容器服务

故障处理

腾讯云

文档目录

故障处理
在 Kubernetes 集群中如何根据无权限信息创建 RBAC 权限策略

最近更新时间：2024-07-10 10:35:01

操作场景

本文档介绍账号如何在 Kubernetes 集群中如何根据无权限信息创建 RBAC 权限策略，在特定集群中创建权限集合并绑定对应子账号，绑定后子账号将能管理集群下的资源。

报错示例

当子账号没有特定集群的 RBAC 权限并尝试获取资源时，将出现如下报错：

- 获取 Namespace 下的资源：

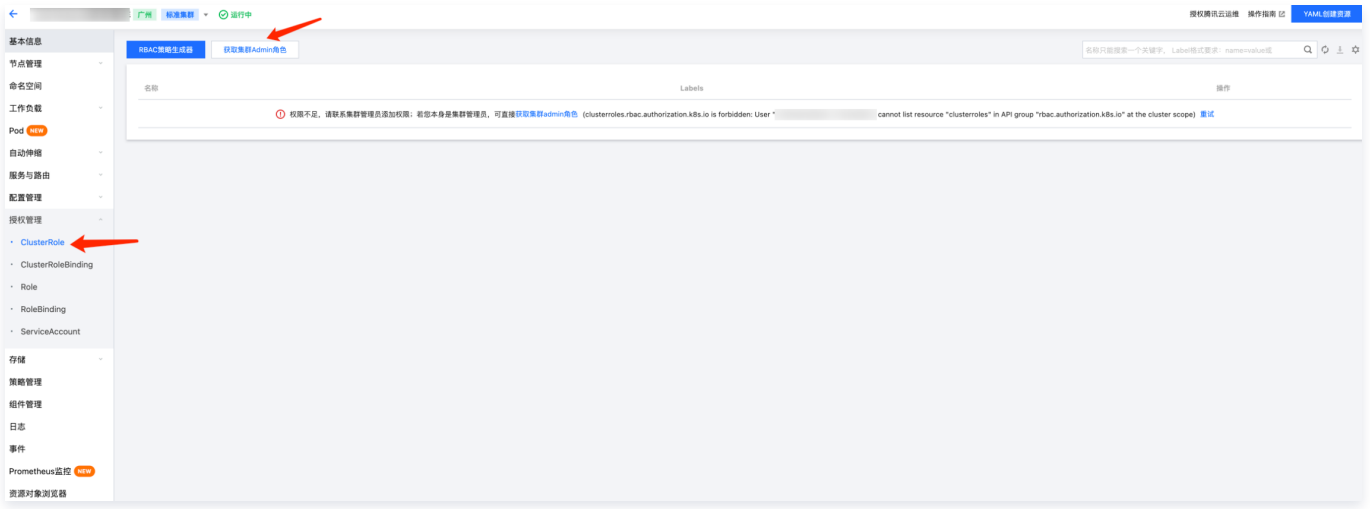
- 获取集群维度下的资源：

前提条件

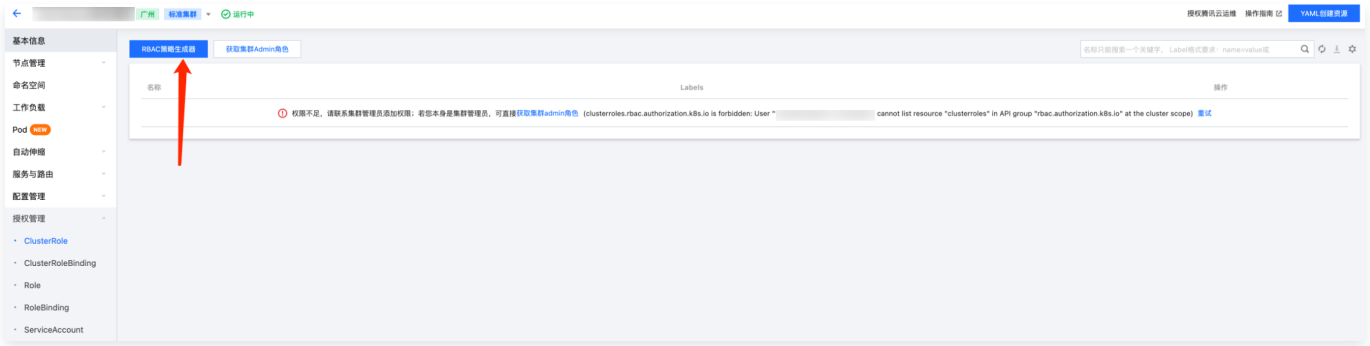
使用主账号或拥有该集群 RBAC admin 权限的子账号进行授权操作。

操作步骤

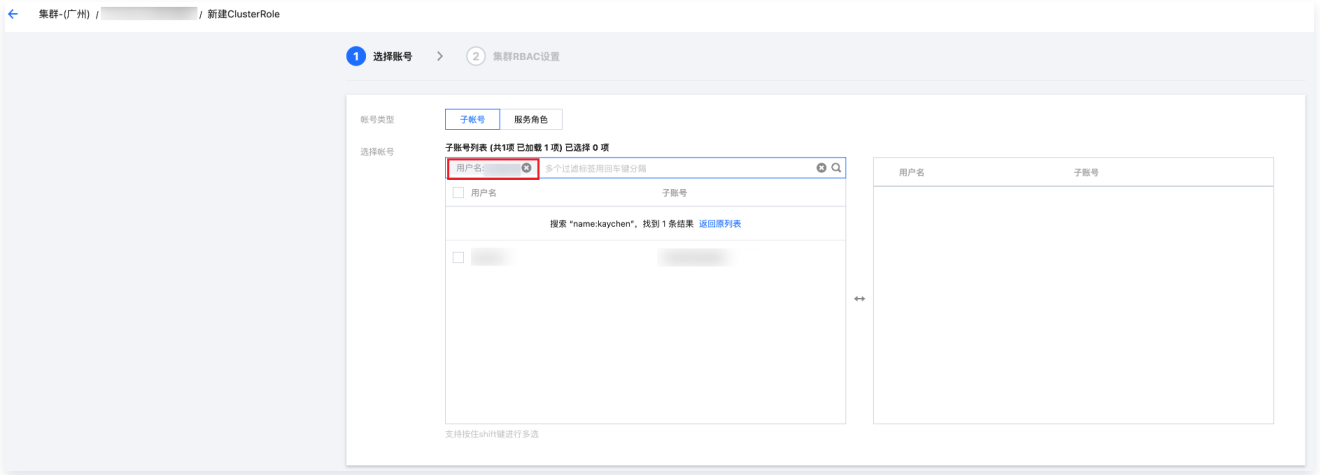
1. 登录 容器服务控制台，单击左侧导航栏中的**集群**。
2. 如果是主账号或者已被授权过 AcquireClusterAdminRole 接口的权限，但并没有该集群的 RBAC admin 权限，可以在**集群管理**页面，选择目标集群，进入集群详情页。在**授权管理 > ClusterRole** 中，通过**获取集群 Admin 角色**进行授权。如下图所示：



3. 在**授权管理 > ClusterRole** 中，单击 **RBAC 策略生成器**，选择子账户进行授权。如下图所示：



在**新建 ClusterRole** 页面，检索**对应子账号**，单击**下一步**。



4. 在**集群 RBAC 设置**中，给子账号授予权限。如下图所示：

- Namespace列表：选择是授予 Namespace 级别还是 Cluster 级别的权限。
- 权限：权限种类分别包含对集群不同范围的权限。
  - 管理员：对所有命名空间下资源的读写权限，拥有集群节点、存储卷、命名空间、配额的读写权限，可配置子账号和权限的读写权限。
  - 运维人员：对所有命名空间下资源的读写权限，拥有集群节点、存储卷、命名空间、配额的读写权限。
  - 开发人员：对所有命名空间或所选命名空间下控制台可见资源的读写权限。
  - 只读用户：对所有命名空间或所选命名空间下控制台可见资源的只读权限。
  - 自定义权限：由您所选择的 ClusterRole 决定，请在确定所选 ClusterRole 对各类资源的操作权限后再进行授权，以免子账号获得不符合预期的权限。

5. 单击**完成**，完成按策略生成器授予权限的操作。

权限示例

集群 Admin

```
apiVersion: "rbac.authorization.k8s.io/v1beta1"
kind: "ClusterRole"
metadata:
  name: "tke:admin"
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
rules:
-
  apiGroups:
  - "*"
  resources:
  - "*"
  verbs:
  - "*"
-
  nonResourceURLs:
  - "*"
  verbs:
  - "*"
```

集群运维管理人员

```
apiVersion: "rbac.authorization.k8s.io/v1beta1"
kind: "ClusterRole"
metadata:
  name: "tke:ops"
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
rules:
-
  apiGroups:
```

```
  - ""
  resources:
  - "pods"
  - "pods/attach"
  - "pods/exec"
  - "pods/portforward"
  - "pods/proxy"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "configmaps"
  - "endpoints"
  - "persistentvolumeclaims"
  - "replicationcontrollers"
  - "replicationcontrollers/scale"
  - "secrets"
  - "serviceaccounts"
  - "services"
  - "services/proxy"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "bindings"
  - "events"
  - "limitranges"
  - "namespaces/status"
  - "replicationcontrollers/status"
  - "pods/log"
  - "pods/status"
  - "resourcequotas"
  - "resourcequotas/status"
  - "componentstatuses"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "namespaces"
```

```
  - "nodes"
  - "persistentvolumes"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "apps"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "replicasets"
  - "replicasets/scale"
  - "statefulsets"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "autoscaling"
  resources:
  - "horizontalpodautoscalers"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "autoscaling.cloud.tencent.com"
  resources:
  - "horizontalpodcronscalers"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
```

```
  - "batch"
  resources:
  - "cronjobs"
  - "jobs"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "extensions"
  - "networking.k8s.io"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "ingresses"
  - "replicasets"
  - "replicasets/scale"
  - "replicationcontrollers/scale"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterserviceclasses"
  - "clusterserviceplans"
  - "clusterservicebrokers"
  - "serviceinstances"
  - "servicebindings"
  verbs:
  - "create"
  - "delete"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterservicebrokers/status"
  - "clusterserviceclasses/status"
  - "clusterserviceplans/status"
  - "serviceinstances/status"
  - "serviceinstances/reference"
```

```
  - "servicebindings/status"
  verbs:
  - "update"
-
  apiGroups:
  - "storage.k8s.io"
  resources:
  - "storageclasses"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "policy"
  resources:
  - "poddisruptionbudgets"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "networking.istio.io"
  - "config.istio.io"
  - "rbac.istio.io"
  - "authentication.istio.io"
  - "install.istio.io"
  - "security.istio.io"
  resources:
  - "*"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "apiextensions.k8s.io"
  resources:
  - "customresourcedefinitions"
  verbs:
  - "get"
  - "list"
  - "create"
  - "watch"
  - "patch"
```

```
  - "update"
  - "delete"
  - "deletecollection"
-
  apiGroups:
  - "networking.tke.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "create"
  - "watch"
  - "patch"
  - "update"
  - "delete"
  - "deletecollection"
-
  apiGroups:
  - "cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "create"
  - "watch"
  - "patch"
  - "update"
  - "delete"
  - "deletecollection"
-
  apiGroups:
  - "ccs.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "create"
  - "watch"
  - "patch"
  - "update"
  - "delete"
  - "deletecollection"
-
  apiGroups:
  - "cls.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "create"
  - "watch"
  - "patch"
  - "update"
  - "delete"
  - "deletecollection"
```

集群开发人员

```
---
apiVersion: "rbac.authorization.k8s.io/v1beta1"
kind: "ClusterRole"
metadata:
  name: "tke:dev"
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
rules:
-
  apiGroups:
  - ""
  resources:
  - "pods"
  - "pods/attach"
  - "pods/exec"
  - "pods/portforward"
  - "pods/proxy"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "configmaps"
  - "endpoints"
  - "persistentvolumeclaims"
  - "replicationcontrollers"
  - "replicationcontrollers/scale"
  - "secrets"
  - "serviceaccounts"
  - "services"
  - "services/proxy"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "events"
  - "replicationcontrollers/status"
  - "pods/log"
  - "pods/status"
  - "componentstatuses"
  verbs:
```

```
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "namespaces"
  - "nodes"
  - "persistentvolumes"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "apps"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "replicasets"
  - "replicasets/scale"
  - "statefulsets"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "autoscaling"
  resources:
  - "horizontalpodautoscalers"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "autoscaling.cloud.tencent.com"
  resources:
  - "horizontalpodcronscalers"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "batch"
  resources:
```

```
  - "cronjobs"
  - "jobs"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "extensions"
  - "networking.k8s.io"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "ingresses"
  - "replicasets"
  - "replicasets/scale"
  - "replicationcontrollers/scale"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterserviceclasses"
  - "clusterserviceplans"
  - "clusterservicebrokers"
  - "serviceinstances"
  - "servicebindings"
  verbs:
  - "create"
  - "delete"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterservicebrokers/status"
  - "clusterserviceclasses/status"
  - "clusterserviceplans/status"
  - "serviceinstances/status"
  - "serviceinstances/reference"
  - "servicebindings/status"
  verbs:
```

```
  - "update"
-
  apiGroups:
  - "storage.k8s.io"
  resources:
  - "storageclasses"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "policy"
  resources:
  - "poddisruptionbudgets"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "networking.istio.io"
  - "config.istio.io"
  - "rbac.istio.io"
  - "authentication.istio.io"
  - "security.istio.io"
  - "install.istio.io"
  resources:
  - "*"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "apiextensions.k8s.io"
  resources:
  - "customresourcedefinitions"
  verbs:
  - "get"
  - "list"
  - "create"
  - "watch"
  - "patch"
  - "update"
  - "delete"
  - "deletecollection"
-
  apiGroups:
  - "networking.tke.cloud.tencent.com"
  resources:
```

```
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "ccs.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "cls.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
```

集群指定Namespace开发人员

```
apiVersion: "rbac.authorization.k8s.io/v1beta1"
kind: "ClusterRole"
metadata:
  name: "tke:ns:dev"
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
rules:
-
  apiGroups:
  - ""
  resources:
  - "pods"
  - "pods/attach"
  - "pods/exec"
  - "pods/portforward"
  - "pods/proxy"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
```

```
-
  apiGroups:
  - ""
  resources:
  - "configmaps"
  - "endpoints"
  - "persistentvolumeclaims"
  - "replicationcontrollers"
  - "replicationcontrollers/scale"
  - "secrets"
  - "serviceaccounts"
  - "services"
  - "services/proxy"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "nodes"
  - "persistentvolumes"
  - "events"
  - "replicationcontrollers/status"
  - "pods/log"
  - "pods/status"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "apps"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "replicasets"
  - "replicasets/scale"
  - "statefulsets"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "autoscaling"
  resources:
  - "horizontalpodautoscalers"
```

```
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "autoscaling.cloud.tencent.com"
  resources:
  - "horizontalpodcronscalers"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "batch"
  resources:
  - "cronjobs"
  - "jobs"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "extensions"
  - "networking.k8s.io"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "ingresses"
  - "replicasets"
  - "replicasets/scale"
  - "replicationcontrollers/scale"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterserviceclasses"
  - "clusterserviceplans"
```

```
  - "clusterservicebrokers"
  - "serviceinstances"
  - "servicebindings"
  verbs:
  - "create"
  - "delete"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterservicebrokers/status"
  - "clusterserviceclasses/status"
  - "clusterserviceplans/status"
  - "serviceinstances/status"
  - "serviceinstances/reference"
  - "servicebindings/status"
  verbs:
  - "update"
-
  apiGroups:
  - "storage.k8s.io"
  resources:
  - "storageclasses"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "policy"
  resources:
  - "poddisruptionbudgets"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "networking.istio.io"
  - "config.istio.io"
  - "rbac.istio.io"
  - "authentication.istio.io"
  - "security.istio.io"
  - "install.istio.io"
  resources:
  - "*"
  verbs:
  - "create"
  - "delete"
  - "deletecollection"
  - "get"
```

```
  - "list"
  - "patch"
  - "update"
  - "watch"
-
  apiGroups:
  - "apiextensions.k8s.io"
  resources:
  - "customresourcedefinitions"
  verbs:
  - "get"
  - "list"
  - "create"
  - "watch"
  - "patch"
  - "update"
  - "delete"
  - "deletecollection"
-
  apiGroups:
  - "networking.tke.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "ccs.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "cls.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
```

集群只读人员

```
apiVersion: "rbac.authorization.k8s.io/v1beta1"
kind: "ClusterRole"
metadata:
```

```
  name: "tke:ro"
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
rules:
-
  apiGroups:
  - ""
  resources:
  - "pods"
  - "pods/attach"
  - "pods/exec"
  - "pods/portforward"
  - "pods/proxy"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "configmaps"
  - "endpoints"
  - "persistentvolumeclaims"
  - "replicationcontrollers"
  - "replicationcontrollers/scale"
  - "secrets"
  - "serviceaccounts"
  - "services"
  - "services/proxy"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "nodes"
  - "persistentvolumes"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "events"
  - "replicationcontrollers/status"
  - "pods/log"
  - "pods/status"
  - "componentstatuses"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "apps"
  resources:
```

```
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "replicasets"
  - "replicasets/scale"
  - "statefulsets"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "autoscaling"
  resources:
  - "horizontalpodautoscalers"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "autoscaling.cloud.tencent.com"
  resources:
  - "horizontalpodcronscalers"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "storage.k8s.io"
  resources:
  - "storageclasses"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "batch"
  resources:
  - "cronjobs"
  - "jobs"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "extensions"
  - "networking.k8s.io"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "ingresses"
  - "replicasets"
  - "replicasets/scale"
  - "replicationcontrollers/scale"
```

```
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterserviceclasses"
  - "clusterserviceplans"
  - "clusterservicebrokers"
  - "serviceinstances"
  - "servicebindings"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "policy"
  resources:
  - "poddisruptionbudgets"
  verbs:
  - "get"
  - "list"
-
  apiGroups:
  - "networking.istio.io"
  - "config.istio.io"
  - "rbac.istio.io"
  - "authentication.istio.io"
  - "security.istio.io"
  - "install.istio.io"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "apiextensions.k8s.io"
  resources:
  - "customresourcedefinitions"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "networking.tke.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "cloud.tencent.com"
  resources:
```

```
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "ccs.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "cls.cloud.tencent.com"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
```

集群指定Namespace只读人员

```
apiVersion: "rbac.authorization.k8s.io/v1beta1"
kind: "ClusterRole"
metadata:
  name: "tke:ns:ro"
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
rules:
-
  apiGroups:
  - ""
  resources:
  - "pods"
  - "pods/attach"
  - "pods/exec"
  - "pods/portforward"
  - "pods/proxy"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "nodes"
  - "persistentvolumes"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
```

```
  - "configmaps"
  - "endpoints"
  - "persistentvolumeclaims"
  - "replicationcontrollers"
  - "replicationcontrollers/scale"
  - "secrets"
  - "serviceaccounts"
  - "services"
  - "services/proxy"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - ""
  resources:
  - "events"
  - "replicationcontrollers/status"
  - "pods/log"
  - "pods/status"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "apps"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "replicasets"
  - "replicasets/scale"
  - "statefulsets"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "autoscaling"
  resources:
  - "horizontalpodautoscalers"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "autoscaling.cloud.tencent.com"
  resources:
  - "horizontalpodcronscalers"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "batch"
```

```
  resources:
  - "cronjobs"
  - "jobs"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "extensions"
  - "networking.k8s.io"
  resources:
  - "daemonsets"
  - "deployments"
  - "deployments/rollback"
  - "deployments/scale"
  - "ingresses"
  - "replicasets"
  - "replicasets/scale"
  - "replicationcontrollers/scale"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "servicecatalog.k8s.io"
  resources:
  - "clusterserviceclasses"
  - "clusterserviceplans"
  - "clusterservicebrokers"
  - "serviceinstances"
  - "servicebindings"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
  - "policy"
  resources:
  - "poddisruptionbudgets"
  verbs:
  - "get"
  - "list"
-
  apiGroups:
  - "networking.istio.io"
  - "config.istio.io"
  - "rbac.istio.io"
  - "authentication.istio.io"
  - "security.istio.io"
  - "install.istio.io"
  resources:
  - "*"
  verbs:
  - "get"
  - "list"
  - "watch"
-
  apiGroups:
```

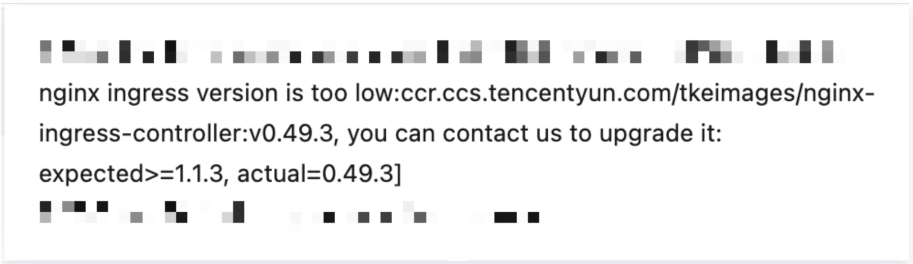
```
  - "apiextensions.k8s.io"
  resources:
  - "customresourcedefinitions"
  verbs:
  - "get"
  - "list"
  - "watch"
```

NginxIngress 版本过低导致集群无法升级到 1.22 以上

最近更新时间：2025-05-09 14:39:02

问题现象

nginx-ingress-controller 镜像版本过低会导致集群无法正常升级到 Kubernetes 1.22版本及以上，相关报错如下图所示：



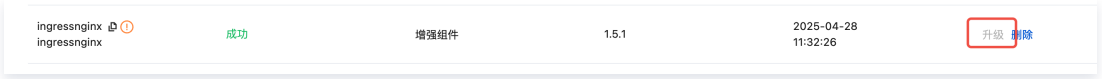
问题原因

由于 Kubernetes 1.22版本 APIServer 已经废弃 extensions/v1beta1 版本的 Ingress 资源，集群升级到1.22后，之前安装的 nginx-ingress-controller 组件镜像版本如果低于 v1.1.3，会导致 nginx-ingress-controller 组件无法正常工作。

解决方案

请按照以下步骤将低版本的 nginx-ingress-controller 组件升级到 v1.1.3 版本：

1. 登录 容器服务控制台，在左侧导航栏中选择**集群**。
2. 在集群列表中，单击目标集群 ID，进入集群详情页。
3. 选择左侧菜单栏中的**组件管理**，在组件管理页面将 ingress-nginx 插件升级到最新版本（若已是可以忽略）：



4. 编辑对应 nginx ingress controller 的 workload，修改对应 nginx ingress controller 的镜像 tag 为 v1.1.3：

```
spec:
  affinity: {}
  containers:
  - args:
    - /nginx-ingress-controller
    - --validating-webhook=:8443
    - --validating-webhook-certificate=/usr/local/certificates/cert
    - --validating-webhook-key=/usr/local/certificates/key
    - --election-id=ingress-controller-leader
    - --ingress-class=test
    - --configmap=kube-system/test-ingress-nginx-controller
    - --publish-service=kube-system/test-ingress-nginx-controller
    - --tcp-services-configmap=kube-system/test-ingress-nginx-tcp
    - --udp-services-configmap=kube-system/test-ingress-nginx-udp
    env:
    - name: POD_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.name
    - name: POD_NAMESPACE
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: metadata.namespace
    image: ccr.ccs.tencentyun.com/tkeimages/nginx-ingress-controller:v1.1.3
    imagePullPolicy: Always
    lifecycle:
      preStop:
        exec:
          command:
          - /wait-shutdown
```

5. 编辑对应 nginx ingress controller 的 workload，修改 `--election-id` 参数，添加 ingress-class 后缀，例如 `--ingress-class` 为 **test**，则修改 `--election-id` 参数 `ingress-controller-leader` 改为 `ingress-controller-leader-` **`test`**：

```
spec:
  affinity: {}
  containers:
  - args:
    - /nginx-ingress-controller
    - --validating-webhook=:8443
    - --validating-webhook-certificate=/usr/local/certificates/cert
    - --validating-webhook-key=/usr/local/certificates/key
    - --election-id=ingress-controller-leader-test
    - --ingress-class=test
    - --configmap=kube-system/test-ingress-nginx-controller
    - --publish-service=kube-system/test-ingress-nginx-controller
    - --tcp-services-configmap=kube-system/test-ingress-nginx-tcp
    - --udp-services-configmap=kube-system/test-ingress-nginx-udp
    env:
    - name: POD_NAME
```

