESOURCE 权限或 privileged 的特权，basesize 限制单容器数据空间不起作用。

更多关于容器存储空间分配的内容，请参考用户手册 > 节点管理 > 节点概述 > 数据盘空间分配说明。

步骤 6 重置节点后登录该节点，执行如下命令进入容器，查看 docker 容器容量是否已扩容。

docker exec -it *container_id* **/bin/sh** 或 **kubectl exec -it** *container_id* **/bin/sh**

df -h

```
# df -h
Filesystem                                                                                      Size  Used Avail Use% Mounted on
/dev/mapper/docker-253:1-787293-631c1bde2cbe82e39f32253b216ba914cb183b168b54700b3e5b9a54ee40a0d1   15G  229M   15G   2% /
tmpfs                                                                                           32G     0   32G   0% /dev
tmpfs                                                                                           32G     0   32G   0% /sys/fs/cgroup
/dev/mapper/vgpaas-kubernetes                                                                  9.8G   37M  9.2G   1% /etc/hosts
/dev/vda1                                                                                       40G  5.2G   33G  14% /etc/hostname
shm                                                                                             64M     0   64M   0% /dev/shm
tmpfs                                                                                           32G   16K   32G   1% /run/secrets/kubernetes.io/serviceaccount
tmpfs                                                                                           32G     0   32G   0% /proc/acpi
tmpfs                                                                                           32G     0   32G   0% /sys/firmware
tmpfs                                                                                           32G     0   32G   0% /proc/scsi
tmpfs                                                                                           32G     0   32G   0% /proc/kbox
tmpfs                                                                                           32G     0   32G   0% /proc/oom_extend
```

----结束

14.2 如何使容器重启后所在容器 IP 仍保持不变？

单节点场景

如果集群下仅有 1 个节点时，要使容器重启后所在容器 IP 保持不变，需在工作负载中配置主机网络，在工作负载的 yaml 中的 spec.spec.下加入 hostNetwork: true 字段。

多节点场景

如果集群下有多个节点时，除进行以上操作外，还需要设置节点的亲和策略，但工作负载创建后实例运行数不得超过亲和的节点数。

完成效果

进行如上设置并在工作负载运行后，工作负载实例 ip 与节点 ip 将保持一致，重启工作负载后 ip 也将保持不变。

14.3 云容器引擎 CCE 和微服务引擎的区别是什么？

对于使用者而言，云容器引擎关注的重点是 pod 的部署，微服务引擎关注的是服务的使用。

对于技术实现来看，微服务引擎是对云容器引擎的再一次封装。

基础概念

云容器引擎（CCE）

云容器引擎（Cloud Container Engine，简称 CCE）提供高度可扩展的、高性能的企业级 Kubernetes 集群，支持运行 Docker 容器，提供了 Kubernetes 集群管理、容器应用全生命周期管理、应用服务网格、Helm 应用模板、插件管理、应用调度、监控与运维等容器全栈能力，为您提供一站式容器平台服务。借助云容器引擎，您可以在天翼云上轻松部署、管理和扩展容器化应用程序。

应用管理与运维平台（ServiceStage）

ServiceStage 应用管理与运维平台是一个应用托管和微服务管理平台，可以帮助企业简化部署、监控、运维和治理等应用生命周期管理工作。ServiceStage 面向企业提供微服务、移动和 Web 类应用开发的全栈解决方案，帮助您的各类应用轻松上云，聚焦业务创新，帮助企业数字化快速转型。