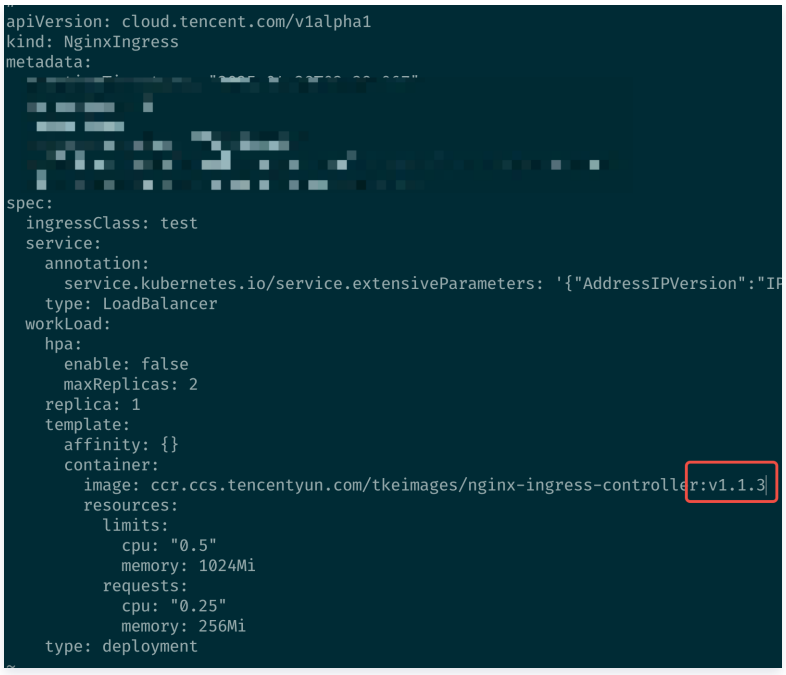
6. 修改 nginx-ingress-controller 对应的 ValidatingWebhookConfiguration 资源，将 `admissionReviewVersions` 和 `apiVersions` 修改为 `v1`，`path` 修改为 `/networking/v1/ingresses`：

```
kubectl edit validatingwebhookconfiguration {ingress-class}-ingress-nginx-admission
```

```
webhooks:
- admissionReviewVersions:
  - v1
  clientConfig:
    caBundle: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk
vWUR6SXhNalV3TkRBME1ETXpOREV3V2pBUApNUTB3Q3dZRFZRUUt
Uy9jdU1GMW5LYVRhRmYKUUhYT1RvVnFsR3U0U0tOWE1GVXdEZ1lE
Dh5Cm1GOHBNQW9HQ0NxR1NNNDlCQU1DQTBnQU1FVUNJUUN3VEhZd
NBVEUtLS0tLQo=
    service:
      name: test-ingress-nginx-controller-admission
      namespace: kube-system
      path: /networking/v1/ingresses
      port: 443
  failurePolicy: Fail
  matchPolicy: Equivalent
  name: validate.nginx.ingress.kubernetes.io
  namespaceSelector: {}
  objectSelector: {}
  rules:
  - apiGroups:
    - networking.k8s.io
    apiVersions:
    - v1
    operations:
    - CREATE
    - UPDATE
    resources:
    - ingresses
    scope: '*'
```

7. 将 NginxIngress CR 资源中的镜像 tag 修改为 v1.1.3：

```
kubectl edit nginxingress {ingress-class}
```


apiVersion: cloud.tencent.com/v1alpha1
kind: NginxIngress
metadata:
spec:
ingressClass: test
service:
annotation:
service.kubernetes.io/service.extensiveParameters: '{"AddressIPVersion":"IP
type: LoadBalancer
workLoad:
hpa:
enable: false
maxReplicas: 2
replica: 1
template:
affinity: {}
container:
image: ccr.ccs.tencentyun.com/tkeimages/nginx-ingress-controller:v1.1.3
resources:
limits:
cpu: "0.5"
memory: 1024Mi
requests:
cpu: "0.25"
memory: 256Mi
type: deployment

节点常见报错与处理

最近更新时间：2024-03-25 15:41:11

节点异常关键字排障

当使用 TKE 集群服务的节点检查能力时，可能会检查出节点异常，包含建议您进一步排查的关键字，本文档总结了节点检查中出现的关键字、异常现象及对应的检查方式（通过正则表达式匹配内核 / dev / dmesg 日志进行检查），您可以根据本文档登录节点进一步排查异常原因。

节点异常关键字含义及检测方式

关键字	含义	匹配用的正则表达式
OOMKilling	进程 OOM killing	Killed process \d+ (.+) total-vm:\d+kB, anon-rss:\d+kB, file-rss:\d+kB.*
TaskHung	进程长时间冻结（IO、NFS 等问题）	task [\S]+:\w+ blocked for more than \w+ seconds\.
UnregisterNetDevice	网络设备泄露，如存在没有注册的网络设备	unregister_netdevice: waiting for \w+ to become free. Usage count = \d+
KernelOops	内核出现空指针	BUG: unable to handle kernel NULL pointer dereference at .*
KernelOops	除0错误	divide error: 0000 [#\d+] SMP
Ext4Error	Ext4 文件系统故障	EXT4-fs error .*
Ext4Warning	Ext4 文件系统警告	EXT4-fs warning .*
IOError	Buffer 写入设备阻塞	Buffer I/O error .*
MemoryReadError	内存故障	CE memory read error .*
FilesystemIsReadOnly	文件系统只读，无法写入	Remounting filesystem read-only
TCPMemOverFlow	TCP 内存不足	TCP: out of memory -- consider tuning tcp_mem
TCPSkOverFlow	Socket 太多引发	TCP: too many orphaned sockets
NFOverFlow	conntrack 表满	nf_conntrack: table full, dropping packet
ARPOverFlow	arp 表满	\w+: neighbor table overflow!
BlockIOError	Buffer 写入设备阻塞	Buffer I/O error on device (.+), logical block \d+
BlockIOError	Blkio 请求阻塞	blk_update_request: I/O error, dev \w+, sector \d+
FileOpenLimit	打开文件超过系统上限	VFS: file-max limit \d+ reached
SlabFreeErr	释放 Slab 错误	cache_from_obj: Wrong slab cache. (.+) but object is from (.+)
MemPageFailed	Page 分配错误	page allocation failure(.) order:[3-5],(.+)
SoftLockUp	CPU 长时间没有调度切换	BUG: soft lockup - CPU#\d+ stuck for (.+)
SchedInAtomic	占有自旋锁时调用 sleep	BUG: scheduling while atomic:(.+)
RCUStall	CPU RCU 长时间卡顿	INFO: \w+ self-detected stall on CPU (.+)
PCICardErr	PCI Card 错误，如虚拟设备	Card not present on Slot(.+)

CBS-CSI 常见报错和处理

最近更新时间：2024-12-12 09:57:22

cbs 盘创建相关问题

1. no available storage in zone

现象： kubectl describe pvc 发现类型为 ProvisioningFailed 的事件，内容包含 `no available storage in zone` 。
原因： 资源售罄或者该 zone 不支持这种类型的 cbs 盘。
解决措施： 用户可切换到有资源的 region/zone，或联系 cbs 售后提供资源。

2. disk is sold out

现象： cbs 插件报错 `disk is sold out` 。
原因： cbs 磁盘售罄。
解决措施： 联系 cbs 售后上架资源后重建 Pod。

3. InvalidParameter

现象1： pvc 一直处于 pending 状态，pv 无法被创建出来，插件报错 `Key start with a word that is reserved for the system` 。
原因： cbs 插件在创建云硬盘时，会将集群的云标签集成到云硬盘上。而当云标签 key 中带有 qcloud. tencent 或 project 字段时，会导致云硬盘创建失败。
解决措施： 修改相关集群云标签，避免在标签 key 中出现 qcloud. tencent 或 project 字段。

```
[TencentCloudSDKError] Code=InvalidParameter, Message=(16ab33025ebd)Key (tencentCloudPorject) start with a
word that is reserved for the system
```

现象2： pvc 一直处于 pending 状态，pv 无法被创建出来，插件报错 `tag value contains illegal characters` 。
原因： cbs 插件在创建云硬盘时，会将集群的云标签继承到云硬盘上。当云标签 value 为空值时，会导致云硬盘创建失败。
解决措施： 修改相关集群云标签，避免将标签 value 配置为空。

```
[TencentCloudSDKError] Code=InvalidParameter, Message=(ec82d0f8807c)tag value contains illegal characters.
Supports UTF-8-encoded characters, digits, spaces, and special characters (+-=._:/@()[]｛｝【】),
```

4. disk size is invalid

现象： describe pvc 或查看插件日志可以看到报错 `disk size is invalid` 。
原因： 用户配置的 cbs 盘大小不符合规范导致买盘失败，不同类型的 cbs 盘，支持大小区间不同，具体查看 云硬盘类型文档 。
解决措施： 修改 pvc 容量大小，使其处于 cbs 支持区间范围内。

5. WaitForFirstConsumer 挂载模式下创盘失败

现象： StorageClass 中选择 WaitForFirstConsumer 的挂载模式，在 workload 中配置了 nodeName 参数，上述情况下会创盘失败。
原因： WaitForFirstConsumer 挂载模式依赖调度器触发云硬盘创建，而指定了 nodeName 参数会导致 Pod 在调度时跳过调度器，从而无法通知插件进行云硬盘创建。
解决措施： 使用 nodeSelector 方式替换 nodeName 参数，或在 sc 中直接指定 Immediate 的挂载模式，如有需要也可以在 sc 参数中指定可用区。

6. 服务端返回 InternalError

现象： 插件报错 `InternalError` 。
原因： 一般为 cbs 服务端问题。
解决措施： 联系 cbs 售后确定是否 cbs 服务问题，待恢复后即可正常买盘。

```
[TencentCioudSDKError] Code=InternalError, Message=内部服务错误，请稍后重试。
```

7. 服务端返回 UnauthorizedOperation.NotHavePaymentRight

现象： 创建包年包月云硬盘失败，插件报错 `UnauthorizedOperation.NotHavePaymentRight` 。
原因： TKE_QCSRole 角色没有关联策略 QcloudCVMFinanceAccess。
解决措施： 请参考 指引文档 给角色关联策略添加权限。

cbs 盘 attach 相关问题

基本概念

attach：将 cbs 盘关联到 Pod 将被调度运行的 node 上面。

1. 黑石机器 attach 失败

原因：黑石机器默认不支持挂载 cbs 盘。

2. attach 云盘太慢（超过 10 分钟）

现象：pvc/pv 对象创建都比较及时，但是 volumeattachment 对象的创建延迟了6分钟甚至更近，导致 csi 组件 attach 也延迟。查看 kcm 日志，发现请求延迟很大。
原因：kcm 每一分钟就全量 Get 所有的 volumeattachment，当集群内 volumeattachment 对象数较多时会触发请求限频。
解决措施：让 kcm 全量 Get 所有 volumeattachment 的时间间隔变长，避免被限频，可申请修改 kcm 相关参数。

3. exceed max volume count

现象：业务 Pod 处于 Pending 状态，无法完成调度，Describe Pod 有告警 `node(s) exceed max volume count`
原因：单个 CVM 默认只支持 attach 20 块云硬盘，cbs 组件在除去系统盘和数据盘后，默认只支持再挂载 18 块云硬盘。

cbs 盘 mount 相关问题

1. 插件注册到 kubelet 失败

现象：Describe Pod 或在 kubelet 日志中有如下报错。

```
MountVolume.MountDevice, driver name com.tencent.cloud.csi.cbs not found in the list of registered CSI
drivers
```

原因：
1. 用户删除了集群中的 cbs 组件。
2. kubelet 的 rootdir 变更了，默认值为 /var/lib/kubelet。
3. csi-cbs-node 获取节点 instanceid 为空。

排查思路：
1. 登录集群，通过 `kubectl get pod -nkube-system |grep cbs` 查看是否已安装 cbs 组件。
2. 查看报错的 Pod 位置，登录节点，通过 `ps -ef |grep kubelet` 查看 kubelet 进程有无 rootdir 参数，有且不为默认值，则为原因 2。
3. csi-cbs-node driver-registrar 容器日志中查看 driverNodeID must not be empty 报错信息。

解决措施：
1. 控制台安装 cbs 组件。
2. 控制台修改组件配置，将 rootdir 参数指定为正确的值。
3. 控制台升级 cbs 组件。

2. /dev/vd* is already mounted

现象：业务 Pod 挂载 cbs 失败，插件报错如下信息。

```
mount failed: exit status 32
Mounting command: mount
Mounting arguments: -t ext4 -o defaults /dev/vd* /var/lib/kubelet/plugins/kubernetes.io/csi/pv/pvc-
***/globalmount
Output: mount: /dev/vd* is already mounted or /var/lib/kubelet/plugins/kubernetes.io/csi/pv/pvc-
***/globalmount busy
```

原因：旧版本组件 globalmount 目录存在重复挂载问题。
解决措施：控制台升级 cbs 组件。

3. mounting failed: Invalid argument

现象：业务 Pod 挂载 cbs 失败，插件报错如下信息。

```
mount failed: exit status 255
Mounting command: mount
Mounting arguments: -t ext4 -o defaults /dev/vd* /var/lib/kubelet/plugins/kubernetes.io/csi/pv/pvc-***/globalmount
Output: mount: mounting /dev/vd* on /var/lib/kubelet/plugins/kubernetes.io/csi/pv/pvc-***/globalmount failed: Invalid argument
```

原因：cbs 盘已格式化为 gpt 格式，无法通过 cbs 组件来挂载。

```
/ # blkid -p -s TYPE -s PTTYPE -o export /dev/vdk
DEVNAME=/dev/vdk
PTTYPE=gpt
/ # blkid -p -s TYPE -s PTTYPE -o export /dev/vdj
DEVNAME=/dev/vdj
PTTYPE=gpt
```

解决措施：将盘直接格式化为 ext4，或通过 pvc 新建未格式化的云盘，并将业务 pod 指向新建的 pvc。

4. 业务 Pod 启动时间较长

现象：挂载 cbs 的业务 Pod 启动时间较长，kubelet 包含如下日志。

```
Setting volume ownership for %s and fsGroup set. If the volume has a lot of files then setting volume ownership could be slow, see https://github.com/kubernetes/kubernetes/issues/69699
```

原因：业务负载中指定了 fsGroup 参数，导致 kubelet 在完成 cbs 挂载后，会把挂载目录下所有文件进行一次权限修改，修改为 fsGroup 指定权限。若挂载目录下文件很多，kubelet 存储准备工作就会长时间卡在权限修改这一步，直到挂载目录下所有文件权限修改完成。

解决措施：

方案一：不指定 fsgroup 参数，用户需自行确保权限匹配。

方案二：在 Pod template 中配置 `spec.securityContext.fsGroupChangePolicy` 参数为 `OnRootMismatch`，只要目录下文件权限已匹配就不会去刷权限（集群版本需不小于 1.20）。

节点磁盘爆满排障处理

最近更新时间：2024-11-13 11:32:21

本文档介绍 TKE 集群中多场景下可能发生的磁盘爆满问题，并给出对应的排查思路及解决方案，请按照下文中的步骤进行排查并解决。

可能原因

kubelet 支持 gc 和驱逐机制，可通过 `--image-gc-high-threshold`、`--image-gc-low-threshold`、`--eviction-hard`、`--eviction-soft` 及 `--eviction-minimum-reclaim` 等参数进行控制以实现磁盘空间的释放。当配置不正确，或者节点上有其他非 K8S 管理的进程在不断写数据到磁盘，将会占用大量空间时将导致磁盘爆满。
磁盘爆满将影响 K8S 运行，主要涉及 kubelet 和容器运行时两个关键组件。请按照以下步骤进行分析：
1. 执行 `df` 命令，查看 kubelet 和容器运行时所使用的目录是否存在于该磁盘。
2. 对应实际结果，选择以下方式进行问题精确定位：

- 容器运行时使用的目录所在磁盘爆满
- Kubelet 使用的目录所在磁盘爆满

问题定位及解决思路

容器运行时使用的目录所在磁盘爆满

如果容器运行时使用的目录所在磁盘空间爆满，将可能会造成容器运行时无响应。例如，当前容器运行时为 docker，则执行 docker 相关的命令将会一直 hang 住，kubelet 日志也将看到 PLEG unhealthy，而 CRI 调用 timeout 也将导致容器无法创建或销毁，外在现象通常表现为 Pod 一直 ContainerCreating 或一直 Terminating。

docker 默认使用的目录

- `/var/run/docker`：用于存储容器运行状态，通过 dockerd 的 `--exec-root` 参数指定。
- `/var/lib/docker`：用于持久化容器相关的数据。例如，容器镜像、容器可写层数据、容器标准日志输出及通过 docker 创建的 volume 等。

Pod 启动过程事件

Pod 在启动过程中，可能会出现以下类似事件：

```
Warning  FailedCreatePodSandBox 53m kubelet, 172.22.0.44  Failed create pod sandbox: rpc error: code =
DeadlineExceeded desc = context deadline exceeded
```

```
Warning  FailedCreatePodSandBox  2m (x4307 over 16h)  kubelet, 10.179.80.31  (combined from similar
events): Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod
"apigateway-6dc48bf8b6-l8xrw": Error response from daemon: mkdir
/var/lib/docker/aufs/mnt/1f09d6c1c9f24e8daaea5bf33a4230de7dbc758e3b22785e8ee21e3e3d921214-init: no space
left on device
```

```
Warning  Failed   5m1s (x3397 over 17h)  kubelet, ip-10-0-151-35.us-west-2.compute.internal  (combined
from similar events): Error: container create failed: container_linux.go:336: starting container process
caused "process_linux.go:399: container init caused \"rootfs_linux.go:58: mounting \\\"/sys\\\" to rootfs
\\\"/var/lib/dockerd/storage/overlay/051e985771cc69f3f699895a1dada9ef6483e912b46a99e004af7bb4852183eb/merg
ed\\\" at
\\\"/var/lib/dockerd/storage/overlay/051e985771cc69f3f699895a1dada9ef6483e912b46a99e004af7bb4852183eb/merg
ed/sys\\\" caused \\\"no space left on device\\\"\""
```

Pod 删除过程事件

Pod 在删除过程中，可能会出现以下类似事件：

```
Normal  Killing  39s (x735 over 15h)  kubelet, 10.179.80.31  Killing container with id
docker://apigateway:Need to kill Pod
```

Kubelet 使用的目录所在磁盘爆满

kubelet 默认使用的目录

`/var/lib/kubelet`：通过 kubelet 的 `--root-dir` 参数指定，用于存储插件信息、Pod 相关的状态以及挂载的 volume（例如，`emptyDir`、`ConfigMap` 及 `Secret`）。

Pod 事件

Kubelet 使用的目录所在磁盘空间爆满（通常是系统盘），新建 Pod 时无法成功进行 mkdir，导致 Sandbox 也无法创建成功，Pod 通常会出现以下类似事件：

```
  Warning  UnexpectedAdmissionError  44m kubelet, 172.22.0.44  Update plugin resources failed due to
failed to write checkpoint file "kubelet_internal_checkpoint": write /var/lib/kubelet/device-
plugins/.728425055: no space left on device, which is unexpected.
```

处理步骤

当容器运行时为 docker 时发生磁盘爆满问题，dockerd 也会因此无法正常响应，在停止时会卡住，从而导致无法直接重启 dockerd 来释放空间。需要先手动清理部分文件腾出空间以确保 dockerd 能够停止并重启。恢复步骤如下：

1. 手动删除 docker 的部分 log 文件或可写层文件。通常删除 log 文件，示例如下：

```
$ cd /var/lib/docker/containers
$ du -sh * # 找到比较大的目录
$ cd dda02c9a7491fa797ab730c1568ba06cba74cecd4e4a82e9d90d00fa11de743c
$ cat /dev/null > dda02c9a7491fa797ab730c1568ba06cba74cecd4e4a82e9d90d00fa11de743c-json.log.9 # 删除log
文件
```

> **注意：**
> - 删除文件时，建议使用 `cat /dev/null >` 方式进行删除，不建议使用 `rm`。使用 `rm` 方式删除的文件，不能够被 docker 进程释放掉，该文件所占用的空间也就不会被释放。
> - log 的后缀数字越大表示时间越久远，建议优先删除旧日志。

2. 执行以下命令，将该 Node 标记为不可调度，并将其已有的 Pod 驱逐到其它节点。

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

该步骤可以确保 dockerd 重启时将原节点上 Pod 对应的容器删掉，同时确保容器相关的日志（标准输出）与容器内产生的数据文件（未挂载 volume 及可写层）也会被清理。

3. 执行以下命令，重启 dockerd。

```
systemctl restart dockerd
# or systemctl restart docker
```

4. 等待 dockerd 重启恢复，Pod 调度到其它节点后，排查磁盘爆满原因并进行数据清理和规避操作。
5. 执行以下命令，取消节点不可调度标记。

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

如何规避磁盘爆满？

需确保 kubelet 的 gc 和驱逐相关参数进行配置正确。若配置无问题，即便磁盘达到爆满地步，此时事发节点上的 Pod 也已自动驱逐到其它节点上，不会出现 Pod 一直 ContainerCreating 或 Terminating 的问题。

节点高负载排障处理

最近更新时间：2024-11-13 11:32:21

本文档介绍如何在 TKE 集群中，通过工具定位异常是否由高负载造成，请按照以下步骤进行问题排查。

现象描述

节点高负载将会导致进程无法获得足够运行所需的 CPU 时间片，通常表现为网络 Timeout、健康检查失败或服务不可用。

问题定位及解决思路

有时节点在低 cpu ‘us’（user）、高 cpu ‘id’（idle）的条件下，仍会出现负载很高的情况。通常是由于文件 IO 性能达到瓶颈，导致 IO Wait 过多，使节点整体负载升高，影响其它进程的性能。
本文以 top、atop 及 iotop 工具为例，来判断磁盘 I/O 是否正在降低应用性能。

查看平均负载及等待时间

1. 登录节点，执行 `top` 命令查看当前负载。返回结果如下：

> **说明：**
> 可通过较高的 `load average` 值得知该节点正在承接大量的请求，也可以通过 `Cpu(s)`、`Mem`、`%CPU` 及 `%MEM` 列的数据获取哪些进程正在占用大多数资源。

```
top - 19:42:06 up 23:59,  2 users,  load average: 34.64, 35.80, 35.76
Tasks: 679 total,   1 running, 678 sleeping,   0 stopped,   0 zombie
Cpu(s): 15.6%us,  1.7%sy,  0.0%ni, 74.7%id,  7.9%wa,  0.0%hi,  0.1%si,  0.0%st
Mem:  32865032k total, 30989168k used,  1875864k free,   370748k buffers
Swap:  8388604k total,     5440k used,  8383164k free,  7982424k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 9783 mysql     20   0 17.3g  16g 8104 S 186.9 52.3   3752:33 mysqld
 5700 nginx     20   0 1330m  66m 9496 S  8.9  0.2   0:20.82 php-fpm
 6424 nginx     20   0 1330m  65m 8372 S  8.3  0.2   0:04.97 php-fpm
 6573 nginx     20   0 1330m  64m 7368 S  8.3  0.2   0:01.49 php-fpm
 5927 nginx     20   0 1320m  56m 9272 S  7.6  0.2   0:12.54 php-fpm
 5956 nginx     20   0 1330m  65m 8500 S  7.6  0.2   0:12.70 php-fpm
 6126 nginx     20   0 1321m  57m 8964 S  7.3  0.2   0:09.72 php-fpm
 6127 nginx     20   0 1319m  54m 9520 S  6.6  0.2   0:08.73 php-fpm
 6131 nginx     20   0 1320m  56m 9404 S  6.6  0.2   0:09.43 php-fpm
 6174 nginx     20   0 1321m  56m 8444 S  6.3  0.2   0:08.92 php-fpm
 5790 nginx     20   0 1319m  54m 9468 S  5.6  0.2   0:17.33 php-fpm
 6575 nginx     20   0 1320m  55m 8212 S  5.6  0.2   0:02.11 php-fpm
 6160 nginx     20   0 1310m  44m 8296 S  4.0  0.1   0:10.05 php-fpm
 5597 nginx     20   0 1310m  46m 9556 S  3.6  0.1   0:21.03 php-fpm
 5786 nginx     20   0 1310m  45m 8528 S  3.6  0.1   0:15.53 php-fpm
 5797 nginx     20   0 1310m  46m 9444 S  3.6  0.1   0:14.02 php-fpm
 6158 nginx     20   0 1310m  45m 8324 S  3.6  0.1   0:10.20 php-fpm
 5698 nginx     20   0 1310m  46m 9184 S  3.3  0.1   0:20.62 php-fpm
 5779 nginx     20   0 1309m  44m 8336 S  3.3  0.1   0:15.34 php-fpm
 6540 nginx     20   0 1306m  40m 7884 S  3.3  0.1   0:02.46 php-fpm
 5553 nginx     20   0 1300m  36m 9568 S  3.0  0.1   0:21.58 php-fpm
 5722 nginx     20   0 1310m  45m 8552 S  3.0  0.1   0:17.25 php-fpm
 5920 nginx     20   0 1302m  36m 8208 S  3.0  0.1   0:14.23 php-fpm
 6432 nginx     20   0 1310m  45m 8420 S  3.0  0.1   0:05.86 php-fpm
 5285 nginx     20   0 1302m  38m 9696 S  2.7  0.1   0:23.41 php-fpm
```

2. 在返回结果界面，可查看核的 `wa` 值，`wa`（wait）表示 IO WAIT 的 CPU 占用。默认显示所有核的平均值，按1查看每个核的 `wa` 值。如下所示：

> **说明：**

> `wa` 通常为0%，如果经常浮动在1%之上，说明存储设备的速度已经太慢，无法跟上 CPU 的处理速度。

```
top - 19:42:08 up 23:59,  2 users,  load average: 34.64, 35.80, 35.76
Tasks: 679 total,   1 running, 678 sleeping,   0 stopped,   0 zombie
Cpu0  : 29.5%us,  3.7%sy,  0.0%ni, 48.7%id, 17.9%wa,  0.0%hi,  0.1%si,  0.0%st
Cpu1  : 29.3%us,  3.7%sy,  0.0%ni, 48.9%id, 17.9%wa,  0.0%hi,  0.1%si,  0.0%st
Cpu2  : 26.1%us,  3.1%sy,  0.0%ni, 64.4%id,  6.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu3  : 25.9%us,  3.1%sy,  0.0%ni, 65.5%id,  5.4%wa,  0.0%hi,  0.1%si,  0.0%st
Cpu4  : 24.9%us,  3.0%sy,  0.0%ni, 66.8%id,  5.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu5  : 24.9%us,  2.9%sy,  0.0%ni, 67.0%id,  4.8%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu6  : 24.2%us,  2.7%sy,  0.0%ni, 68.3%id,  4.5%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu7  : 24.3%us,  2.6%sy,  0.0%ni, 68.5%id,  4.2%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu8  : 23.8%us,  2.6%sy,  0.0%ni, 69.2%id,  4.1%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu9  : 23.9%us,  2.5%sy,  0.0%ni, 69.3%id,  4.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu10 : 23.3%us,  2.4%sy,  0.0%ni, 68.7%id,  5.6%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu11 : 23.3%us,  2.4%sy,  0.0%ni, 69.2%id,  5.1%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu12 : 21.8%us,  2.4%sy,  0.0%ni, 60.2%id, 15.5%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu13 : 21.9%us,  2.4%sy,  0.0%ni, 60.6%id, 15.2%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu14 : 21.4%us,  2.3%sy,  0.0%ni, 72.6%id,  3.7%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu15 : 21.5%us,  2.2%sy,  0.0%ni, 73.2%id,  3.1%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu16 : 21.2%us,  2.2%sy,  0.0%ni, 73.6%id,  3.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu17 : 21.2%us,  2.1%sy,  0.0%ni, 73.8%id,  2.8%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu18 : 20.9%us,  2.1%sy,  0.0%ni, 74.1%id,  2.9%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu19 : 21.0%us,  2.1%sy,  0.0%ni, 74.4%id,  2.5%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu20 : 20.7%us,  2.0%sy,  0.0%ni, 73.8%id,  3.4%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu21 : 20.8%us,  2.0%sy,  0.0%ni, 73.9%id,  3.2%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu22 : 20.8%us,  2.0%sy,  0.0%ni, 74.4%id,  2.8%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu23 : 20.8%us,  1.9%sy,  0.0%ni, 74.4%id,  2.8%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  32865032k total, 30209248k used,  2655784k free,   370748k buffers
Swap:  8388604k total,     5440k used,  8383164k free,  7986552k cached
```

监视磁盘 IO 统计信息

1. 执行命令 `atop` ，查看当前磁盘 IO 状态。本例中，磁盘 `sda` 显示 `busy 100%` ，表示已达到严重性能瓶颈。

```
ATOP - lemp              2017/01/23  19:42:32             ---------                10s elapsed
PRC | sys    3.18s | user  33.24s | #proc    679 | #tslpu    28 | #zombie    0 | #exit       0 |
CPU | sys      29% | user    330% | irq       1% | idle   1857% | wait    182% | curscal  69% |
CPL | avg1   33.00 | avg5   35.29 | avg15  35.59 | csw    62610 | intr   76926 | numcpu    24 |
MEM | tot    31.3G | free    2.1G | cache   7.6G | dirty  41.0M | buff  362.1M | slab    1.2G |
SWP | tot     8.0G | free    8.0G |              |              | vmcom  23.9G | vmlim  23.7G |
DSK |          sda | busy    100% | read       4 | write   1789 | MBw/s   2.84 | avio 5.58 ms |
NET | transport    | tcpi   10357 | tcpo    9065 | udpi       0 | udpo       0 | tcpao    174 |
NET | network      | ipi    10360 | ipo     9065 | ipfrw      0 | deliv  10359 | icmpo      0 |
NET | eth0      4% | pcki    6649 | pcko    6136 | si 1478 Kbps | so 4115 Kbps | erro       0 |
NET | lo      ---- | pcki    4082 | pcko    4082 | si 8967 Kbps | so 8967 Kbps | erro       0 |

PID   TID  THR  SYSCPU  USRCPU  VGROW  RGROW  RDDSK  WRDSK ST EXC S CPUNR  CPU CMD       1/12
 9783     -  156   0.21s  19.44s     0K  -788K     4K  1344K --    - S     4 197% mysqld
 5596     -    1   0.10s   0.62s 47204K 47004K     0K   220K --    - S    18   7% php-fpm
 6429     -    1   0.06s   0.34s 19840K 19968K     0K     0K --    - S    21   4% php-fpm
 6210     -    1   0.03s   0.30s -5216K -5204K     0K     0K --    - S    19   3% php-fpm
 5757     -    1   0.05s   0.27s 26072K 26012K     0K     4K --    - S    13   3% php-fpm
 6433     -    1   0.04s   0.28s -2816K -2816K     0K     0K --    - S    11   3% php-fpm
 5846     -    1   0.06s   0.22s -2560K -2660K     0K     0K --    - S     7   3% php-fpm
 5791     -    1   0.05s   0.21s  5764K  5692K     0K     0K --    - S    22   3% php-fpm
 5860     -    1   0.04s   0.21s 48088K 47724K     0K     0K --    - S     1   3% php-fpm
 6231     -    1   0.04s   0.20s  -256K    -4K     0K     0K --    - S     1   2% php-fpm
 6154     -    1   0.03s   0.21s -3004K -3184K     0K     0K --    - S    21   2% php-fpm
```

```
  6573     -    1   0.04s   0.20s  -512K  -168K     0K     0K --    - S     4   2% php-fpm
  6435     -    1   0.04s   0.19s -3216K -2980K     0K     0K --    - S    15   2% php-fpm
  5954     -    1   0.03s   0.20s     0K   164K     0K     4K --    - S     0   2% php-fpm
  6133     -    1   0.03s   0.19s 41056K 40432K     0K     0K --    - S    18   2% php-fpm
  6132     -    1   0.02s   0.20s 37836K 37440K     0K     0K --    - S    11   2% php-fpm
  6242     -    1   0.03s   0.19s -12.2M -12.3M     0K     4K --    - S    12   2% php-fpm
  6285     -    1   0.02s   0.19s 39516K 39420K     0K     0K --    - S     3   2% php-fpm
  6455     -    1   0.05s   0.16s 29008K 28560K     0K     0K --    - S    14   2% php-fpm
```

2. 查看占用磁盘 IO 的进程，有如下两种方法：
   - 继续在该界面按 **d**，可查看哪些进程正在使用磁盘 IO。返回结果如下：

```
ATOP - lemp                2017/01/23  19:42:46                ---------                2s elapsed
PRC | sys    0.24s | user   1.99s | #proc    679 | #tslpu    54 | #zombie    0 | #exit      0 |
CPU | sys      11% | user    101% | irq       1% | idle   2089% | wait    208% | curscal  63% |
CPL | avg1   38.49 | avg5   36.48 | avg15  35.98 | csw     4654 | intr    6876 | numcpu    24 |
MEM | tot    31.3G | free    2.2G | cache   7.6G | dirty  48.7M | buff  362.1M | slab    1.2G |
SWP | tot     8.0G | free    8.0G |              |              | vmcom  23.9G | vmlim  23.7G |
DSK |          sda | busy    100% | read       2 | write    362 | MBw/s   2.28 | avio 5.49 ms |
NET | transport    | tcpi    1031 | tcpo     968 | udpi       0 | udpo       0 | tcpao     45 |
NET | network      | ipi     1031 | ipo      968 | ipfrw      0 | deliv   1031 | icmpo      0 |
NET | eth0      1% | pcki     558 | pcko     508 | si  762 Kbps | so 1077 Kbps | erro       0 |
NET | lo      ---- | pcki     406 | pcko     406 | si 2273 Kbps | so 2273 Kbps | erro       0 |

  PID          TID          RDDSK          WRDSK         WCANCL          DSK         CMD          1/5
 9783            -             0K          468K             16K          40%         mysqld
 1930            -             0K          212K              0K          18%         flush-8:0
 5896            -             0K          152K              0K          13%         nginx
  880            -             0K          148K              0K          13%         jbd2/sda5-8
 5909            -             0K           60K              0K           5%         nginx
 5906            -             0K           36K              0K           3%         nginx
 5907            -            16K            8K              0K           2%         nginx
 5903            -            20K            0K              0K           2%         nginx
 5901            -             0K           12K              0K           1%         nginx
 5908            -             0K            8K              0K           1%         nginx
 5894            -             0K            8K              0K           1%         nginx
 5911            -             0K            8K              0K           1%         nginx
 5900            -             0K            4K              4K           0%         nginx
 5551            -             0K            4K              0K           0%         php-fpm
 5913            -             0K            4K              0K           0%         nginx
 5895            -             0K            4K              0K           0%         nginx
 6133            -             0K            0K              0K           0%         php-fpm
 5780            -             0K            0K              0K           0%         php-fpm
 6675            -             0K            0K              0K           0%         atop
```

   - 也可以执行命令 `iotop -oPa` 查看哪些进程占用磁盘 IO。返回结果如下：

```
Total DISK READ: 15.02 K/s | Total DISK WRITE: 3.82 M/s
  PID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO>    COMMAND
 1930 be/4 root          0.00 B   1956.00 K  0.00 % 83.34 % [flush-8:0]
 5914 be/4 nginx         0.00 B      0.00 B  0.00 % 36.56 % nginx: cache manager process
  880 be/3 root          0.00 B     21.27 M  0.00 % 35.03 % [jbd2/sda5-8]
 5913 be/2 nginx        36.00 K   1000.00 K  0.00 %  8.94 % nginx: worker process
 5910 be/2 nginx         0.00 B   1048.00 K  0.00 %  8.43 % nginx: worker process
 5896 be/2 nginx        56.00 K    452.00 K  0.00 %  6.91 % nginx: worker process
 5909 be/2 nginx        20.00 K   1144.00 K  0.00 %  6.24 % nginx: worker process
 5890 be/2 nginx        48.00 K    692.00 K  0.00 %  6.07 % nginx: worker process
 5892 be/2 nginx        84.00 K    736.00 K  0.00 %  5.71 % nginx: worker process
 5901 be/2 nginx        20.00 K    504.00 K  0.00 %  5.46 % nginx: worker process
```

```
    5899 be/2 nginx          0.00 B    596.00 K  0.00 %  5.14 % nginx: worker process
    5897 be/2 nginx         28.00 K   1388.00 K  0.00 %  4.90 % nginx: worker process
    5908 be/2 nginx         48.00 K    700.00 K  0.00 %  4.43 % nginx: worker process
    5905 be/2 nginx         32.00 K   1140.00 K  0.00 %  4.36 % nginx: worker process
    5900 be/2 nginx          0.00 B   1208.00 K  0.00 %  4.31 % nginx: worker process
    5904 be/2 nginx         36.00 K   1244.00 K  0.00 %  2.80 % nginx: worker process
    5895 be/2 nginx         16.00 K    780.00 K  0.00 %  2.50 % nginx: worker process
    5907 be/2 nginx          0.00 B   1548.00 K  0.00 %  2.43 % nginx: worker process
    5903 be/2 nginx         36.00 K   1032.00 K  0.00 %  2.34 % nginx: worker process
    6130 be/4 nginx          0.00 B     72.00 K  0.00 %  2.18 % php-fpm: pool www
    5906 be/2 nginx         12.00 K    844.00 K  0.00 %  2.10 % nginx: worker process
    5889 be/2 nginx         40.00 K   1164.00 K  0.00 %  2.00 % nginx: worker process
    5894 be/2 nginx         44.00 K    760.00 K  0.00 %  1.61 % nginx: worker process
    5902 be/2 nginx         52.00 K    992.00 K  0.00 %  1.55 % nginx: worker process
    5893 be/2 nginx         64.00 K    972.00 K  0.00 %  1.22 % nginx: worker process
    5814 be/4 nginx         36.00 K     44.00 K  0.00 %  1.06 % php-fpm: pool www
    6159 be/4 nginx          4.00 K      4.00 K  0.00 %  1.00 % php-fpm: pool www
    5693 be/4 nginx          0.00 B      4.00 K  0.00 %  0.86 % php-fpm: pool www
    5912 be/2 nginx         68.00 K    300.00 K  0.00 %  0.72 % nginx: worker process
    5911 be/2 nginx         20.00 K    788.00 K  0.00 %  0.72 % nginx: worker process
```

通过 `man iotop` 命令可以查看以下几个参数的含义。返回结果如下：

```
   -o, --only
          Only show processes or threads actually doing I/O, instead of showing all processes or
threads. This can be dynamically toggled by pressing o.
   -P, --processes
          Only show processes. Normally iotop shows all threads.

   -a, --accumulated
          Show accumulated I/O instead of bandwidth. In this mode, iotop shows the amount of I/O
processes have done since iotop started.
```

其他原因

节点上部署了其他非 K8S 管理的服务可能造成节点高负载问题。例如，在节点上安装不被 K8S 所管理的数据库。

节点内存碎片化排障处理

最近更新时间：2024-08-28 16:51:01

本文档介绍如何判断 TKE 集群中存在问题是否由内存碎片化引起，并给出解决方法，请按照以下步骤进行排查并解决。

问题分析

内存页分配失败，内核日志将会出现以下报错：

```
mysqld: page allocation failure. order:4, mode:0x10c0d0
```

- `mysqld`：为被分配内存的程序。
- `order`：表示需要分配连续页的数量（2^order），本例中4则表示 2^4 = 16 个连续的页。
- `mode`：内存分配模式的标识，在内核源码文件 `include/linux/gfp.h` 中定义，通常是多个标识项与运算的结果。不同版本内核有一定区别。例如，在新版内核中 `GFP_KERNEL` 是 `__GFP_RECLAIM | __GFP_IO | __GFP_FS` 的运算结果，而 `__GFP_RECLAIM` 是 `___GFP_DIRECT_RECLAIM|___GFP_KSWAPD_RECLAIM` 的运算结果。

> ⚠ 注意：
> - 当 order 值为0时，说明系统已经完全没有可用内存。
> - 当 order 值较大时，说明内存已碎片化，无法分配连续的大页内存。

现象描述

容器启动失败

K8S 会为每个 Pod 创建 netns 来隔离 network namespace，内核初始化 netns 时会为其创建 nf_conntrack 表的 cache，需要申请大页内存，如果此时系统内存已经碎片化，无法分配到足够的大页内存内核就会出现如下报错（`v2.6.33 - v4.6`）：

```
runc:[1:CHILD]: page allocation failure: order:6, mode:0x10c0d0
```

Pod 将会一直处于 ContainerCreating 状态，dockerd 启动容器也将失败。日志报错如下：

```
Jan 23 14:15:31 dc05 dockerd: time="2019-01-23T14:15:31.288446233+08:00" level=error msg="containerd:
start container" error="oci runtime error: container_linux.go:247: starting container process caused
\"process_linux.go:245: running exec setns process for init caused \\\"exit status 6\\\"\"\n"
id=5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad36d70b8640cb01c
Jan 23 14:15:31 dc05 dockerd: time="2019-01-23T14:15:31.317965799+08:00" level=error msg="Create container
failed with error: invalid header field value \"oci runtime error: container_linux.go:247: starting
container process caused \\\"process_linux.go:245: running exec setns process for init caused \\\\\\\"exit
status 6\\\\\\\"\\\"\\n\""
```

Kubelet 日志报错如下：

```
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352386   26037 remote_runtime.go:91] RunPodSandbox from
runtime service failed: rpc error: code = 2 desc = failed to start sandbox container for pod
"matchdataserver-1255064836-t4b2w": Error response from daemon: {"message":"invalid header field value
\"oci runtime error: container_linux.go:247: starting container process caused \\\"process_linux.go:245:
running exec setns process for init caused \\\\\\\"exit status 6\\\\\\\"\\\"\\n\""}
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352496   26037 kuberuntime_sandbox.go:54] CreatePodSandbox
for pod "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)" failed: rpc error:
code = 2 desc = failed to start sandbox container for pod "matchdataserver-1255064836-t4b2w": Error
response from daemon: {"message":"invalid header field value \"oci runtime error: container_linux.go:247:
starting container process caused \\\"process_linux.go:245: running exec setns process for init caused
\\\\\\\"exit status 6\\\\\\\"\\\"\\n\""}
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352518   26037 kuberuntime_manager.go:618] createPodSandbox
for pod "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)" failed: rpc error:
```

```
code = 2 desc = failed to start sandbox container for pod "matchdataserver-1255064836-t4b2w": Error
response from daemon: {"message":"invalid header field value \"oci runtime error: container_linux.go:247:
starting container process caused \\\"process_linux.go:245: running exec setns process for init caused
\\\\\\\"exit status 6\\\\\\\"\\\"\\n\""}
Jan 23 14:15:31 dc05 kubelet: E0123 14:15:31.352580   26037 pod_workers.go:182] Error syncing pod
485fd485-1ed6-11e9-8661-0a587f8021ea ("matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-
0a587f8021ea)"), skipping: failed to "CreatePodSandbox" for "matchdataserver-1255064836-
t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)" with CreatePodSandboxError: "CreatePodSandbox for pod
\"matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)\" failed: rpc error: code =
2 desc = failed to start sandbox container for pod \"matchdataserver-1255064836-t4b2w\": Error response
from daemon: {\"message\":\"invalid header field value \\\"oci runtime error: container_linux.go:247:
starting container process caused \\\\\\\"process_linux.go:245: running exec setns process for init caused
\\\\\\\\\\\\\\\\\"exit status 6\\\\\\\\\\\\\\\\\"\\\\\\\"\\\\n\\\"\"}"
Jan 23 14:15:31 dc05 kubelet: I0123 14:15:31.372181   26037 kubelet.go:1916] SyncLoop (PLEG):
"matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)", event:
&pleg.PodLifecycleEvent{ID:"485fd485-1ed6-11e9-8661-0a587f8021ea", Type:"ContainerDied",
Data:"5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad36d70b8640cb01c"}
Jan 23 14:15:31 dc05 kubelet: W0123 14:15:31.372225   26037 pod_container_deletor.go:77] Container
"5b9be8c5bb121264899fac8d9d36b02150269d41ce96ba6ad36d70b8640cb01c" not found in pod's containers
Jan 23 14:15:31 dc05 kubelet: I0123 14:15:31.678211   26037 kuberuntime_manager.go:383] No ready sandbox
for pod "matchdataserver-1255064836-t4b2w_basic(485fd485-1ed6-11e9-8661-0a587f8021ea)" can be found. Need
to start a new one
```

执行命令 `cat /proc/buddyinfo` 查看 slab，系统不存在大块内存时返回0数量较多。如下所示：

```
$ cat /proc/buddyinfo
Node 0, zone      DMA      1      0      1      0      2      1      1      0      1      1      3
Node 0, zone    DMA32   2725    624    489    178      0      0      0      0      0      0      0
Node 0, zone   Normal   1163   1101    932    222      0      0      0      0      0      0      0
```

系统 OOM

内存碎片化严重，系统缺少大页内存，即使是当前系统总内存较多的情况下，也会出现给程序分配内存失败的现象。此时系统会做出内存不够用的误判，认为需要停止一些进程来释放内存，从而导致系统 OOM。

处理步骤

1. 周期性地或者在发现大块内存不足时，先进行 drop_cache 操作。

```
echo 3 > /proc/sys/vm/drop_caches
```

2. 必要时执行以下操作进行内存整理。

> **注意：**
> 请谨慎执行此步骤，该操作开销较大，会造成业务卡顿。

```
echo 1 > /proc/sys/vm/compact_memory
```

集群 DNS 解析异常排障处理

最近更新时间：2024-10-29 16:09:22

排查思路

1. 确保集群 DNS 正常运行

容器内解析 DNS 通过集群 DNS（通常是 CoreDNS），首先要确保集群 DNS 运行正常。通过 kubelet 启动参数 `--cluster-dns` 查看 DNS 服务的 Cluster IP：

```
$ ps -ef | grep kubelet
... /usr/bin/kubelet --cluster-dns=172.16.14.217 ...
```

找到 DNS 的 Service：

```
$ kubectl get svc -n kube-system | grep 172.16.14.217
kube-dns               ClusterIP    172.16.14.217    <none>        53/TCP,53/UDP              47d
```

检查是否存在 endpoint：

```
$ kubectl -n kube-system describe svc kube-dns | grep -i endpoints
Endpoints:         172.16.0.156:53,172.16.0.167:53
Endpoints:         172.16.0.156:53,172.16.0.167:53
```

检查 endpoint 对应的 Pod 是否正常：

```
$ kubectl -n kube-system get pod -o wide | grep 172.16.0.156
kube-dns-898dbbfc6-hvwlr             3/3       Running   0          8d        172.16.0.156   10.0.0.3
```

2. 确保 Pod 能与集群 DNS 通信

检查 Pod 是否能连上集群 DNS，可以在 Pod 里执行 telnet 命令，查看 DNS 的53端口：

```
# 连 dns service 的 cluster ip
$ telnet 172.16.14.217 53
```

> **说明：**
> 如果容器内没有 telnet 等测试工具，可以 使用 nsenter 进入 netns 抓包，然后利用宿主机上的 telnet 进行测试。

若检查到网络不通，则需要排查以下网络设置：

- 检查节点的安全组设置，需要放开集群的容器网段。
- 检查是否还有防火墙规则，检查 iptables。
- 检查 kube-proxy 是否正常运行，集群 DNS 的 IP 是 Cluster IP，会经过 kube-proxy 生成的 iptables 或 ipvs 规则进行转发。

3. 抓包

如果集群 DNS 正常运行，Pod 能与集群 DNS 通信检查，那么可以通过抓包进一步检查。如果问题容易复现，可以使用 nsenter 进入 netns 抓容器内的包：

```
tcpdump -i any port 53 -w dns.pcap
# tcpdump -i any port 53 -nn -tttt
```

若仍无法分析，可以在请求链路上的多个点抓包分析，如 Pod 的容器内、宿主机 cbr0网桥、宿主机主网卡（eth0）、CoreDNS Pod 所在宿主机主网卡、cbr0 以及容器内。等待问题复现并拉通对比分析，评估丢包点。

现象和原因

5秒延时

如果 DNS 查询经常延时5秒才返回，通常是遇到内核 conntrack 冲突导致的丢包，根本原因是内核 conntrack 模块的 bug，netfilter 做 NAT 时可能发生资源竞争导致部分报文丢弃。

- 只有多个线程或进程，并发从同一个 socket 发送相同五元组的 UDP 报文时，有一定概率会发生。
- glibc，musl（Alpine Linux 的 libc 库）都使用 “parallel query”，即并发发出多个查询请求，因此易碰到这样的冲突，造成查询请求被丢弃。
- 由于 ipvs 也使用了 conntrack，使用 kube-proxy 的 ipvs 模式，因此无法避免该问题。

规避方法

使用 LocalDNS 可以规避该问题。容器的 DNS 请求都发往本地的 DNS 缓存服务（dnsmasq，nscd 等），不需要走 DNAT，也不会发生 conntrack 冲突。另外，也可避免 DNS 服务成为性能瓶颈。
使用 LocalDNS 缓存有两种方式：

- 每个容器自带一个 DNS 缓存服务。
- 每个节点运行一个 DNS 缓存服务，所有容器都把本节点的 DNS 缓存作为自己的 nameserver。

解析外部域名超时

可能原因：

- 上游 DNS 故障。
- 上游 DNS 的 ACL 或防火墙拦截了报文。

所有解析都超时

如果集群内某个 Pod 不管解析 Service 还是外部域名都失败，通常是 Pod 与集群 DNS 之间通信有问题。
可能原因：

- 节点防火墙没放开集群网段，导致如果 Pod 跟集群 DNS 的 Pod 不在同一个节点就无法通信，DNS 请求也就无法被收到。
- kube-proxy 异常。

集群 Kube-Proxy 异常排障处理

最近更新时间：2024-11-14 17:25:32

在使用 TKE 集群服务的过程中，某些场景下，可能会出现服务访问不通的问题，如果确认后端 Pod 访问正常，则可能是由于 kube-proxy 组件版本较低，导致节点上的 iptables 或 ipvs 服务转发规则下发失败。本文档整理了低版本 kube-proxy 存在的若干问题，并给出相应的修复指引。若本文档无法解决您所遇到的问题，请 联系我们 来寻求帮助。

kube-proxy 未能正确适配节点 iptables 后端

错误信息示例

```
Failed to execute iptables-restore: exit status 2 (iptables-restore v1.8.4 (legacy): Couldn't load target
'KUBE-MARK-DROP':No such file or directory
```

问题原因

1. kube-proxy 在执行 iptables-restore 时，所依赖的 `KUBE-MARK-DROP` Chain 不存在，导致同步规则失败后退出。 `KUBE-MARK-DROP` Chain 由 kubelet 负责维护。
2. 一些高版本的 OS 使用的 iptables 后端为 nft，而低版本 kube-proxy 使用的 iptables 后端为 legacy。当低版本 kube-proxy 运行在高版本 OS 上时，会因为 iptables 后端不匹配而读不到 `KUBE-MARK-DROP` Chain。高版本 OS 包括：
   - tlinux2.6 (tk4)
   - tlinux3.1
   - tlinux3.2
   - CentOS8
   - Ubuntu20

修复指引

升级 kube-proxy，需要按如下逻辑处理：

TKE 集群版本	修复策略
1.24	升级 kube-proxy 到 v1.24.4-tke.5 及以上
1.22	升级 kube-proxy 到 v1.22.5-tke.11 及以上
1.20	升级 kube-proxy 到 v1.20.6-tke.31 及以上
1.18	升级 kube-proxy 到 v1.18.4-tke.35 及以上
1.16	升级 kube-proxy 到 v1.16.3-tke.34 及以上
1.14	升级 kube-proxy 到 v1.14.3-tke.28 及以上
1.12	升级 kube-proxy 到 v1.12.4-tke.32 及以上
1.10	升级 kube-proxy 到 v1.10.5-tke.20 及以上

> **说明：**
> TKE 最新版本信息，请参见 TKE Kubernetes Revision 版本历史 。

kube-proxy 操作 iptables 锁相关的问题

其它组件未挂载 iptables 锁导致并发写入失败

错误信息示例

```
Failed to execute iptables-restore: exit status 1 (iptables-restore: line xxx failed)
```

问题原因

1. iptables 相关命令（如 iptables-restore）在向内核写入 iptables 规则时，为了避免多个实例并发写入，会利用 file lock 来做同步，linux 下该文件一般为：`/run/xtables.lock`
2. 对于要调用 iptables 相关命令的 Pod，如 kube-proxy, kube-router 以及客户侧的 HostNetwork Pod，如果没有挂载该文件，可能发生如上并发写入的错误。

修复指引

对于要调用 iptables 相关命令的 Pod 需要将主机侧 `/run/xtables.lock` 文件挂载到 Pod 中，配置方式如下：

```
        volumeMounts:
        - mountPath: /run/xtables.lock
          name: xtables-lock

      volumes:
      - hostPath:
          path: /run/xtables.lock
          type: FileOrCreate
        name: xtables-lock
```

iptables-restore 版本低导致不支持阻塞写入

错误信息示例

```
Failed to execute iptables-restore: exit status 4 (Another app is currently holding the xtables lock.
Perhaps you want to use the -w option?)
```

问题原因

1. iptables 相关命令（如 iptables-restore）在向内核写入 iptables 规则时，为了避免多个实例并发写入，会利用 file lock 来做同步，iptables-restore 在执行时首先尝试获取 file lock，如果当前有其它进程持有锁，则退出。
2. 该报错是一个软错误，kube-proxy 会在下个同步周期（或下个 Service 相关的事件触发时）再次尝试执行，如果重试多次都获取不到锁，则表现为规则同步时延较大。
3. 高版本 iptables-restore 提供了一个 `-w(--wait)` 选项，如设置 -w=5 时，iptables-restore 会在拿锁操作阻塞 5s，这使得 5s 内一旦其它进程释放锁，iptables-restore 可以继续操作。

修复指引

1. 如果 kube-proxy 为节点侧二进制部署，可以通过升级节点 OS 版本来提升 iptables-restore 版本，需要按如下逻辑处理：

节点 OS	升级目标版本
CentOS	7.2 及以上
Ubuntu	20.04 及以上
Tencent Linux	2.4 及以上

2. 如果 kube-proxy 为集群内 Daemonset 部署，则可以通过升级 kube-proxy 来提升 iptables-restore 版本，需要按如下逻辑处理：

TKE 集群版本	修复策略
> 1.12	不存在此问题，无需修复
1.12	升级 kube-proxy 到 v1.12.4-tke.31 及以上
< 1.12	升级 TKE 集群到高版本

> **说明：**

> TKE 最新版本信息，请参见 TKE Kubernetes Revision 版本历史 。

其它组件持有 iptables 锁时间过长

错误信息示例

```
Failed to ensure that filter chain KUBE-SERVICES exists: error creating chain "KUBE-EXTERNAL-SERVICES":
exit status 4: Another app is currently holding the xtables lock. Stopped waiting after 5s.
```

问题原因

1. iptables 相关命令（如 iptables-restore）在向内核写入 iptables 规则时，为了避免多个实例并发写入，会利用 file lock 来做同步，iptables-restore 在执行时首先尝试获取 file lock，如果当前有其它进程持有锁，则阻塞特定时间（取决于-w 参数的值，默认5s），该时间内拿到锁，则继续操作，拿不到则退出。
2. 该报错说明其它组件持有 iptables file lock 时间超过 5s。

修复指引

尽量减小其它组件持有 iptables file lock 的时间，如 TKE 控制台组件管理提供的 NetworkPolicy（kube-router）组件，其低版本持有 iptables 锁的时间较长，可以通过升级来解决，当前最新版为：`v1.3.2`

kube-proxy 到 kube-apiserver 连接异常

错误信息示例

```
Failed to list *core.Endpoints: Stream error http2.StreamError{StreamID:0xea1, Code:0x2, Cause:error(nil)}
when reading response body, may be caused by closed connection. Please retry.
```

问题原因

低版本 Kubernetes 调用 go http2 的包存在一个 bug，该 bug 导致客户端会使用到一个 apiserver 的已经关闭的连接，kube-proxy 踩中这个 bug 后，会导致同步规则失败。更多详情可参考 Issue87615 、PR95981 。

修复指引

升级 kube-proxy，需要按如下逻辑处理：

TKE 集群版本	修复策略
> 1.18	不存在此问题，无需修复
1.18	升级 kube-proxy 到 v1.18.4-tke.26 及以上
< 1.18	升级 TKE 集群到高版本

> **说明：**
> TKE 最新版本信息，请参见 TKE Kubernetes Revision 版本历史 。

kube-proxy 首次启动发生 panic，重启后正常

错误信息示例

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x50 pc=0x1514fb8]
```

问题原因

1. 该版本 kube-proxy 社区的代码存在 bug，初始化时统计加载的内核模块有缺失，导致有变量未初始化即使用。
2. 日志不够详尽，未输出是否能使用 ipvs 模式的判断结果。更多详情可参考 Issue89729 、PR89823 、PR89785 。

修复指引

升级 kube-proxy，需要按如下逻辑处理：

TKE 集群版本	修复策略
> 1.18	不存在此问题，无需修复
1.18	升级 kube-proxy 到 v1.18.4-tke.26 及以上
< 1.18	不存在此问题，无需修复

> **说明：**
> TKE 最新版本信息，请参见 TKE Kubernetes Revision 版本历史 。

kube-proxy 不间断 panic

错误信息示例

```
Observed a panic: "slice bounds out of range" (runtime error: slice bounds out of range)
```

问题原因

kube-proxy 社区的代码存在 bug，在执行 iptables-save 时将标准输出和标准错误定向到同一个 buffer 中，而这两者的输出先后顺序是不确定的，这导致 buffer 中的数据格式不符合预期，在处理时发生 panic。更多详情可参考 Issue78443 、PR78428 。

修复指引

升级 kube-proxy，需要按如下逻辑处理：

TKE 集群版本	修复策略
> 1.14	不存在此问题，无需修复
1.14	升级 kube-proxy 到 v1.14.3-tke.27 及以上
1.12	升级 kube-proxy 到 v1.12.4-tke.31 及以上
< 1.12	不存在此问题，无需修复

> **说明：**
> TKE 最新版本信息，请参见 TKE Kubernetes Revision 版本历史 。

kube-proxy ipvs 模式下周期性占用较高 CPU

问题原因

kube-proxy 频繁刷新节点 Service 转发规则导致，触发原因：

- kube-proxy 周期性同步规则较为频繁。
- 业务 Service 或 Pod 变更频繁。

修复指引

如果是 kube-proxy 周期性同步规则较为频繁导致，需要调整其相关参数，旧版本 kube-proxy 的参数默认为：

```
--ipvs-min-sync-period=1s（最小刷新间隔1s）
--ipvs-sync-period=5s（5s周期性刷新）
```

这导致 kube-proxy 每5s刷新一次节点 iptables 规则，消耗较多 CPU，推荐改为：

```
--ipvs-min-sync-period=0s（发生事件实时刷新）
```

```
--ipvs-sync-period=30s（30s周期性刷新）
```

以上配置值为参数默认值，您可以自行选择是否配置这些参数。

节点上运行多个 kube-proxy 进程

问题原因

原因之一是修改过运行时的存储目录，但未同步迁移数据，导致拉起多个进程，实际在生效的进程是一个不受 pod 管理的“裸”进程，该问题并不直接影响服务访问，但会导致后续的组件升级等针对 pod 的更新操作不实际“生效”。

修复指引

在 TKE 节点上下载并以 root 身份运行以下修复脚本，来清理非 pod 所管理的 kube-proxy 进程，下载方式：

```
wget http://mirrors.tencentyun.com/install/cts/linux/tke/kubeproxy/fix-kube-proxy.sh
```

集群 API Server 网络无法访问排障处理

最近更新时间：2024-12-12 16:31:12

开启内网访问后无法访问

您可以直接在容器服务控制台上 开启内网访问 。如果开启内网访问之后仍出现无法访问的情况，建议您对应集群类型进行如下检查：

托管集群

参考 查看节点安全组配置 检查集群中节点的安全组是否正确放通30000-32768端口区间。

独立集群

1. 参考 查看节点安全组配置 检查集群中节点的安全组是否正确放通30000-32768端口区间。
2. 开启内网访问时，您已通过控制台设置了 VPC 子网网段，请检查集群中 Master 节点是否正确放通该 VPC 子网网段。
3. 检查集群中 Master 节点的安全组是否正确放通 Master 节点所在的 VPC 网段或 VPC 子网网段。

开启公网访问后无法访问

您可以直接在容器服务控制台上 开启公网访问 。如果开启公网访问之后仍出现无法访问的情况，建议您对应集群类型进行如下检查：

托管集群

检查安全组来源 CIDR 是否正确设置，或将来源 `0.0.0.0/0` 设置为全放通之后，再进行公网访问测试。

> **注意：**
> 此操作具有一定安全风险，建议您测试完成后尽快修改来源 IP。

独立集群

独立集群开启公网访问之后，会在集群中自动创建 `default/kubelb-internet` Service 对象。该 Service 会自动绑定一个公网类型的 CLB，默认不会为该 CLB 绑定安全组（即全放通），且 EXTERNAL-IP 字段显示即为此 CLB 的 VIP。如下所示：

```
$ kubectl get service kubelb-internet
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)          AGE
kubelb-internet   LoadBalancer   172.16.252.94   152.136.8.98    443:32750/TCP    3m4s
```

1. 检查 `default/kubelb-internet` Service 对象已绑定的 CLB 是否设置了安全组，并且安全组是否正确配置。
2. 参考 查看节点安全组配置 检查集群中 Master 节点的安全组是否正确放通30000 - 32768端口区间。
3. 检查集群中 Master 节点的安全组是否正确放通 Master 节点所在的 VPC 网段或 VPC 子网网段。

Service&Ingress 网络无法访问排障处理

最近更新时间：2025-03-26 11:18:42

Service 提供公网或内网服务无法访问

提供公网服务或者内网服务的 Service，如果出现无法访问或者 CLB 端口状态异常的情况，建议您进行如下检查：

1. 参考 查看节点安全组配置 检查集群中节点的安全组是否正确放通30000-32768端口区间。
2. 如果是公网服务，则进一步检查节点是否有公网带宽（仅针对 传统账户类型 ）。
3. 如果 Service 的 type 为 loadbalancer 类型，可忽略 CLB，直接检查 NodeIP + NodePort 是否可以访问。
4. 检查 Service 是否可以在集群中正常访问。

Ingress 提供公网服务无法访问

提供公网服务的 Ingress 如果出现无法访问的情况，建议您进行如下检查：

1. 若请求返回504，请参考 查看节点安全组配置 检查集群中节点的安全组是否正确放通30000-32768端口区间。
2. 检查 Ingress 绑定的 CLB 是否设置了未放通443端口的安全组。
3. 检查 Ingress 后端服务 Service 是否可以在集群中正常访问。
4. 若请求返回404，请检查 Ingress 转发规则是否正确设置。

集群内 Service 无法访问

在 TKE 集群内，Pod 之间一般是通过 Service 的 DNS 名称 `my-svc.my-namespace.svc.cluster.local` 来进行互访，如果在 Pod 内发生 Service 无法访问的情况，建议您进行如下检查：

1. 检查 Service 的 `spec.ports` 字段是否正确。
2. 测试 Service 的 ClusterIP 是否可通。若可通，则说明集群内 DNS 解析存在问题，可参考 集群 DNS 解析异常排障处理 进一步检查。
3. 测试 Service 的 Endpoint 是否可通。若不通，可参考 集群中不同节点上的容器（Pod）无法互访 进一步检查。
4. 检查当前 Pod 所在节点上的 iptables 或者 ipvs 转发规则是否完整。

因 CLB 回环问题导致服务访问不通或访问 Ingress 存在延时

背景信息

为了使集群内的业务通过 CLB 访问 LoadBalancer 类型的 Service 时链路更短，性能更好，社区版本的 Kube-Proxy 在 IPVS 模式下会将 CLB IP 绑定到本地 dummy 网卡，使得该流量在节点发生短路，不再经过 CLB，直接在本地转发到 Endpoint。但这种优化行为和腾讯云 CLB 的健康检查机制会产生冲突，下面将做具体分析，并给出对应解法。同样，用户使用容器服务 TKE 时，会遇到因 CLB 回环问题导致服务访问不通或访问 Ingress 存在几秒延时的现象。

现象描述

CLB 回环问题可能导致存在以下现象：

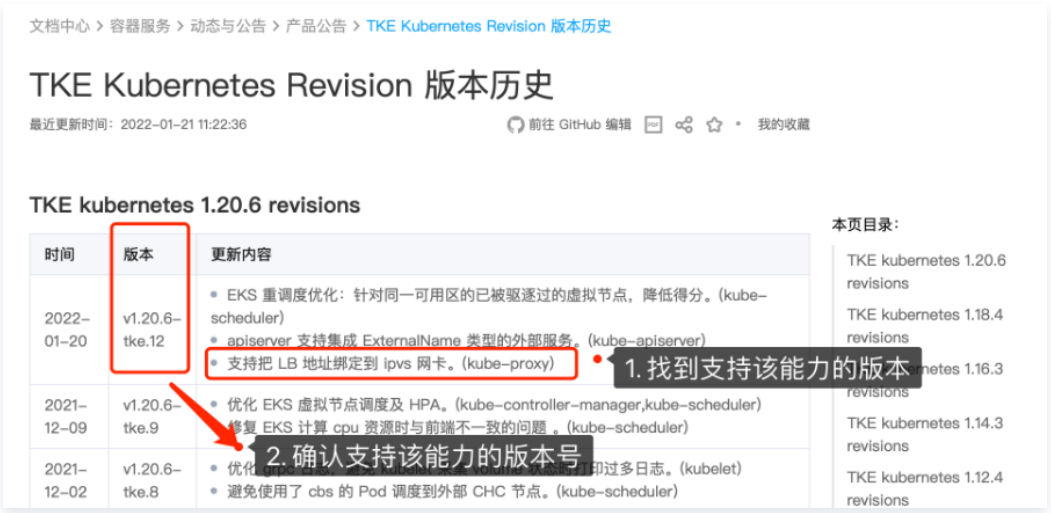
- IPtables 或 IPVS 模式下，访问本集群内网 Ingress 会出现4秒延时或不通。
- IPVS 模式下，集群内访问本集群 LoadBanacer 类型的内网 Service 出现完全不通，或者连通不稳定。

规避方法

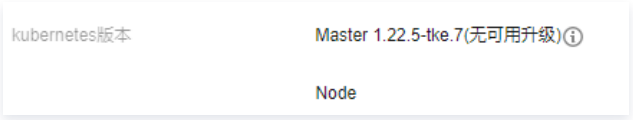
避免四层回环问题

解决使用 Service 时可能遇到的回环问题，步骤如下：

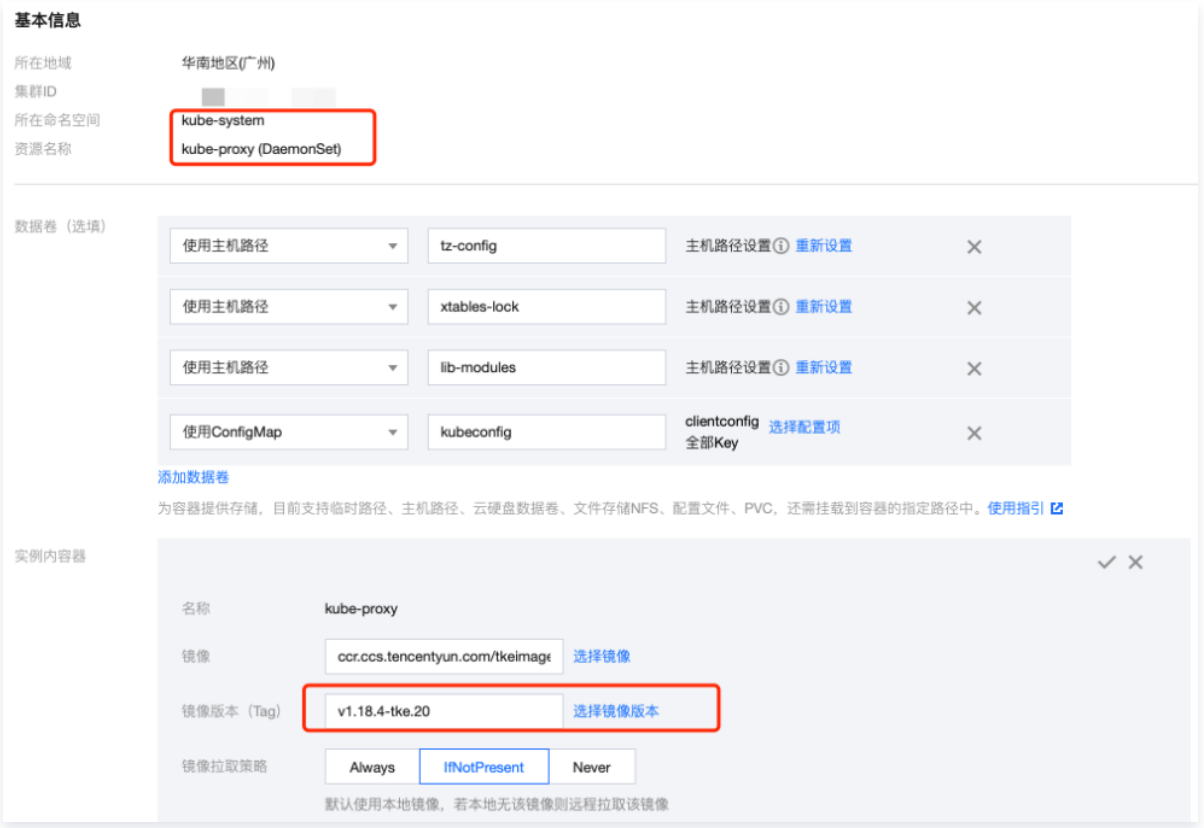
1. 检查 kube-proxy 的版本是否是最新版本，如果不是请参照如下步骤升级：
   1.1 解决该问题需要 kube-proxy 支持把 LB 地址绑定到 IPVS 网卡，可以在 TKE Kubernetes Revision 版本历史 里找到支持该能力的版本号，例如：v1.24.4-tke.6、v1.22.5-tke.13、v1.20.6-tke.12、v1.18.4-tke.20、v1.16.3-tke.25、v1.14.3-tke.24、v1.12.4-tke.30。后续更高的版本默认包含该能力：



1.2 确定集群的版本：您可以登录 容器服务控制台 ，在集群的**基本信息**页查看到当前集群的版本号，如下图所示的集群是1.22.5的版本：



1.3 找到 kube-system 命名空间下面名为 kube-proxy 的 DaemonSet，更新其镜像的版本号，选择支持该能力的版本号，或者大于该版本号的版本。例如，1.18 版本的集群要选择的镜像版本号要大于 v1.18.4-tke.20。如下图所示：



2. 为所有的 Service 添加注解：

```
service.cloud.tencent.com/prevent-loopback: "true"
```

效果如下：

- kube-proxy 将据此信息将 CLB VIP 绑定到本地，可解决回环问题。
- service-controller 将调用 CLB 接口，将其健康探测 IP 改为 100.64 网段 IP，可解决健康检查问题。

避免七层回环问题

解决使用 Ingress 时可能遇到的回环问题，步骤如下：

1. 提交工单 申请开通 CLB 7层 SNAT 和 Keep Alive 能力的白名单。

> **注意：**
> 该能力将启用长连接，CLB 与后端服务之间使用长连接，CLB 不再透传源 IP，请从 XFF 中获取源 IP。为保证正常转发，请在 CLB 上打开安全组默认放通或者在 CVM 的安全组上放通 100.127.0.0/16。更多请参考 配置监听器 。

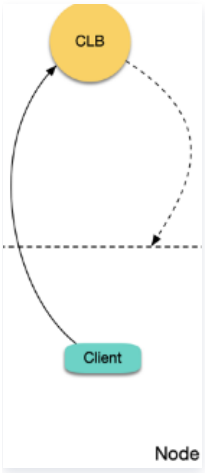
2. 利用 TkeServiceConfig 的能力增强 Ingress 的配置能力,对于存量的 Ingress，需要添加一个 annotation：`ingress.cloud.tencent.com/tke-service-config-auto: "true"`，详情见 Ingress Annotation 说明 。
   目的：为该 Ingress 自动生成对应的 TkeServiceConfig，提供 Ingress 额外的配置。
   效果：该 Ingress 会生成一个名为：<IngressName>-auto-ingress-config 的 TkeServiceConfig 类型资源。
3. 在生成名为：<IngressName>-auto-ingress-config 的 TkeServiceConfig 里面添加一个字段，字段名为 `spec.loadBalancer.l7Listeners[].keepaliveEnable`，字段值为1。注意每一个端口都需要添加该字段，如下图所示：

```
apiVersion: cloud.tencent.com/v1alpha1
kind: TkeServiceConfig
metadata:
  name: jetty-ingress-config
  namespace: default
spec:
  loadBalancer:
    l7Listeners:
    - protocol: HTTP
      keepaliveEnable: 1
      port: 80
      domains:
      - domain: ""      # domain为空表示使用VIP作为域名
        rules:
        - url: "/health"
          forwardType: HTTP # 指定后端协议为 HTTP
          healthCheck:
            enable: false
    - protocol: HTTPS
      keepaliveEnable: 1
      port: 443
      domains:
      - domain: "sample.tencent.com"
        rules:
        - url: "/"
          forwardType: HTTPS # 指定后端协议为 HTTPS
          session:
            enable: true
            sessionExpireTime: 3600
          healthCheck:
```

更多请参考 Ingress 使用 TkeServiceConfig 配置 CLB 。

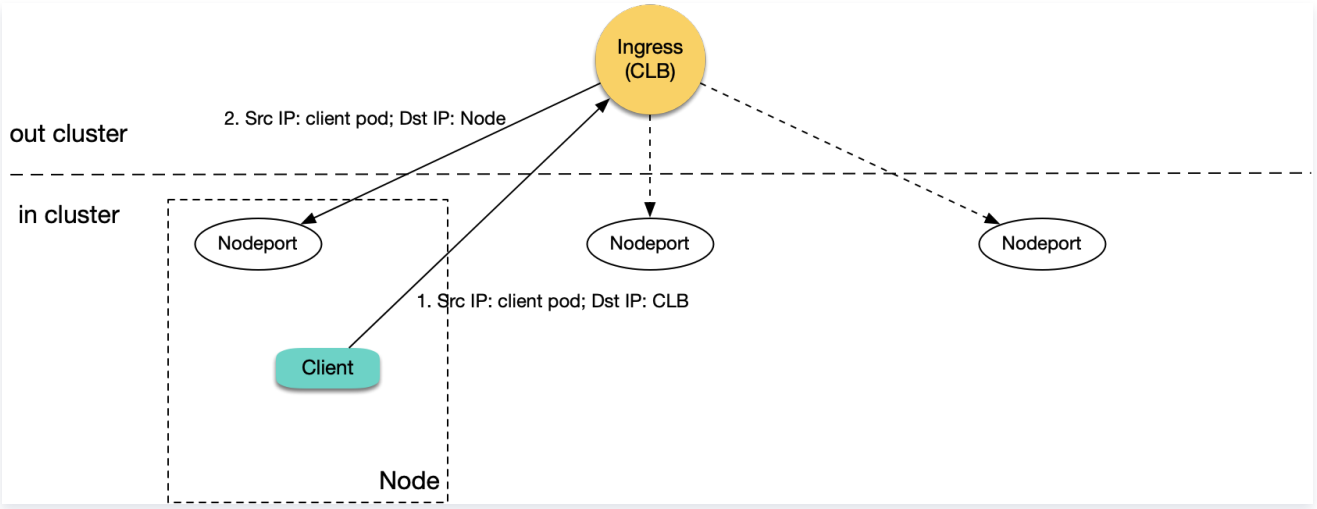
问题原因

问题根本原因在于 CLB 将请求转发到后端 RS 时，报文的源目的 IP 都在同一节点内，而 Linux 默认忽略收到源 IP 是本机 IP 的报文，导致数据包在子机内部回环，如下图所示：



分析七层回环问题

使用 TKE CLB 类型的 Ingress，会为每个 Ingress 资源创建一个 CLB 以及80、443的7层监听器规则（HTTP/HTTPS），并为 Ingress 每个 location 绑定对应 TKE 各个节点某个相同的 NodePort 作为 RS（每个 location 对应一个 Service，每个 Service 都通过各个节点的某个相同 NodePort 暴露流量），CLB 根据请求匹配 location 转发到相应的 RS（即 NodePort），流量经过 NodePort 后会再经过 Kubernetes 的 iptables 或 ipvs 转发给对应的后端 Pod。集群中的 Pod 访问本集群的内网 Ingress，CLB 将请求转发给其中一台节点的对应 NodePort：



如上图，当被转发的这台节点恰好也是发请求的 Client 所在节点时：

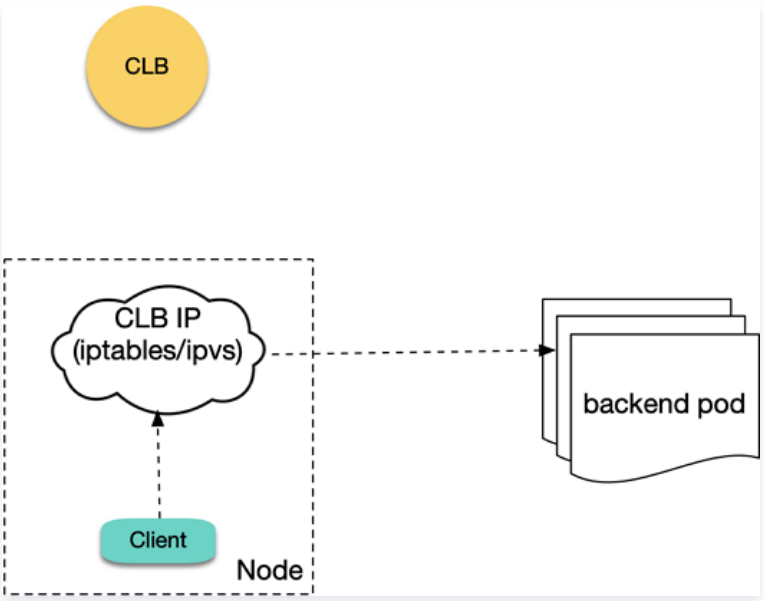
1. 集群中的 Pod 访问 CLB，CLB 将请求转发到任意一台节点的对应 NodePort。
2. 报文到达 NodePort 时，目的 IP 是节点 IP，源 IP 是 Client Pod 的真实 IP，因为 CLB 不进行 SNAT，会将真实源 IP 透传过去。
3. 由于源 IP 与目的 IP 都在这台机器内，因此将导致回环，CLB 将收不到来自 RS 的响应。

访问集群内 Ingress 的故障现象大多为几秒延时，原因是7层 CLB 如果请求 RS 后端超时（大概4s），会重试下一个 RS，所以如果 Client 设置的超时时间较长，出现回环问题的现象就是请求响应慢，有几秒的延时。如果集群只有一个节点，CLB 没有可以重试的 RS，则现象为访问不通。

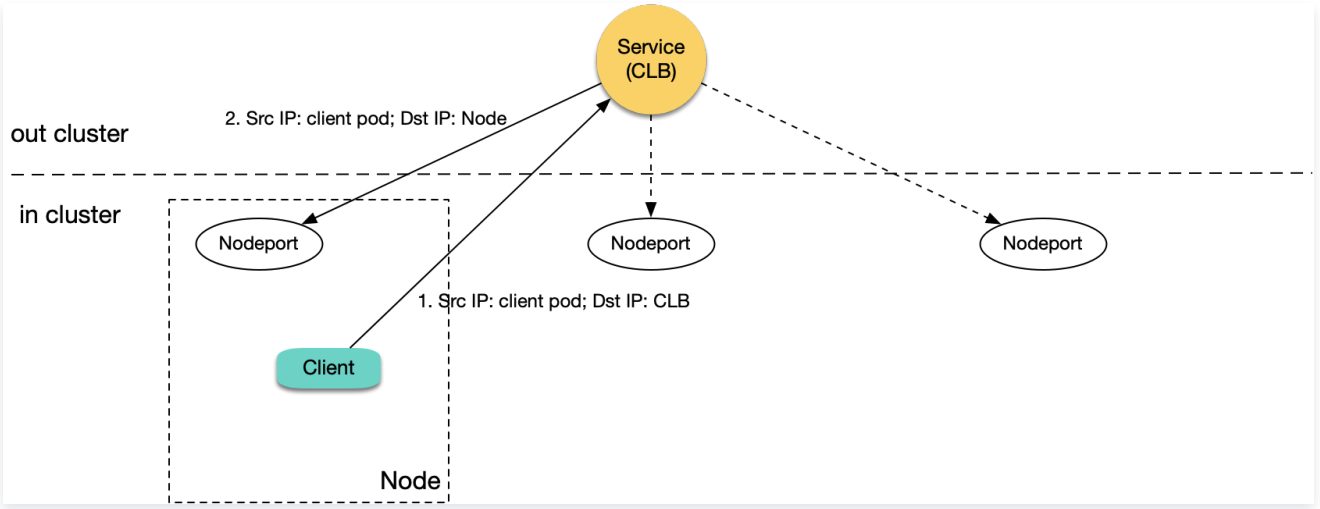
分析四层回环问题

当使用 LoadBalancer 类型的内网 Service 时暴露服务时，会创建内网 CLB 并创建对应的4层监听器（TCP/UDP）。当集群内 Pod 访问 LoadBalancer 类型 Service 的 `EXTERNAL-IP` 时（即 CLB IP），原生 Kubernetes 实际上不会去真正访问 LB，而是直接通过 iptables 或 ipvs

转发到后端 Pod（不经过 CLB），如下图所示：



因此原生 Kubernetes 的逻辑不存在回环问题。但在 TKE 的 ipvs 模式下，Client 访问 CLB IP 的包会真正到 CLB，如果在 ipvs 模式下，Pod 访问本集群 LoadBalancer 类型 Service 的 CLB IP 将存在回环问题，情况跟上述内网 Ingress 回环类似，如下图所示：



不同的是，四层 CLB 不会重试下一个 RS，当遇到回环时，现象通常是联通不稳定。如果集群只有一个节点，那将导致完全不通。

为什么 TKE 的 ipvs 模式不用原生 Kubernetes 类似的转发逻辑（不经过 LB，直接转发到后端 Pod）？

背景是因为以前 TKE 的 ipvs 模式集群使用 LoadBalancer 内网 Service 暴露服务，内网 CLB 对后端 NodePort 的健康探测会全部失败，原因如下：

- ipvs 主要工作在 INPUT 链，需要将要转发的 VIP（Service 的 Cluster IP 和 EXTERNAL-IP）当成本机 IP，才好让报文进入 INPUT 链交给 ipvs 处理。
- kube-proxy 的做法是将 Cluster IP 和 EXTERNAL-IP 都绑到名称为 `kube-ipvs0` 的 dummy 网卡，该网卡仅仅用来绑定 VIP (内核自动为其生成 local 路由)，不用于接收流量。
- 内网 CLB 对 NodePort 的探测报文源 IP 是 CLB 自身的 VIP，目的 IP 是 Node IP。当探测报文到达节点时，节点发现源 IP 为本机 IP（因为其被绑定到了 `kube-ipvs0`），就将其丢弃掉。所以 CLB 的探测报文永远无法收到响应，也就全部探测失败，虽然 CLB 有全死全活逻辑（全部探测失败视为全部可以被转发），但也相当于探测未起到任何作用，在某些情况下将造成一些异常。

为解决上述问题，TKE 的修复策略是：ipvs 模式不绑定 EXTERNAL-IP 到 `kube-ipvs0` 。也就是集群内 Pod 访问 CLB IP 的报文不会进入 INPUT 链，而是直接出节点网卡，真正到达 CLB，这样健康探测的报文进入节点时将不会被当成本机 IP 而丢弃，同时探测响应报文也不会进入 INPUT 链导致出不去。

虽然这种方法修复了 CLB 健康探测失败的问题，但也导致集群内 Pod 访问 CLB 的包真正到了 CLB，由于访问集群内的服务，报文又会被转发回其中一台节点，也就存在了回环的可能性。

> **说明：**

> 相关问题已提交至 社区，但目前还未有效解决。

常见问题

为什么公网 CLB 不存在回环问题?

使用公网 Ingress 和 LoadBalancer 类型公网 Service 不存在回环问题，主要是公网 CLB 收到的报文源 IP 是子机的出口公网 IP，而子机内部无法感知自己的公网 IP，当报文转发回子机时，不认为公网源 IP 是本机 IP，也就不存在回环。

CLB 是否有避免回环机制?

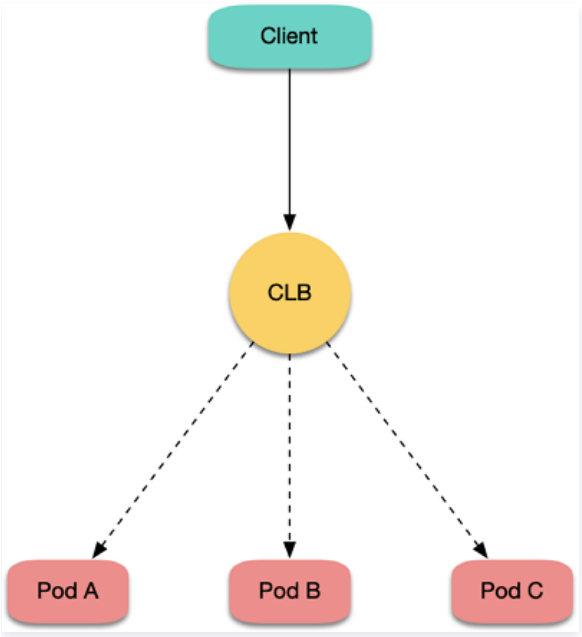
有。CLB 会判断源 IP，如果发现后端 RS 也有相同 IP，就不考虑转发给这个 RS，而是选择其他 RS。但是源 Pod IP 跟后端 RS IP 并不相同，CLB 也不知道这两个 IP 是在同一节点，所以同样可能会转发过去，也就可能发生回环。

Client 与 Server 反亲和部署能否规避?

如果将 Client 与 Server 通过反亲和性部署，避免 Client 跟 Server 部署在同一节点，能否规避 CLB 回环问题?
默认情况下，LB 通过节点 NodePort 绑定 RS，可能转发给任意节点 NodePort，此时 Client 与 Server 是否在同一节点都可能发生回环。但如果为 Service 设置 `externalTrafficPolicy: Local`，LB 就只会转发到有 Server Pod 的节点，如果 Client 与 Server 通过反亲和调度在不同节点，则此时不会发生回环，所以反亲和 + `externalTrafficPolicy: Local` 可以规避回环问题（包括内网 Ingress 和 LoadBalancer 类型内网 Service）。

VPC-CNI 的 LB 直通 Pod 是否也存在 CLB 回环问题?

TKE 通常用的 Global Router 网络模式（网桥方案），还有一种是 VPC-CNI（弹性网卡方案）。目前 LB 直通 Pod 只支持 VPC-CNI 的 Pod，即 LB 不绑 NodePort 作为 RS，而是直接绑定后端 Pod 作为 RS，如下图所示：



这样即可绕过 NodePort，不再像之前一样可能会转发给任意节点。但如果 Client 与 Server 在同一节点，同样可能会发生回环问题，通过反亲和可以规避。

有什么建议?

反亲和与 `externalTrafficPolicy: Local` 的规避方式不太优雅。一般来讲，访问集群内的服务避免访问本集群的 CLB，因为服务本身在集群内部，从 CLB 绕一圈不仅会增加网络链路的长度，还会引发回环问题。
访问集群内服务尽量使用 Service 名称，例如 `server.prod.svc.cluster.local`，通过这样的配置将不会经过 CLB，也不会导致回环问题。
如果业务有耦合域名，不能使用 Service 名称，可以使用 coredns 的 rewrite 插件，将域名指向集群内的 Service，coredns 配置示例如下：

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
```

```
  Corefile: |2-
        .:53 {
            rewrite name roc.example.com server.prod.svc.cluster.local
        ...
```

如果多个 Service 共用一个域名，可以自行部署 Ingress Controller (例如 nginx-ingress)：

1. 用上述 rewrite 的方法将域名指向自建的 Ingress Controller。
2. 将自建的 Ingress 根据请求 location (域名+路径) 匹配 Service，再转发给后端 Pod。整段链路不经过 CLB，同样能规避回环问题。

Service&Ingress 常见报错和处理

最近更新时间：2025-03-26 11:18:42

Kubernetes 通过声明式的方式管理资源，声明式 API 只需要声明一个期望的状态，系统就会自行调节以满足该状态。但声明式 API 也引入新的问题：无法感知资源当前状态信息，对任务的流程把握不够清晰。
为了保证负载均衡实例配置信息的一致性，整个 Service/Ingress 是作为一个资源整体进行同步的。若 CLB 类型的 Service/Ingress 有任何监听器级别的配置错误问题，会导致整个负载均衡同步失败，并以 Event 的形式反馈问题让用户进行处理。资源同步正确时，也会有该资源正常完成同步的 Event 更新。
Service/Ingress 作为用户直接对外提供服务的资源对象，如果异常将导致服务不可用，影响服务质量。本文提供了常见的 Service/Ingress 报错原因和处理办法。

如何查看 Service/Ingress Event 的报错信息?

通过控制台

1. 登录 容器服务控制台，选择左侧导航栏中的**集群**。
2. 在**集群管理**页面中，选择需要更新 YAML 的集群 ID，进入集群基本信息页面。
3. 选择**服务与路由** > **Service** 或 **Ingress**，进入 Service 或 Ingress 信息页面。
4. 单击某个具体的 Service 或 Ingress 名称。
5. 在**事件**页签，可查看当前 Service 或 Ingress 的事件信息。如下图所示，可以查看当前 Service/Ingress 的 Event 报错代码。

> **注意：**
> 资源事件只保存最近 1 小时内发生的事件，请尽快查阅。

通过命令行

- 获取异常 Ingress 资源列表和报错信息：

```
kubectl get event | grep ingress
```

- 获取异常 Service 资源列表和报错信息：

```
kubectl get event | grep service
```

Service Event 报错原因和处理办法

错误码	错误内容	解决方案	不修改可能存在的风险
E4001	TKE_QCSRoles 授权	登录访问管理控制台，检查 TKE 服务账户授权，并重新添加授权。详情可参见 服务授权相关角色权限说明 。	集群维度，组件不能正常工作
E4004	负载均衡数量超出上限	提交工单 申请负载均衡数量 Quota。	新增资源没有流量接入

E4005	负载均衡创建参数错误	检查创建参数：service.kubernetes.io/service.extensiveParameters，详情可参见 Service Annotation 说明 。	新增资源没有流量接入
E4008	子网 IP 不足	三种方案： 1. 更换其他有足够 IP 的子网新建子网。 2. 更新 Service 注解使用新的子网 ID。 3. 改为使用公网类型的负载均衡。	新增资源没有流量接入
E4009	欠费	您需要充值账户。	新增资源没有流量接入
E4011	使用已有负载均衡不存在	登录负载均衡控制台，找到与当前集群相同的 VPC 下的负载均衡实例，确认负载均衡 ID，使用一个真实有效负载均衡 ID，详情可参见 Service 使用已有 CLB 。	新增资源没有流量接入
E4012	使用已有负载均衡是其他 TKE 管理的资源	使用已有负载均衡必须是用户自己在负载均衡控制台新建的，详情可参见 Service 使用已有 CLB 。	新增资源没有流量接入
E4013	使用已有负载均衡是其他集群使用的资源	不支持跨集群使用负载均衡，使用其他负载均衡或者删除该资源。详情可参见 多 Service 复用 CLB 。	新增资源没有流量接入
E4014	使用已有负载均衡存在端口冲突	多个 Service 声明使用了同一个端口，修改报错 Service 的端口声明，使用其他端口避免冲突，详情可参见 Service 使用已有 CLB 。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4016	用户未开通复用功能	申请开白 Service 复用能力，详情可参见 多 Service 复用 CLB 。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4026	外挂配置未找到	不阻塞用户配置，详情可参见 Service 负载均衡配置 ，有两种方案： 1. 删除 Service 中该外挂配置功能。 2. 注解添加对应名称的 TkeServiceConfig 资源。	暂无
E4033	开启直连，但是没有工作负载后端支持直连接入	工作负载使用弹性网卡网络模式，并且关闭 HostNetwork 删除直连注解，使用 NodePort 接入，直连 Service 使用说明请参见 使用 LoadBalancer 直连 Pod 模式 Service 。	后端更新可能失败，用户滚动更新时可能导致断流。
E4036	后端四元组冲突	负载均衡 VIP、监听器协议、后端 IP、后端端口，四元组必须保持唯一。负载均衡限制，用户需要在 Pod 上监听多个端口，分别进行绑定解决这个问题。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4037	子网不存在	三种方案： 1. 更换其他有足够 IP 的子网新建子网。 2. 更新 Service 注解使用新的子网 ID。 3. 改为使用公网类型的负载均衡。	新增资源没有流量接入
E4062	证书过期	证书服务添加新的证书，并更新扩展协议注解中声明的 Secret 资源内容，按照文档格式填写证书 ID，详情可参见 Service 扩展协议 。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4075	跨地域功能，地域 ID 错误	检查 Service 中的跨地域注解，地域 ID 可参见 地域与可用区 。	新增资源没有流量接入

Ingress Event 报错原因和处理办法

错误码	错误内容	解决方案	不修改可能存在的风险
E4003	负载均衡数量到上限	提交工单 申请负载均衡数量 Quota。	新增资源没有流量接入
E4005	转发规则数量到上限	提交工单 申请负载均衡数量 Quota。	新增资源没有流量接入
E4008	TKE_QCSRole 授权被删除	登录访问管理控制台，检查 TKE 服务账户授权，并重新添加授权。详情可参见 服务授权相关角色权限说明 。	集群维度，组件不能正常工作

E4009	TLS 字段未配置 Secret 名称	若您需要 HTTPS 协议的转发规则，则需要修改 Ingress 里的 TLS 字段，配置 HTTPS 监听器需要的证书。详情可参见 Ingress 证书配置。若您不需要 HTTPS 协议的转发规则，删除 TLS 字段，使用 HTTP 协议进行服务暴露。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4010	TLS 字段配置 Secret 无法找到	新建 Ingress 中声明的 Secret 资源，并按照文档格式填写证书 ID，详情可参见 Ingress 证书配置。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4011	TLS 字段配置 Secret 内容错误，没有证书 ID	更新 TLS 中声明的 Secret 资源内容，按照文档格式填写证书 ID，详情可参见 Ingress 证书配置。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4012	证书状态异常	证书服务添加新的证书，并更新 TLS 中声明的 Secret 资源内容，按照文档格式填写证书 ID，详情可参见 Ingress 证书配置。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4018	指定已有负载均衡不存在	到负载均衡控制台找到与当前集群相同的 VPC 下的 负载均衡实例，确认负载均衡 ID，使用一个真实有效负载均衡 ID，详情可参见 Ingress 使用已有 CLB。	新增资源没有流量接入
E4019	指定已有负载均衡是 TKE 创建的	使用已有负载均衡必须是用户自己在负载均衡控制台新建的，详情可参见 Ingress 使用已有 CLB。	新增资源没有流量接入
E4020	指定已有负载均衡是其他 Ingress 使用的	使用已有负载均衡必须是用户自己在负载均衡控制台新建的，详情可参见 Ingress 使用已有 CLB。	新增资源没有流量接入
E4021	指定已有负载均衡监听器未排空	登录负载均衡控制台，删除该负载均衡所有的监听器。	新增资源没有流量接入
E4022	kubernetes.io/ingress.http-rules，注解格式不正确	参考 Ingress Annotation 说明，确认注解内容是否合法，建议使用控制台对资源进行更新。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4023	kubernetes.io/ingress.https-rules，注解格式不正确	参考 Ingress Annotation 说明，确认注解内容是否合法，建议使用控制台对资源进行更新。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4027	账户欠费	您需要充值账户。	新增资源没有流量接入
E4031	转发规则存在不合法字符	修改转发规则 Rule 字段。负载均衡转发路径不支持正则。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4034	IPv6 CLB，未声明 host 字段。（IPv4 可以不填 Host，默认为 VIP，IPv6 不支持）	补充 Ingress 中所有 Host 字段，不能留空。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4035	证书 ID 格式错误	更新 TLS 中声明的 Secret 资源内容，按照文档格式填写证书 ID，详情可参见 Ingress 证书配置。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4039	证书过期	证书服务添加新的证书，并更新 TLS 中声明的 Secret 资源内容，按照文档格式填写证书 ID，详情可参见 为 TKE Ingress 证书续期。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4040	Ingress 存在域名没有声明对应证书。	修改 TLS 字段， 配置 HTTPS 监听器需要的证书，详情可参见 Ingress 证书配置。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4041	Ingress 中转发规则指定的 Service 不存在。	如果您的 Service 确实不存在，需要删除 Ingress 中使用该 Service 的转发规则如果您需要使用该 Service，则在与 Ingress 相同的命名空间下，新建该名称 Service 资源。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4042	Ingress 中转发规则指定的 Service 不存在对应转发端口	如果您的 Service 确实不存在这样的端口，需要删除 Ingress 中使用该 Service 的转发规则如果是端口配置问题，需要更新成新端口。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4043	Ingress 指定绑定的 TkeServiceConfig 资源不存在	不阻塞用户配置，详情可参见 Ingress 使用 TkeServiceConfig 配置 CLB，有两种方案：删除 Service 中该外挂配置功能注解添加对应名称的 TkeServiceConfig 资源。	暂无

E4044	kubernetes.io/ingress.rule-mix，填了非合法值	改成 true 或 false，详情可参见 Ingress 混合使用 HTTP 及 HTTPS 协议 。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4046	资源带宽注解配置错误，格式错误或带宽范围错误	带宽合法值 1-2048	新增资源没有流量接入
E4047	Ingress 中转发规则指定的 Service 为 ClusterIP 类型，没有转发端口接入	修改报错 Service 为 NodePort 类型	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4048	Ingress 存在域名声明了多个默认证书	TLS 字段中声明了多个没有 Host 配置的 Secret。删除至只剩一个。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4049	Ingress 存在某个固定域名声明了多个证书	TLS 字段中有一个域名声明了多个 Secret。删除至只剩一个。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4051	用户手动外挂配置指定了系统自动化生成的配置	详情可参见 Ingress 使用 TkeServiceConfig 配置 CLB，改用其他名称的资源。	暂无
E4052	Ingress 中转发规则指定的域名不符合正则要求	检查并修改错误域名一般错误原因域名不带"."，如 Host: test 域名用大写，如 Host: Test.com 正则： `(\*|[a-z0-9]([-a-z0-9]*[a-z0-9])?)(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)+`	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4053	子网 IP 用尽，无法创建负载均衡	三种方案： 1. 更换其他有足够 IP 的子网新建子网。 2. 更新 Service 注解使用新的子网 ID。 3. 改为使用公网类型的负载均衡。	新增资源没有流量接入
E4054	后端数量达到上限	提交工单 申请负载均衡后端数量 Quota。	后端更新可能失败，用户滚动更新时可能导致断流
E4055	子网不存在或格式错误，无法创建负载均衡	三种方案： 1. 更换其他有足够 IP 的子网新建子网。 2. 更新 Service 注解使用新的子网 ID。 3. 改为使用公网类型的负载均衡。	新增资源没有流量接入
E4060	账户未开白，无法为用户开启 SNAT Pro 功能	提交工单 申请负载均衡开白 SNAT Pro 能力。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4066	集群初始化失败，CRD 无法创建	用户集群存在问题，需要 提交工单 。	集群维度，组件不能正常工作
E4068	自动重定向规则和用户其他声明规则冲突	使用自动重定向功能建议不要再声明其他转发规则，详情可参见 Ingress 重定向 。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4071	跨地域配置错误，CLB 所在 VPC 和当前集群所在 VPC 不在同一个云联网中	使用云联网关联两个 VPC 更换其他云联网内的 VPC 详情可参见 Ingress 跨域绑定 。	新增资源没有流量接入
E4074	节点欠费可能，后端绑定失败	CLB 后端绑定问题，可能节点遭到封锁。	后端更新可能失败，用户滚动更新时可能导致断流
E4081	kubernetes.io/ingress.https-rules，注解格式不正确（配置冲突）	建议通过控制台修改配置，详情可参见 Ingress Annotation 说明 。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4082	NoSelector Service 不支持绑定，Ingress 在直连场景下，声明使用了类似资源	NoSelector Service 后端不支持直连接入，需要改用 NodePort。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。

E4084	跨域绑定 1.0 方案下，不能使用 SNAT Pro 功能	系统限制，需要调整技术方案。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。
E4098	指定已有负载均衡，负载均衡 ID 格式错误	登录负载均衡控制台找到与当前集群相同的 VPC 下的负载均衡实例，确认负载均衡 ID，使用一个真实有效负载均衡 ID，详情可参见 Ingress 使用已有 CLB 。	新增资源没有流量接入
E4101	指定已有负载均衡监听器存在冲突	检查 80/443 端口是否已经被其他资源占用。	资源同步被阻塞，用户更新可能导致负载均衡后端不能正常更新。

Nginx Ingress 偶现 Connection Refused

最近更新时间：2024-08-01 18:16:11

现象描述

当用户在使用 Nginx Ingress 时，减少 Nginx Ingress Controller 副本过程中，可能出现 Connection Refused 的问题，此时 CLB 批量解绑 RS，TCP/UDP 监听器存量连接停止转发。

可能原因

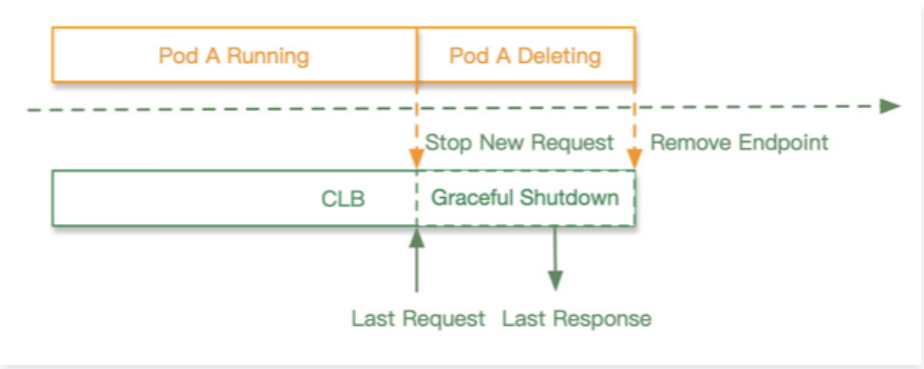
查看 Nginx Ingress 源码，Nginx Ingress Controller 工作负载没有优雅停机的能力， Pod 在收到 kill signal 之后直接退出。如下图所示：

```
func main() {
	err := exec.Command("bash", "-c", "pkill -SIGTERM -f nginx-ingress-controller").Run()
	if err != nil {
		klog.ErrorS(err, "terminating ingress controller")
		os.Exit(1)
	}

	// wait for the NGINX process to terminate
	timer := time.NewTicker(time.Second * 1)
	for range timer.C {
		if !nginx.IsRunning() {
			timer.Stop()
			break
		}
	}
}
```

解决思路

若您使用 TKE Service 优雅停机 的能力，当 Pod 需要被删除时，Pod 能够处理完已接受到的请求，此时入流量关闭，但出流量仍能走通。直到处理完所有已有请求和 Pod 真正删除时，出入流量才进行关闭。待到优雅停机时间结束后，Pod 才被真正的删除。如下图所示：



处理步骤

> **注意：**
> 仅针对 直连场景 生效，请检查您的集群是否支持直连模式。

步骤1

在 kube-system 命名空间下名为 ****-ingress-nginx-controller 的 Service 里使用 Annotation 标明使用优雅停机。
以下为使用 Annotation 标明使用优雅停机示例，完整 Service Annotation 说明可参见 Service Annotation 说明 。

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

```
    service.cloud.tencent.com/direct-access: "true" ## 开启直连 Pod 模式
    service.cloud.tencent.com/enable-grace-shutdown: "true"  # 表示使用优雅停机
  name: my-service
spec:
  selector:
    app: MyApp
```

步骤2

在 kube-system 命名空间下名为 ****-ingress-nginx-controller 的 Deployment 里的 wait-shutdown 前面加一段时间的 sleep。示例如下：

```
 spec:
  template:
    spec:
      containers:
      - name: controller
        lifecycle:
          preStop:
            exec:
              command:
              - /bin/sh
              - -c
              - sleep 30 && /wait-shutdown      # 在这一行添加 sleep 时间，此处以 30s 为例，具体可按业务请求耗时调
整
```

步骤3

相应增大 terminationGracePeriodSeconds 时长，默认是 30s。在此处示例中 sleep 30，则需相应调大 terminationGracePeriodSeconds 至 60s 即可。
更多内容请参考 Service 优雅停机 。如果通过以上步骤仍无法解决您的问题，您可以 联系我们 反馈和解决问题。

CLB Ingress 创建报错排障处理

最近更新时间：2024-03-25 15:41:12

现象描述

创建 CLB 类型的 Ingress 报错，错误码 `E6009` 。如下图所示：

```
Address:          10.8.4.119
Default backend:  default-http-backend:80 (<error: endpoints "default-http-backend" not found>)
TLS:
  wildcard-pagoda-2020 terminates
Rules:
  Host                    Path  Backends
  ----                    ----  --------
  cas.pagoda.com.cn
                          /   cas-api-nodeport:8080 (10.12.27.229:8080,10.12.89.129:8080,10.12.96.32:8080)
  cas-verify.pagoda.com.cn
                          /   cas-api-verify-nodeport:8080 (10.12.4.160:8080)
Annotations:              field.cattle.io/publicEndpoints:
                            [{"addresses":["10.8.4.119"],"port":80,"protocol":"HTTP","serviceName":"cas:cas-api-nodeport","ingressName":"cas:cas-internal","hostname":...
                          ingress.cloud.tencent.com/direct-access: false
                          kubernetes.io/ingress.class: qcloud
                          kubernetes.io/ingress.http-rules:
                            [{"host":"cas.pagoda.com.cn","path":"/","backend":{"serviceName":"cas-api-nodeport","servicePort":"8080"}},{"host":"cas-verify.pagoda.com....
                          kubernetes.io/ingress.https-rules:
                            [{"host":"cas.pagoda.com.cn","path":"/","backend":{"serviceName":"cas-api-nodeport","servicePort":"8080"}},{"host":"cas-verify.pagoda.com....
                          kubernetes.io/ingress.qcloud-loadbalance-id: lb-4ymlu8dc
                          kubernetes.io/ingress.rule-mix: true
                          kubernetes.io/ingress.subnetId: subnet-ewhttulw
                          qcloud_cert_id: rUWS5Gd4
Events:
  Type     Reason                 Age                From                  Message
  ----     ------                 ----               ----                  -------
  Warning  EnsureServiceFailed    15m (x61 over 74m)  loadbalancer-controller  IngressError. ErrorCode: E6009 Details: UnexpectedError. OriginError: Internal error occurred: failed calling webhook "validate.nginx.ingress.kubern
etes.io": an error on the server ("") has prevented the request from succeeding
  Normal   EnsureIngressSuccess  6s (x4 over 6m3s)   loadbalancer-controller  Ingress Sync Success. RetrunCode: S2000
```

可能原因

Nginx Ingress 社区1.0.0之前的版本，不支持 networking.k8s.io/v1 类型资源的 Validating Webhook 回调。需要在负责验证的 CRD 里面，去掉 v1类型资源的验证。

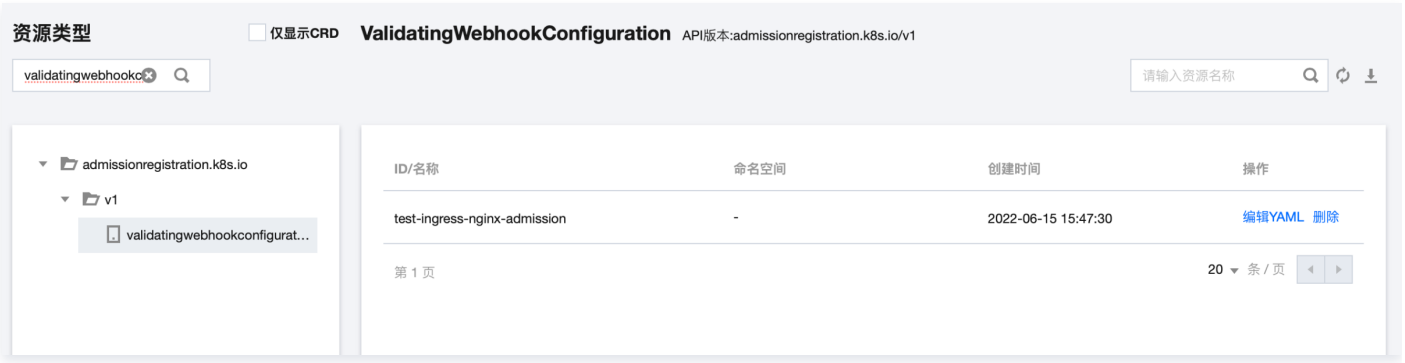
解决思路

您可参考以下两种方法处理问题：

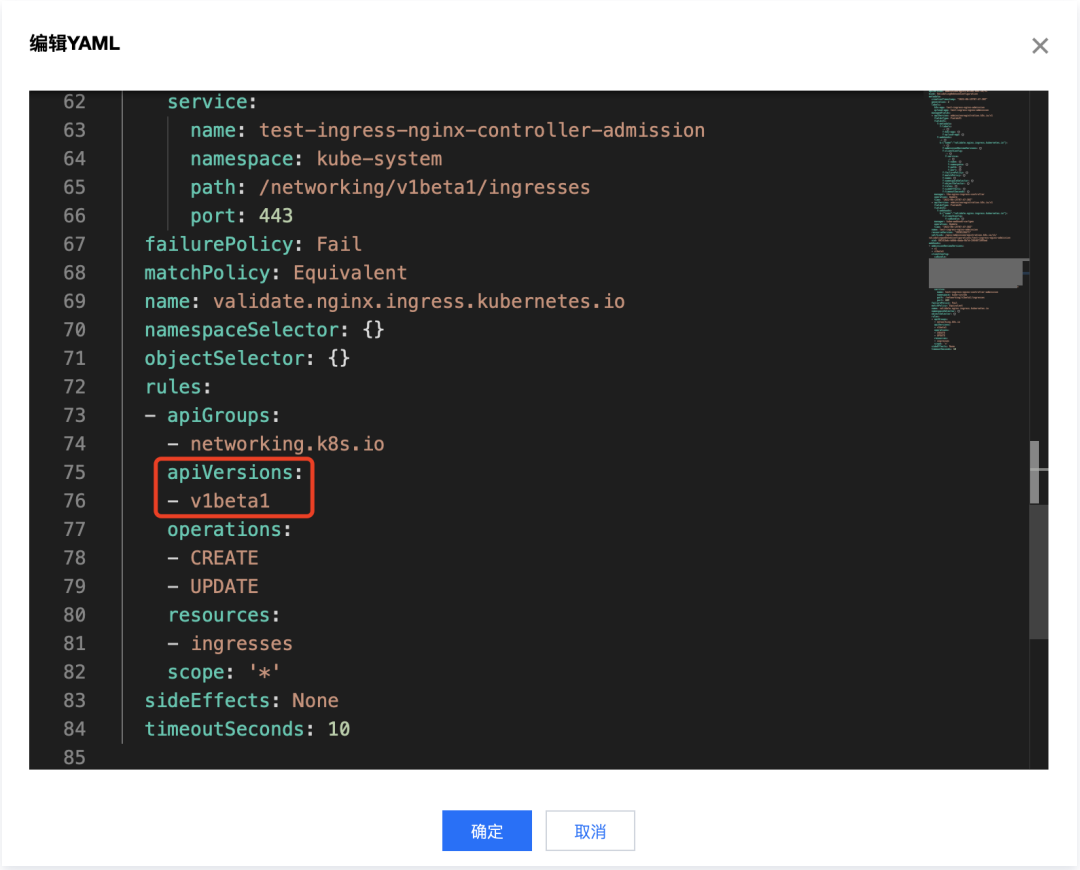
方法1：取消 v1类型资源的验证

将 validatingwebhookconfigurations 类型资源的 webhooks.rules 的 apiVersions 字段调整为 v1beta1。

1. 登录 容器服务控制台，选择集群所在地域。
2. 在“集群管理”列表中，单击集群名称，进入集群详情页。
3. 选择左侧导航中的“资源对象浏览器”，并在资源类型页中搜索： `validatingwebhookconfigurations` 。如下图所示：



4. 在搜索结果中选择 `validatingwebhookconfigurations` ，单击资源对象列表右侧的**编辑YAML**，检查每一个资源对象的 webhooks.rules 的 apiVersions 字段是否为 v1beta1。



5. 升级组件。上述步骤解决的是存量 Nginx Ingress 实例资源验证的问题，要避免新增实例出现类似的问题，需要升级 Nginx Ingress 扩展组件。升级组件步骤如下：

   5.1 在集群详情页，选择左侧导航中的**组件管理**。

   5.2 单击 Nginx Ingress 右侧的**升级**，将 Nginx Ingress 升级到1.1.0版本。

方法2：取消资源的验证

1. 登录 容器服务控制台，选择集群所在地域。
2. 在“集群管理”列表中，单击集群名称，进入集群详情页。
3. 选择左侧导航中的“资源对象浏览器”，并在资源类型页中搜索：`validatingwebhookconfigurations`。
4. 在搜索结果中选择 `validatingwebhookconfigurations`，单击资源对象列表右侧的**删除**。
5. 升级组件。上述步骤解决的是存量 Nginx Ingress 实例资源验证的问题，要避免新增实例出现类似的问题，需要升级 Nginx Ingress 扩展组件。升级组件步骤如下：

   5.1 在集群详情页，选择左侧导航中的**组件管理**。

   5.2 单击 Nginx Ingress 右侧的**升级**，将 Nginx Ingress 升级到1.1.0版本。

Pod 网络无法访问排查处理

最近更新时间：2024-01-09 10:40:31

本文档介绍 TKE 集群中多场景下可能发生的常见网络问题，并给出对应的排查思路。当遇到此类问题时，建议您首先按照下文中的检查建议进行排查，若确认检查项无误后仍不能正常访问，请您 联系我们 寻求帮助。

集群中不同节点上的容器（Pod）无法互访

同一集群中不同节点上的 Pod 可以直接互访，当出现一个节点上 Pod 无法访问其他节点上 Pod 时，建议您进行如下检查：

1. 检查上述不同节点间是否可以互访。
2. 检查节点安全组是否正确放通容器网段和对端节点所在的 VPC 网段或 VPC 子网网段。

同一个 VPC 内的节点与容器（Pod）无法互访

同一个 VPC 内的节点和 Pod 可以直接互访，当出现无法互访的情况时，建议您进行如下检查：

1. 检查对端节点和 Pod 所在节点是否可以互访。
2. Pod 所在节点的安全组是否正确放通对端节点所在的 VPC 子网网段。
3. 对端节点的安全组是否正确放通容器网段。

不同 VPC 内的节点与容器（Pod）或容器（Pod）与容器（Pod）间无法互访

不同 VPC 间的互访需要先通过 云联网 或 对等连接 完成打通。如果打通之后仍出现无法互访的情况，建议您进行如下检查：

1. 检查节点与节点是否可以互访。
2. 检查对端节点的安全组是否正确放通 VPC 网段和容器网段。
3. 检查 Pod 所在节点的安全组是否正确放通对端节点所在的 VPC 网段或 VPC 子网网段。

如果容器（Pod）与容器（Pod）间无法互访，需进一步执行如下检查：

1. 检查 Pod 所在节点的安全组是否正确放通对端 VPC 网段（或者节点所在的 VPC 子网网段）和容器网段。
2. 如需查看 Pod 的源 IP，可执行以下命令，进一步修改 `ip_masq_agent` 的配置，彼此增加对方容器网段。

```
kubectl -n kube-system edit configmap ip-masq-agent-config
```

IDC 与容器（Pod）无法访问

IDC 与 Pod 互访需要先通过 云联网 或 专线网关 完成打通。如果打通之后仍出现无法互访的情况，建议您进行如下检查：

- 检查 IDC 防火墙是否放通容器网段和 CVM 网段。
- 检查 CVM 安全组是否放通 IDC 网段。
- 检查 IDC 是否使用 BGP 协议：
  - 若未使用 BGP 协议，则需在 IDC 内配置访问容器网段下一跳路由到专线网关。
  - 若使用 BGP 协议，则会自动同步，通常不需要任何配置。除非 IDC 有特殊的静态配置，否则可联系运维人员配置访问容器网段下一跳到专线网关。

> **说明**
> - 如需在 IDC 中查看 Pod 的 IP，则需要放通 IDC 网段。
> - 默认情况下，除 VPC 之外的包会经过 SNAT 处理转换为 NodeIP。放通 IDC 网段时需配置为不经过 SNAT 处理。
> - 放通 IDC 网段的方法为：执行命令 `kubectl -n kube-system edit configmap ip-masq-agent-config`，修改 ip-masq-agent 配置，并将 IDC 网段添加到 NonMasqueradeCIDRs 列表中。

相关操作

查看节点上的 iptables 或 ipvs 转发规则

您可以通过执行以下命令，查看节点上的 iptables 或者 ipvs 转发规则。

- 执行以下命令，检查 iptables 转发规则。

```
iptables-save
```

- 执行以下命令，检查 ipvs 转发规则。

```
ipvsadm -Ln -t/-u ip:port
```

抓包命令

以下抓包命令可用于分析集群内不同节点上的容器（Pod）互访不通的情况。

- 执行以下命令，在源 Pod 的节点上抓网卡 eth0 的包。

```
tcpdump -nn -vv -i eth0 host <对端pod-ip>
```

- 执行以下命令，在对端 Pod 的节点上抓网卡 eth0 的包。

```
tcpdump -nn -vv -i eth0 host <源pod-ip>
```

- 执行以下命令，在对端 Pod 的 netns 中抓网卡 eth0 的包。

```
tcpdump -nn -vv -i eth0 port <请求的端口号>
```

容器内抓包定位网络问题

在使用 Kubernetes 运行应用的时候，可能会遇到一些网络问题，其中比较常见的是服务端无响应（超时）及回包内容不正常。如果您无法在相关配置上定位到问题点，则需要确认数据包最终是否被路由到容器里，或者报文到达容器的内容和出容器的内容是否符合预期，并通过分析报文进一步缩小问题范围。本文提供脚本，可一键进入容器网络命名空间（netns），并使用宿主机上的 tcpdump 进行抓包。

使用脚本一键进入 Pod netns 抓包

当发现某个服务不通时，建议将其副本数调为1，并按照以下步骤进行抓包。

1. 执行以下命令，获取该 Pod 副本所在的节点和 Pod 名称。

```
kubectl get pod -o wide
```

2. 登录 Pod 所在节点，将如下脚本粘贴到 Shell，注册函数到当前登录的 Shell。

```
function e() {
    set -eu
    ns=${2-"default"}
    pod=`kubectl -n $ns describe pod $1 | grep -A10 "^Containers:" | grep -Eo 'docker://.*$' | head -n
1 | sed 's/docker:\/\/\(.*\)$/\1/'`
    pid=`docker inspect -f {{.State.Pid}} $pod`
    echo "entering pod netns for $ns/$1"
    cmd="nsenter -n --target $pid"
    echo $cmd
    $cmd
}
```

您可前往 脚本原理 了解并开始使用脚本。

3. 执行以下命令，一键进入 Pod 所在的 netns。

```
e POD_NAME NAMESPACE
```

具体示例如下：

```
e istio-galley-58c7c7c646-m6568 istio-system
e proxy-5546768954-9rxg6 # 省略 NAMESPACE 默认为 default
```

> **说明**
> 进入 Pod 所在的 netns 之后，可在宿主机上执行 `ip a` 或 `ifconfig` 命令来查看容器的网卡，执行 `netstat -tunlp` 命令查看当前容器的监听端口。

4. 执行以下命令，使用 tcpdump 进行抓包。

```
tcpdump -i eth0 -w test.pcap port 80
```

使用 wireshark 分析报文

您可以执行 `Ctrl+C` 操作停止抓包，使用 `scp` 或 `sz` 命令将抓取的包下载到本地使用 `wireshark` 分析。分析过程中，您可能会用到以下常见的 `wireshark` 过滤语法：

- 使用 telnet 连接并发送测试文本，例如 `"lbtest"`。执行以下命令，查看已发送的测试报文是否传递到容器。

```
tcp contains "lbtest"
```

- 如果容器提供 HTTP 服务，则可使用 curl 发送测试路径请求。
  执行以下命令过滤 uri，查看报文是否传递到容器。

```
http.request.uri=="/mytest"
```

脚本原理

- 查看指定 Pod 运行的容器 ID。

```
kubectl describe pod <pod> -n mservice
```

- 获得容器进程的 Pid。

```
docker inspect -f {{.State.Pid}} <container>
```

- 进入该容器的 network namespace。

```
nsenter -n --target <PID>
```

上述脚本依赖的宿主机命名包含有：kubectl、docker、nsenter、grep、head、sed。

查看节点安全组配置

1. 登录容器服务控制台，选择左侧导航栏中的 集群。
2. 选择集群 ID，进入集群详情页。
3. 选择左侧导航栏中的**节点管理** > **节点**。
4. 在“节点列表”页面，选择需查看安全组的节点 ID。
5. 在节点管理页面，选择**详情**页签，并单击“主机信息”中的节点 ID。如下图所示：

6. 在节点“基本信息”页面中，选择**安全组**页签，并在页面中查看该节点的安全组是否正确放通30000-32768端口区间。如下图所示：

Pod 状态异常与处理措施
Pod 异常排查概述

最近更新时间：2024-08-02 17:03:41

为满足一定的业务需求，用户往往需要对容器服务集群进行一系列复杂的自定义配置。而当集群中的 Pod 出现某种异常时，可能一时无法直接通过异常状态准确定位异常原因。
基于以上现象，您可参考 Pod 异常问题 系列文档进行问题排查、定位及解决。

常用命令

排查过程的常用命令如下：

- 查看 Pod 状态：

```
kubectl get pod <pod-name> -o wide
```

- 查看 Pod 的 yaml 配置：

```
kubectl get pod <pod-name> -o yaml
```

- 查看 Pod 事件：

```
kubectl describe pod <pod-name>
```

- 查看容器日志：

```
kubectl logs <pod-name> [-c <container-name>]
```

Pod 状态

下表中列举了 Pod 的状态信息：

状态	描述
Error	Pod 启动过程中发生错误。
NodeLost	Pod 所在节点失联。
Unkown	Pod 所在节点失联或其他未知异常。
Waiting	Pod 等待启动。
Pending	Pod 等待被调度。
ContainerCreating	Pod 容器正在被创建。
Terminating	Pod 正在被销毁。
CrashLoopBackOff	容器退出，Kubelet 正在将它重启。
InvalidImageName	无法解析镜像名称。
ImageInspectError	无法校验镜像。
ErrImageNeverPull	策略禁止拉取镜像。
ImagePullBackOff	正在重试拉取。
RegistryUnavailable	连接不到镜像中心。

ErrImagePull	通用的拉取镜像出错。
CreateContainerConfigError	不能创建 Kubelet 使用的容器配置。
CreateContainerError	创建容器失败。
RunContainerError	启动容器失败。
PreStartHookError	执行 preStart hook 报错。
PostStartHookError	执行 postStart hook 报错。
ContainersNotInitialized	容器没有初始化完毕。
ContainersNotReady	容器没有准备完毕。
ContainerCreating	容器创建中。
PodInitializing	Pod 初始化中。
DockerDaemonNotReady	Docker 还没有完全启动。
NetworkPluginNotReady	网络插件还没有完全启动。

问题定位

您可根据 Pod 的异常状态，选择对应参考文档进一步定位异常原因：

- Pod 一直处于 ContainerCreating 或 Waiting 状态
- Pod 一直处于 ImagePullBackOff 状态
- Pod 一直处于 Pending 状态
- Pod 一直处于 Terminating 状态
- Pod 健康检查失败
- Pod 处于 CrashLoopBackOff 状态
- Pod 无限重启且流量异常
- 容器进程主动退出

Pod 异常排查工具
使用 Systemtap 定位 Pod 异常退出原因

最近更新时间：2023-09-08 19:13:09

本文介绍如何使用 Systemtap 工具定位 Pod 异常问题原因。

准备工作

请对应您使用节点的操作系统，按照以下步骤进行相关软件包安装：

Ubuntu 操作系统

1. 执行以下命令，安装 Systemtap。

```
apt install -y systemtap
```

2. 执行以下命令，检查所需待安装项。

```
stap-prep
```

返回示例结果如下：

```
Please install linux-headers-4.4.0-104-generic
You need package linux-image-4.4.0-104-generic-dbgsym but it does not seem to be available
 Ubuntu -dbgsym packages are typically in a separate repository
 Follow https://wiki.ubuntu.com/DebuggingProgramCrash to add this repository
apt install -y linux-headers-4.4.0-104-generic
```

3. 根据上述返回结果可知目前需要 dbgsym 包，但已有软件源中并不包含，需使用第三方软件源安装。 请执行以下命令，安装 dbgsym。

```
 sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys C8CAB6595FDFF622

 codename=$(lsb_release -c | awk  '{print $2}')
 sudo tee /etc/apt/sources.list.d/ddebs.list << EOF
 deb http://ddebs.ubuntu.com/ ${codename}      main restricted universe multiverse
 deb http://ddebs.ubuntu.com/ ${codename}-security main restricted universe multiverse
 deb http://ddebs.ubuntu.com/ ${codename}-updates  main restricted universe multiverse
 deb http://ddebs.ubuntu.com/ ${codename}-proposed main restricted universe multiverse
 EOF

 sudo apt-get update
```

4. 配置好软件源后，再次运行以下命令。

```
stap-prep
```

返回示例结果如下：

```
Please install linux-headers-4.4.0-104-generic
Please install linux-image-4.4.0-104-generic-dbgsym
```

5. 依次执行以下命令，安装提示所需的包。

```
apt install -y linux-image-4.4.0-104-generic-dbgsym
```

```
apt install -y linux-headers-4.4.0-104-generic
```

CentOS 操作系统

1. 执行以下命令，安装 Systemtap。

```
yum install -y systemtap
```

2. 向软件源 `/etc/yum.repos.d/CentOS-Debug.repo` 配置文件中输入以下内容并保存，本文以默认未安装 `debuginfo` 为例。

```
[debuginfo]
name=CentOS-$releasever - DebugInfo
baseurl=http://debuginfo.centos.org/$releasever/$basearch/
gpgcheck=0
enabled=1
protect=1
priority=1
```

3. 执行以下命令检查所需待安装项，并进行相关项的安装。

> **说明**
> 该命令将会安装 `kernel-debuginfo` 。

```
stap-prep
```

4. 执行以下命令，检查节点是否已安装多个版本 `kernel-devel` 。

```
rpm -qa | grep kernel-devel
```

返回结果如下：

```
kernel-devel-3.10.0-327.el7.x86_64
kernel-devel-3.10.0-514.26.2.el7.x86_64
kernel-devel-3.10.0-862.9.1.el7.x86_64
```

若存在多个版本，则只需保留与当前内核版本相同的版本。假设当前内核版本为 `3.10.0-862.9.1.el7.x86_64` ，结合上述返回结果需执行以下命令删除多余版本。

> **注意**
> - 可通过执行 `uname -r` 命令查看内核版本。
> - 需确保 `kernel-debuginfo` 和 `kernel-devel` 均已安装并且安装版本与当前内核版本相同。

```
rpm -e kernel-devel-3.10.0-327.el7.x86_64 kernel-devel-3.10.0-514.26.2.el7.x86_64
```

问题分析

若 Pod 因异常被停止时，可以使用 Systemtap 来监视进程的信号发送进行问题定位。该过程工作原理如下：

1. Systemtap 将脚本转换为 C 语言代码，调用 gcc 将其编译成 Linux 内核模块并通过 `modprobe` 加载到内核。
2. 根据脚本内容在内核进行 hook。此时即可通过 hook 信号的发送，找出容器进程停止的真正原因。

问题定位

步骤1：获取异常 Pod 中重新自动重启的容器 pid

1. 执行以下命令，获取容器 ID。

```
kubectl describe pod <pod name>
```

返回结果如下：

```
......
Container ID:  docker://5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99cf1c945
......
Last State:     Terminated
 Reason:       Error
 Exit Code:    137
 Started:      Thu, 05 Sep 2019 19:22:30 +0800
 Finished:     Thu, 05 Sep 2019 19:33:44 +0800
```

2. 执行以下命令，通过获取到的 Container ID 反查容器主进程的 pid。

```
docker inspect -f "{{.State.Pid}}" 5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99cf1c945
```

返回结果如下：

```
7942
```

步骤2：根据容器退出状态码缩小排查范围

通过步骤1中返回信息中的 `Exit Code` 可以获取容器上次退出的状态码。本文以137为例，进行以下分析：

- 如果进程是被外界中断信号停止的，则退出状态码将在129 – 255之间。
- 退出状态码为137则表示进程是被 `SIGKILL` 信号停止的，但此时仍不能确定进程停止原因。

步骤3：使用 Systemtap 脚本定位异常原因

假设引发异常的问题可复现，则可以使用 Systemtap 脚本监视容器停止原因。

1. 创建 `sg.stp` 文件，输入以下 Systemtap 脚本内容并保存。

```
global target_pid = 7942
probe signal.send{
 if (sig_pid == target_pid) {
     printf("%s(%d) send %s to %s(%d)\n", execname(), pid(), sig_name, pid_name, sig_pid);
     printf("parent of sender: %s(%d)\n", pexecname(), ppid())
     printf("task_ancestry:%s\n", task_ancestry(pid2task(pid()), 1));
 }
}
```

> **注意**
> 变量 `pid` 的值需替换为 步骤2 中获取的容器主进程 pid，本为以7942为例。

2. 执行以下命令，运行脚本。

```
stap sg.stp
```

当容器进程因异常停止时，脚本可捕捉到事件，并执行输出如下：

```
pkill(23549) send SIGKILL to server(7942)
parent of sender: bash(23495)
task_ancestry:swapper/0(0m0.000000000s)=>systemd(0m0.080000000s)=>vGhyM0(19491m2.579563677s)=>sh(33473m
38.074571885s)=>bash(33473m38.077072025s)=>bash(33473m38.081028267s)=>bash(33475m4.817798337s)=>pkill(3
```

```
3475m5.202486630s)
```

解决方法

通过观察 `task_ancestry` ，可以看到停止进程的所有父进程。例如，此处可以看到一个名为 `vGhyM0` 的异常进程，该现象通常表明程序中存在木马病毒，需要进行相关病毒清理工作以确保容器正常运行。

通过 Exit Code 定位 Pod 异常退出和重启原因

最近更新时间：2024-08-28 16:51:01

本文介绍如何根据 Pod 异常状态信息中的 Exit Code 进一步定位问题。

查看 Pod 异常状态信息

执行以下命令，查看异常 Pod 状态信息。

```
kubectl describe pod <pod name>
```

返回结果如下：

```
Containers:
  kubedns:
    Container ID:  docker://5fb8adf9ee62afc6d3f6f3d9590041818750b392dff015d7091eaaf99cf1c945
    Image:         ccr.ccs.tencentyun.com/library/kubedns-amd64:1.14.4
    Image ID:      docker-pullable://ccr.ccs.tencentyun.com/library/kubedns-
amd64@sha256:40790881bbe9ef4ae4ff7fe8b892498eecb7fe6dcc22661402f271e03f7de344
    Ports:         10053/UDP, 10053/TCP, 10055/TCP
    Host Ports:    0/UDP, 0/TCP, 0/TCP
    Args:
      --domain=cluster.local.
      --dns-port=10053
      --config-dir=/kube-dns-config
      --v=2
    State:          Running
      Started:      Tue, 27 Aug 2019 10:58:49 +0800
    Last State:     Terminated
      Reason:       Error
      Exit Code:    255
      Started:      Tue, 27 Aug 2019 10:40:42 +0800
      Finished:     Tue, 27 Aug 2019 10:58:27 +0800
    Ready:          True
    Restart Count:  1
```

在返回结果的容器列表 `Last State` 字段中， `Exit Code` 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。

退出状态码说明

状态码需在0 – 255之间。

- 0表示正常退出。
- 若因外界中断导致程序退出，则状态码区间为129 – 255。例如，操作系统给程序发送中断信号 `kill -9` 或 `ctrl+c` ，导致程序状态变为 `SIGKILL` 或 `SIGINT` 。
- 通常因程序自身原因导致的异常退出，状态码区间在1 – 128。在某些场景下，也允许程序设置使用129 – 255区间的状态码。

若指定的退出状态码不在0 – 255之间（例如，设置 `exit(-1)` ），此时将会自动执行转换，最终呈现的状态码仍会在0 – 255之间。
若将退出时状态码记为 `code` ，则不同情况下转换方式如下：

- 当指定的退出时状态码为负数，转换公式为：

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

- 当指定的退出时状态码为正数，转换公式为：

  ```
  code % 256
  ```

常见异常状态码

- **137**：表示程序被 `SIGKILL` 中断信号杀死。异常原因可能为：
  - 通常是由于 Pod 中容器内存达到了其资源限制（ `resources.limits` ）。例如，内存溢出（OOM）。由于资源限制是通过 Linux 的 cgroup 实现的，当某个容器内存达到资源限制， cgroup 就会将其强制停止（类似于 `kill -9` ），此时通过 `describe pod` 可以看到 Reason 是 `OOMKilled` 。
  - 宿主机本身资源不够用（OOM），则内核会选择停止一些进程来释放内存。

    > **说明：**
    > 无论是 cgroup 限制，还是因为节点机器本身资源不够导致的进程停止，都可以从系统日志中找到记录。方法如下：
    > Ubuntu 系统日志存储在目录 `/var/log/syslog` ，CentOS 系统日志存储在目录 `/var/log/messages` 中，两者系统日志均可通过 `journalctl -k` 命令进行查看。

  - livenessProbe（存活检查）失败，使得 kubelet 停止 Pod。
  - 被恶意木马进程停止。
- **1和255**：通常表示一般错误，具体原因需要通过容器日志进一步定位。例如，可能是设置异常退出使用 `exit(1)` 或 `exit(-1)` 导致的，而-1将会根据规则转换成255。

Linux 标准中断信号

Linux 程序被外界中断时会发送中断信号，程序退出时的状态码为中断信号值加128。例如， `SIGKILL` 的中断信号值为9，那么程序退出状态码则为9 + 128 = 137。更多标准信号值参考如下表：

信号 Signal	状态码 Value	动作 Action	描述 Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating-point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers; see pipe(7)
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at terminal
SIGTTIN	21,21,26	Stop	Terminal input for background process
SIGTTOU	22,22,27	Stop	Terminal output for background process

C/C++ 退出状态码

`/usr/include/sysexits.h` 中进行了退出状态码标准化（仅限 C/C++），如下表：

定义	状态码	描述
#define EX_OK	0	successful termination
#define EX__BASE	64	base value for error messages
#define EX_USAGE	64	command line usage error
#define EX_DATAERR	65	data format error
#define EX_NOINPUT	66	cannot open input
#define EX_NOUSER	67	addressee unknown
#define EX_NOHOST	68	host name unknown
#define EX_UNAVAILABLE	69	service unavailable
#define EX_SOFTWARE	70	internal software error
#define EX_OSERR	71	system error (e.g., can't fork)
#define EX_OSFILE	72	critical OS file missing
#define EX_CANTCREAT	73	can't create (user) output file
#define EX_IOERR	74	input/output error
#define EX_TEMPFAIL	75	temp failure; user is invited to retry
#define EX_PROTOCOL	76	remote error in protocol
#define EX_NOPERM	77	permission denied
#define EX_CONFIG	78	configuration error
#define EX__MAX 78	78	maximum listed value

状态码参考

更多状态码含义可参考以下表格：

状态码	含义	示例	描述
1	Catchall for general errors	let "var1 = 1/0"	Miscellaneous errors, such as "divide by zero" and other impermissible operations
2	Misuse of shell builtins (according to Bash documentation)	empty_function() {}	Missing keyword or command, or permission problem (and diff return code on a failed binary file comparison).
126	Command invoked cannot execute	/dev/null	Permission problem or command is not an executable
127	"command not found"	illegal_command	Possible problem with $PATH or a typo
128	Invalid argument to exit	exit 3.14159	exit takes only integer args in the range 0 – 255 (see first footnote)
128+n	Fatal error signal "n"	kill -9 $PPID of script	$? returns 137 (128 + 9)
130	Script terminated by Control-C	Ctl-C	Control-C is fatal error signal 2, (130 = 128 + 2, see above)

255*	Exit status out of range	exit -1	exit takes only integer args in the range 0 - 255

Pod 一直处于 ContainerCreating 或 Waiting 状态

最近更新时间：2023-11-13 16:10:02

本文档介绍可能导致 Pod 一直处于 ContainerCreating 或 Waiting 状态的几种情形，以及如何通过排查步骤定位异常原因。请按照以下步骤依次进行排查，定位问题后恢复正确配置即可。

可能原因

- Pod 配置错误
- 挂载 Volume 失败
- 磁盘空间不足
- 节点内存碎片化
- Limit 设置过小或单位错误
- 拉取镜像失败
- CNI 网络错误
- controller-manager 异常
- 安装 docker 时未完全删除旧版本
- 存在同名容器

排查方法

检查 Pod 配置

1. 检查是否打包了正确的镜像。
2. 检查是否配置了正确的容器参数。

检查 Volume 挂载情况

通过检查 Volume 挂载情况，定位引发异常的真正原因。以下列出两种已知原因：

1. Pod 漂移导致未正常解挂磁盘

问题分析
在云托管的 K8S 服务环境下，默认挂载的 Volume 通常为一块存储类型的云硬盘。如果某个节点出现故障，导致 kubelet 无法正常运行或无法与 apiserver 通信，则到达时间阈值后就会触发该节点驱逐，自动在其他节点上启动相同的 Pod（Pod 漂移），被驱逐的节点无法正常运行及确定自身状态，故该节点的 Volume 也未正确执行解挂操作。
由于 cloud-controller-manager 需等待 Volume 正确解挂后，才会调用云厂商接口将磁盘从节点上解挂。因此，Pod 漂移会导致 cloud-controller-manager 在达到一个时间阈值后，强制解挂 Volume 并挂载到 Pod 最新调度的节点上。
造成影响
最终导致 ContainerCreating 的时间相对较长，但通常是可以启动成功的。

2. 命中 K8S 挂载 configmap/secret 时 subpath 的 bug

实践发现，当修改了 Pod 已挂载的 configmap 或 secret 的内容，Pod 中容器又进行了原地重启操作（例如，存活检查失败被 kill 后重启拉起），就会触发该 bug。
如果出现了以上情况，容器将会持续启动不成功。报错示例如下：

```
$ kubectl -n prod get pod -o yaml manage-5bd487cf9d-bqmvm
...
lastState: terminated
containerID: containerd://e6746201faa1dfe7f3251b8c30d59ebf613d99715f3b800740e587e681d2a903
exitCode: 128
finishedAt: 2019-09-15T00:47:22Z
message: 'failed to create containerd task: OCI runtime create failed: container_linux.go:345:
starting container process caused "process_linux.go:424: container init
caused \"rootfs_linux.go:58: mounting \\\"/var/lib/kubelet/pods/211d53f4-d08c-11e9-b0a7-
b6655eaf02a6/volume-subpaths/manage-config-volume/manage/0\\\"
to rootfs
\\\"/run/containerd/io.containerd.runtime.v1.linux/k8s.io/e6746201faa1dfe7f3251b8c30d59ebf613d99715f3b8007
40e587e681d2a903/rootfs\\\"
```

```
at
\\\"/run/containerd/io.containerd.runtime.v1.linux/k8s.io/e6746201faa1dfe7f3251b8c30d59ebf613d99715f3b8007
40e587e681d2a903/rootfs/app/resources/application.properties\\\"
caused \\\"no such file or directory\\\"\"": unknown'
```

解决方法及更多信息请参见 pr82784 。

检查磁盘空间是否不足

启动 Pod 时会调 CRI 接口创建容器，容器运行时组件创建容器时通常会在数据目录下为新建的容器创建一些目录和文件。如果数据目录所在的磁盘空间不足，将会导致容器创建失败。通常会显示以下报错信息：

```
Events:
Type Reason Age From Message
---- ------ ---- ---- -------
Warning FailedCreatePodSandBox 2m (x4307 over 16h) kubelet, 10.179.80.31 (combined from similar events):
Failed create pod sandbox: rpc error: code = Unknown desc = failed to create a sandbox for pod
"apigateway-6dc48bf8b6-l8xrw": Error response from daemon: mkdir
/var/lib/docker/aufs/mnt/1f09d6c1c9f24e8daaea5bf33a4230de7dbc758e3b22785e8ee21e3e3d921214-init: no space
left on device
```

解决步骤及更多信息请参见 磁盘爆满 。

检查节点内存是否碎片化

如果节点出现内存碎片化严重、缺少大页内存问题，即使总体剩余内存较多，但仍会出现申请内存失败的情况。请参考 内存碎片化 进行异常定位及解决。

检查 limit 设置

现象描述

- 执行 `kubectl describe pod` 命令，查看 event 报错信息如下：

  ```
  Pod sandbox changed, it will be killed and re-created。
  ```

- Kubelet 报错如下：

  ```
  to start sandbox container for pod ... Error response from daemon: OCI runtime create failed:
  container_linux.go:348: starting container process caused "process_linux.go:301: running exec setns
  process for init caused \"signal: killed\"": unknown
  ```

解决思路

当 limit 设置过小以至于不足以成功运行 Sandbox 时，也会导致 Pod 一直处于 ContainerCreating 或 Waiting 状态，通常是由于 memory limit 单位设置错误引起的。
例如，误将 memory limit 单位设置为小写字母 `m`，则该单位将会被 K8S 识别成 Byte。**您需要将 memory limit 单位设置为 `Mi` 或 `M`** 。若 memory limit 设置为1024m，则表示其大小将会被限制在1.024Byte以下。较小的内存环境下，pause 容器一启动就会被 cgroup-oom kill 掉，导致 Pod 状态一直处于 ContainerCreating。

检查拉取镜像是否失败

拉取镜像失败也会引起 Pod 发生该问题，镜像拉取失败原因较多，常见原因如下：

- 配置了错误的镜像。
- Kubelet 无法访问镜像仓库。例如，默认 pause 镜像在 gcr.io 上，而国内环境访问需要特殊处理。
- 拉取私有镜像的 imagePullSecret 没有配置或配置有误。
- 镜像太大，拉取超时。可在拉取时适当调整 kubelet 的 `--image-pull-progress-deadline` 和 `--runtime-request-timeout` 选项。

针对上述情况调整配置后，请再次拉取镜像并查看 Pod 状态。

检查 CNI 网络是否错误

检查网络插件的配置是否正确，以及运行状态是否正常。当网络插件配置错误或无法正常运行时，会出现以下提示：

- 无法配置 Pod 网络。
- 无法分配 Pod IP。

检查 controller-manager 是否异常

查看 Master 上 kube-controller-manager 状态，异常时尝试重启。

检查节点已有 docker

若在节点已有 docker 或旧版本 docker 未完全卸载的情况下，又安装了 docker，也可能引发 Pod 出现此问题。
例如，在 CentOS 上，执行以下命令重复安装了 docker：

```
yum install -y docker
```

由于重复安装的 docker 版本不一致，组件间不完全兼容，导致 dockerd 持续无法成功创建容器，使 Pod 状态一直 ContainerCreating。执行 `kubectl describe pod` 命令，查看 event 报错信息如下：

```
Type Reason Age From Message
---- ------ ---- ---- -------
Warning FailedCreatePodSandBox 18m (x3583 over 83m) kubelet, 192.168.4.5 (combined from similar events):
Failed create pod sandbox: rpc error: code = Unknown desc = failed to start sandbox container for pod
"nginx-7db9fccd9b-2j6dh": Error response from daemon: ttrpc: client shutting down: read unix @-
>@/containerd-shim/moby/de2bfeefc999af42783115acca62745e6798981dff75f4148fae8c086668f667/shim.sock: read:
connection reset by peer: unknown
Normal SandboxChanged 3m12s (x4420 over 83m) kubelet, 192.168.4.5 Pod sandbox changed, it will be killed
and re-created.
```

请选择保留 docker 版本，并完全卸载其余版本 docker。

检查是否存在同名容器

节点上存在同名容器会导致创建 sandbox 时失败，也会导致 Pod 一直处于 ContainerCreating 或 Waiting 状态。
执行 `kubectl describe pod` 命令，查看 event 报错信息如下：

```
Warning FailedCreatePodSandBox 2m kubelet, 10.205.8.91 Failed create pod sandbox: rpc error: code =
Unknown desc = failed to create a sandbox for pod "lomp-ext-d8c8b8c46-4v8tl": operation timeout: context
deadline exceeded
Warning FailedCreatePodSandBox 3s (x12 over 2m) kubelet, 10.205.8.91 Failed create pod sandbox: rpc error:
code = Unknown desc = failed to create a sandbox for pod "lomp-ext-d8c8b8c46-4v8tl": Error response from
daemon: Conflict. The container name "/k8s_POD_lomp-ext-d8c8b8c46-4v8tl_default_65046a06-f795-11e9-9bb6-
b67fb7a70bad_0" is already in use by container
"30aa3f5847e0ce89e9d411e76783ba14accba7eb7743e605a10a9a862a72c1e2". You have to remove (or rename) that
container to be able to reuse that name.
```

请修改容器名，确保节点上不存在同名容器。

Pod 一直处于 ImagePullBackOff 状态

最近更新时间：2024-03-25 15:41:12

本文档介绍可能导致 Pod 一直处于 ImagePullBackOff 状态的几种情形，以及如何通过排查步骤定位异常原因。请按照以下步骤依次进行排查，定位问题后恢复正确配置即可。

可能原因

- HTTP 类型 Registry 地址未加入 insecure-registry
- HTTPS 自签发类型 Registry CA 证书未添加至节点
- 私有镜像仓库认证失败
- 镜像文件损坏
- 镜像拉取超时
- 镜像不存在

排查方法

检查 HTTP 类型 Registry 地址是否加入 insecure-registry

Dockerd 默认从 HTTPS 类型的 Registry 拉取镜像。当您使用 HTTP 类型的 Registry 时，请确保已将其地址添加到 insecure-registry 参数中，并重启或 reload Dockerd 使其生效。

检查 HTTPS 自签发类型 Registry CA 证书是否未添加至节点

当您使用 HTTPS 类型 Registry 且其证书属于自签发证书时，Dockerd 将会校验该证书，只有校验成功才可以正常使用镜像仓库。
为确保校验成功，需要将 Registry 的 CA 证书放置到以下位置：

```
/etc/docker/certs.d/<Registry:port>/ca.crt
```

检查私有镜像仓库配置

若 Pod 未配置 imagePullSecret、配置的 Secret 不存在或者有误都会造成 Registry 认证失败，使 Pod 一直处于 ImagePullBackOff 状态。

检查镜像文件是否损坏

若 Push 的镜像文件损坏，下载成功后也不能正常使用，则需要重新 push 镜像文件。

检查镜像是否拉取超时

现象描述

当节点上同时启动大量 Pod 时，可能会导致容器镜像下载需要排队。假设下载队列靠前位置已有许多大容量镜像且需较长的下载时间，则会导致排在队列靠后的 Pod 拉取超时。
默认情况下，kubelet 支持串行下载镜像。如下所示：

```
--serialize-image-pulls   Pull images one at a time. We recommend *not* changing the default value on
nodes that run docker daemon with version < 1.9 or an Aufs storage backend. Issue #10959 has more details.
(default true)
```

解决思路

必要情况下，为避免 Pod 拉取超时，可开启并行下载及控制并发。示例如下：

```
--Registry-qps int32   If > 0, limit Registry pull QPS to this value.  If 0, unlimited. (default 5)
--Registry-burst int32   Maximum size of a bursty pulls, temporarily allows pulls to burst to this number,
while still not exceeding Registry-qps. Only used if --Registry-qps > 0 (default 10)
```

检查镜像是否存在

镜像本身不存在也会导致 Pod 一直处于 ImagePullBackOff 状态，可以通过 kubelet 日志进行确认。如下所示：

```
PullImage "imroc/test:v0.2" from image service failed: rpc error: code = Unknown desc = Error response
from daemon: manifest for imroc/test:v0.2 not found
```

Pod 一直处于 Pending 状态

最近更新时间：2024-03-25 10:35:11

本文档介绍可能导致 Pod 一直处于 Pending 状态的几种情形，以及如何通过排查步骤定位异常原因。请按照以下步骤依次进行排查，定位问题后恢复正确配置即可。

现象描述

当 Pod 一直处于 Pending 状态时，说明该 Pod 还未被调度到某个节点上，需查看 Pod 分析问题原因。例如执行 `kubectl describe pod <pod-name>` 命令，则获取到的事件信息如下：

```
$ kubectl describe pod tikv-0
...
Events:
  Type     Reason            Age                 From               Message
  ----     ------            ----                ----               -------
  Warning  FailedScheduling  3m (x106 over 33m)  default-scheduler  0/4 nodes are available: 1 node(s) had
no available volume zone, 2 Insufficient cpu, 3 Insufficient memory.
```

可能原因

- 节点资源不足
- 不满足 nodeSelector 与 affinity
- Node 存在 Pod 没有容忍的污点
- 低版本 kube-scheduler 的 bug
- kube-scheduler 未正常运行
- 驱逐后其他可用节点与当前节点的有状态应用不在相同可用区

排查方法

检查节点是否资源不足

问题分析

节点资源不足有以下几种情况：

- CPU 负载过高。
- 剩余可以被分配的内存不足。
- 剩余可用 GPU 数量不足（通常在机器学习场景、GPU 集群环境）。

为进一步判断某个 Node 资源是否足够，可执行以下命令获取资源分配信息：

```
kubectl describe node <node-name>
```

在返回信息中，请注意关注以下内容：

- `Allocatable`：表示此节点能够申请的资源总和。
- `Allocated resources`：表示此节点已分配的资源（Allocatable 减去节点上所有 Pod 总的 Request）。

造成影响

前者与后者相减，即可得出剩余可申请的资源大小。如果该值小于 Pod 的 Request，则不满足 Pod 的资源要求，Scheduler 在 Predicates（预选）阶段就会剔除掉该 Node，不会调度 Pod 到该 Node。

检查 nodeSelector 及 affinity 的配置

假设 Pod 中 nodeSelector 指定了节点 Label，则调度器将只考虑调度 Pod 到包含该 Label 的 Node 上。当不存在符合该条件的 Node 时，Pod 将无法被调度。更多相关信息可前往 Kubernetes 官方网站 进行查看。

此外，如果 Pod 配置了 affinity（亲和性），则调度器根据调度算法可能无法发现符合条件的 Node，从而无法调度。affinity 包括以下几类：

- `nodeAffinity`：节点亲和性，可以看作增强版的 nodeSelector，用于限制 Pod 只允许被调度到某一部分符合条件的 Node。

- `podAffinity`：Pod 亲和性，用于将一系列有关联的 Pod 调度到同一个地方，即同一个节点或同一个可用区的节点等。
- `podAntiAffinity`：Pod 反亲和性，防止某一类 Pod 调度到同一个地方，可以有效避免单点故障。例如，将集群 DNS 服务的 Pod 副本分别调度到不同节点，可避免因一个节点出现故障而造成整个集群 DNS 解析失败，甚至使业务中断。

检查 Node 是否存在 Pod 没有容忍的污点

问题分析

假如节点上存在污点（Taints），而 Pod 上没有相应的容忍（Tolerations），Pod 将不会调度到该 Node。在调度之前，可以先通过 `kubectl describe node <node-name>` 命令查看 Node 已设置污点。示例如下：

```
$ kubectl describe nodes host1
...
Taints:             special=true:NoSchedule
...
```

Node 上已设置的污点可通过手动或自动的方式添加，详情请参见 添加污点 。

解决方法

本文提供以下两种方法，通常选择方法2解决该问题：

- 方法1：删除污点
  执行以下命令，删除污点 `special` 。

  ```
  kubectl taint nodes host1 special-
  ```

- 方法2：在 Pod 上增加污点容忍

  > **说明**
  > 本文以向 Deployment 中已创建的 Pod（名称为 `nginx` ）添加容忍为例。

  1.1 参考 使用标准登录方式登录 Linux 实例（推荐），登录 `nginx` 所在的云服务器。

  1.2 执行以下命令，编辑 Yaml。

  ```
  kubectl edit deployment nginx
  ```

  1.3 在 Yaml 文件 `template` 中的 `spec` 处添加容忍。例如，增加已存在 `special` 污点所对应的容忍：

  ```
      tolerations:
      - key: "special"
        operator: "Equal"
        value: "true"
        effect: "NoSchedule"
  ```

1.4 添加完成后如下图所示：

```
template:
  metadata:
    creationTimestamp: null
    labels:
      k8s-app: nginx
      qcloud-app: nginx
  spec:
    containers:
    - image: nginx:latest
      imagePullPolicy: Always
      name: test
      resources:
        limits:
          cpu: 500m
          memory: 1Gi
        requests:
          cpu: 250m
          memory: 256Mi
      securityContext:
        privileged: false
      terminationMessagePath: /dev/termination-log
      terminationMessagePolicy: File
    dnsPolicy: ClusterFirst
    imagePullSecrets:
    - name: qcloudregistrykey
    - name: tencenthubkey
    restartPolicy: Always
    schedulerName: default-scheduler
    securityContext: {}
    terminationGracePeriodSeconds: 30
    tolerations:
    - effect: NoSchedule
      key: special
      operator: Equal
      value: "true"
```

1.5 保存并退出编辑即可成功创建容忍。

检查是否存在低版本 kube-scheduler 的 bug

Pod 一直处于 Pending 状态可能是低版本 `kube-scheduler` 的 bug 导致的，该情况可以通过升级调度器版本进行解决。

检查 kube-scheduler 是否正常运行

请注意检查 Master 上的 `kube-scheduler` 是否运行正常，如异常可尝试重启临时恢复。

检查驱逐后其他可用节点与当前节点的有状态应用是否不在相同可用区

服务部署成功且正在运行时，若此时节点突发故障，就会触发 Pod 驱逐，并创建新的 Pod 副本调度到其他节点上。对于已挂载了磁盘的 Pod，通常需要被调度到与当前故障节点和挂载磁盘所处同一个可用区的新的节点上。若集群中同一个可用区内不具备满足可调度条件的节点时，即使其他可用区内具有满足条件的节点，此类 Pod 仍不会调度。
限制已挂载磁盘的 Pod 不能调度到其他可用区的节点的原因如下：
云上磁盘允许被动态挂载到同一个数据中心上的不同机器，为了有效避免网络时延极大地降低 IO 速率，通常不允许跨数据中心挂载磁盘设备。

相关操作

添加污点

手动添加污点

通过以下或类似方式，可以手动为节点添加指定污点：

```
$ kubectl taint node host1 special=true:NoSchedule
node "host1" tainted
```

> **说明**

> 在某些场景下，可能期望新加入的节点在调整好某些配置之前默认不允许调度 Pod。此时，可以给该新节点添加 `node.kubernetes.io/unschedulable` 污点。

自动添加污点

从 v1.12 开始，Beta 默认开启 `TaintNodesByCondition` 特性，controller manager 将会检查 Node 的 Condition。Node 运行状态异常时，当检查的 Condition 符合如下条件（即符合 Condition 与 Taints 的对应关系），将自动给 Node 加上相应的污点。
例如，检查 Condition 为 `OutOfDisk` 且 Value 为 `True`，则 Node 会自动添加 `node.kubernetes.io/out-of-disk` 污点。
Condition 与污点的对应关系如下：

```
Conditon              Value       Taints
 --------              -----       ------
OutOfDisk             True        node.kubernetes.io/out-of-disk
Ready                 False       node.kubernetes.io/not-ready
Ready                 Unknown     node.kubernetes.io/unreachable
MemoryPressure        True        node.kubernetes.io/memory-pressure
PIDPressure           True        node.kubernetes.io/pid-pressure
DiskPressure          True        node.kubernetes.io/disk-pressure
NetworkUnavailable    True        node.kubernetes.io/network-unavailable
```

当每种 Condition 取特定的值时，将表示以下含义：

- `OutOfDisk` 为 True，表示节点磁盘空间不足。
- `Ready` 为 False，表示节点不健康。
- `Ready` 为 Unknown，表示节点失联。在 `node-monitor-grace-period` 所确定的时间周期内（默认40s）若节点没有上报状态，controller-manager 就会将 Node 状态置为 Unknown。
- `MemoryPressure` 为 True，表示节点内存压力大，实际可用内存很少。
- `PIDPressure` 为 True，表示节点上运行了太多进程，PID 数量不足。
- `DiskPressure` 为 True，表示节点上的磁盘可用空间不足。
- `NetworkUnavailable` 为 True，表示节点上的网络没有正确配置，无法跟其他 Pod 正常通信。

> **说明**
> 上述情况一般属于被动添加污点，但在容器服务中，存在一个主动添加/移出污点的过程：
> 在新增节点时，首先为该节点添加 `node.cloudprovider.kubernetes.io/uninitialized` 污点，待节点初始化成功后再自动移除此污点，以避免 Pod 被调度到未初始化好的节点。

Pod 一直处于 Terminating 状态

最近更新时间：2024-03-25 15:41:13

本文档将为您展示可能导致 Pod 一直处于 Terminating 状态的几种情形，以及如何通过排查步骤定位异常原因。请按照以下步骤依次进行排查，定位问题后恢复正确配置即可。

可能原因

- 磁盘空间不足
- 存在 “i” 文件属性
- Docker 17 版本 bug
- 存在 Finalizers
- 低版本 kubelet list-watch 的 bug
- Dockerd 与 containerd 状态不同步
- Daemonset Controller Bug

排查方法

检查磁盘空间是否不足

当 Docker 的数据目录所在磁盘被写满时，Docker 将无法正常运行，甚至无法进行删除和创建操作。kubelet 调用 Docker 删除容器时将无响应，执行 `kubectl describe pod <pod-name>` 命令，查看 event 通常返回信息如下：

```
Normal  Killing  39s (x735 over 15h)  kubelet, 10.179.80.31  Killing container with id
docker://apigateway:Need to kill Pod
```

解决方法及更多信息请参考 磁盘爆满 。

检查是否存在 “i” 文件属性

现象描述

“i” 文件属性描述可通过 `man chattr` 进行查看，描述示例如下：

```
       A file with the 'i' attribute cannot be modified: it cannot be deleted or renamed, no link can be
created to this file and no data can be written to the file.  Only the superuser or a process possessing
the CAP_LINUX_IMMUTABLE capability can set or clear this attribute.
```

> **注意**
> 如果容器镜像本身或者容器启动后写入的文件存在 “i” 文件属性，此文件将无法被修改或删除。

在进行 Pod 删除操作时，会清理容器目录，若该目录中存在不可删除的文件，会导致容器目录无法删除，Pod 状态也将一直保持 Terminating。此种情况下，kubelet 将会出现以下报错：

```
Sep 27 14:37:21 VM_0_7_centos kubelet[14109]: E0927 14:37:21.922965   14109 remote_runtime.go:250]
RemoveContainer "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257" from runtime service
failed: rpc error: code = Unknown desc = failed to remove container
"19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257": Error response from daemon: container
19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257: driver "overlay2" failed to remove root
filesystem: remove
/data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0391db88534027546868/diff/usr/bin/bash:
operation not permitted
Sep 27 14:37:21 VM_0_7_centos kubelet[14109]: E0927 14:37:21.923027   14109 kuberuntime_gc.go:126] Failed
to remove container "19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257": rpc error: code =
Unknown desc = failed to remove container
"19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257": Error response from daemon: container
19d837c77a3c294052a99ff9347c520bc8acb7b8b9a9dc9fab281fc09df38257: driver "overlay2" failed to remove root
```

```
filesystem: remove
/data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0391db88534027546868/diff/usr/bin/bash:
operation not permitted
```

解决方法

- 彻底解决方法：不在容器镜像中或启动后的容器设置 “i” 文件属性，彻底杜绝此问题发生。
- 临时恢复方法：

  1.1 针对 kubelet 日志报错提示的文件路径，执行 `chattr -i <file>` 命令。示例如下：

  ```
  chattr -i
  /data/docker/overlay2/b1aea29c590aa9abda79f7cf3976422073fb3652757f0391db88534027546868/diff/usr/bin/
  bash
  ```

  1.2 等待 kubelet 自动重试，Pod 即可自动删除。

检查是否存在 Docker 17 版本 bug

现象描述

Docker hang 住，没有任何响应。执行 `kubectl describe pod <pod-name>` 命令查看 event 显示如下：

```
Warning FailedSync 3m (x408 over 1h) kubelet, 10.179.80.31 error determining status: rpc error: code =
DeadlineExceeded desc = context deadline exceeded
```

造成该问题原因可能为17版本 dockerd 的 bug，虽然可以通过 `kubectl -n cn-staging delete pod apigateway-6dc48bf8b6-clcwk --force --grace-period=0` 强制删除 Pod，但执行 `docker ps` 命令后，仍然看得到该容器。

解决方法

升级 Docker 版本至18，该版本使用了新的 containerd，针对很多已有 bug 进行了修复。
若 Pod 仍出现 Terminating 状态，请 在线咨询 联系工程师进行排查。**不建议直接强行删除**，可能会导致业务出现问题。

检查是否存在 Finalizers

现象描述

K8S 资源的 metadata 中如果存在 `finalizers`，通常说明该资源是由某个程序创建的，`finalizers` 中也会添加一个专属于该程序的标识。例如，Rancher 创建的一些资源就会写入 `finalizers` 标识。
若想要删除该程序所创建的资源时，则需要由创建该资源的程序进行删除前的清理，且只有清理完成并将标识从该资源的 `finalizers` 中移除，才可以彻底删除资源。

解决方法

使用 `kubectl edit` 命令手动编辑资源定义，删除 `finalizers`，删除资源便不会再受阻。

检查是否存在低版本 kubelet list-watch 的 bug

历史排查异常过程中发现，使用 v1.8.13 版本的 K8S 时，kubelet 会出现 list-watch 异常的情况。该问题会导致在删除 Pod 后，kubelet 未获取相关事件，并未真正删除，使 Pod 一直处 Terminating 状态。
请参考文档 升级集群 步骤进行集群 Kubernetes 版本升级。

Dockerd 与 containerd 状态不同步

现象描述

docker 在 aufs 存储驱动下如果磁盘爆满，则可能发生内核 panic，报错信息如下：

```
aufs au_opts_verify:1597:dockerd[5347]: dirperm1 breaks the protection by the permission bits on the lower
branch
```

若磁盘曾爆满过，dockerd 日志通常会有以下类似记录，且后续可能发生状态不同步问题。

```
Sep 18 10:19:49 VM-1-33-ubuntu dockerd[4822]: time="2019-09-18T10:19:49.903943652+08:00" level=error
msg="Failed to log msg \"\" for logger json-file: write
/opt/docker/containers/54922ec8b1863bcc504f6dac41e40139047f7a84ff09175d2800100aaccbad1f/54922ec8b1863bcc50
4f6dac41e40139047f7a84ff09175d2800100aaccbad1f-json.log: no space left on device"
```

问题分析

判断 dockerd 与 containerd 的某个容器状态是否同步，可采用以下几种方法：

- 首先通过 `describe pod` 获取容器 ID，再通过 `docker ps` 查看容器状态是否为 dockerd 中所保存的状态。
- 通过 `docker-container-ctr` 查看容器在 containerd 中的状态。示例如下：

```
$ docker-container-ctr --namespace moby --address /var/run/docker/containerd/docker-containerd.sock
task ls |grep a9a1785b81343c3ad2093ad973f4f8e52dbf54823b8bb089886c8356d4036fe0
a9a1785b81343c3ad2093ad973f4f8e52dbf54823b8bb089886c8356d4036fe0    30639    STOPPED
```

若 containerd 中容器状态是 stopped 或者已经无记录，而 docker 中容器状态却是 running，则说明 dockerd 与 containerd 之间容器状态同步存在问题。

解决方法

- 临时解决方法：执行 `docker container prune` 命令或重启 dockerd。
- 彻底解决方法：运行时推荐直接使用 containerd，绕过 dockerd 避免 Docker 本身的 Bug。

Daemonset Controller Bug

K8S 中存在的 Bug 会导致 Daemonset Pod 持续 Terminating，Kubernetes 1.10 和 1.11 版本受此影响。由于 Daemonset Controller 复用 scheduler 的 predicates 逻辑，将 nodeAffinity 的 nodeSelector 数组进行排序（传递的指针参数），导致 spec 与 apiserver 中的值不一致。为了版本控制，Daemonset Controller 又使用了 spec 为 rollingUpdate 类型的 Daemonset 计算 hash。
上述传递过程造成的前后参数不一致问题，导致了 Pod 陷入持续启动和停止的循环。

解决方法

- 临时解决方法：确保 rollingUpdate 类型 Daemonset 使用 nodeSelector 而不使用 nodeAffinity。
- 彻底解决方法：参考文档 升级集群 步骤将集群 Kubernetes 版本升级至 1.12。

Pod 健康检查失败

最近更新时间：2024-03-25 15:41:13

本文档介绍可能导致 Pod 健康检查失败的几种情形，以及如何通过排查步骤定位异常原因。请按照以下步骤依次进行排查，定位问题后恢复正确配置即可。

现象描述

Kubernetes 健康检查包含就绪检查（readinessProbe）和存活检查（livenessProbe），不同阶段的检查失败将会分别出现以下现象：

- Pod IP 从 Service 中摘除。通过 Service 访问时，流量将不会被转发给就绪检查失败的 Pod。
- kubelet 将会停止容器并尝试重启。

引发健康检查失败的诱因种类较多。例如，业务程序存在某个 Bug 导致其不能响应健康检查，使 Pod 处于 Unhealthy 状态。接下来您可按照以下方式进行排查。

可能原因

- 健康检查配置不合理
- 节点负载过高
- 容器进程被木马进程停止
- 容器内进程端口监听故障
- SYN backlog 设置过小

排查方法

检查健康检查配置

如果健康检查配置不合理，会导致 Pod 健康检查失败。例如，容器启动完成后首次探测的时间 `initialDelaySeconds` 设置过短。容器启动较慢时，导致容器还没完全启动就开始探测。同时，若 `successThreshold` 默认值设置为1，则 Pod 健康检查失败一次就会被停止，那么 Pod 将会持续被停止并重启。

检查节点是否负载过高

CPU 占用高（例如跑满）将导致进程无法正常发包收包，通常会出现 timeout，导致 Pod 健康检查失败。请参考 高负载 进行异常问题定位及解决。

检查容器进程是否被木马进程停止

请参考 使用 Systemtap 定位 Pod 异常退出原因 异常问题定位及解决。

检查容器内进程端口是否监听故障

使用 `netstat -tunlp` 检查端口监听是否还存在。分析该命令的返回结果可知：当端口监听不存在时，健康检查探测的连接将会被直接 reset 掉。示例如下：

```
20:15:17.890996 IP 172.16.2.1.38074 > 172.16.2.23.8888: Flags [S], seq 96880261, win 14600, options [mss
1424,nop,nop,sackOK,nop,wscale 7], length 0
20:15:17.891021 IP 172.16.2.23.8888 > 172.16.2.1.38074: Flags [R.], seq 0, ack 96880262, win 0, length 0
20:15:17.906744 IP 10.0.0.16.54132 > 172.16.2.23.8888: Flags [S], seq 1207014342, win 14600, options [mss
1424,nop,nop,sackOK,nop,wscale 7], length 0
20:15:17.906766 IP 172.16.2.23.8888 > 10.0.0.16.54132: Flags [R.], seq 0, ack 1207014343, win 0, length 0
```

分析以上结果可知，健康检查探测连接异常，将会导致健康检查失败。可能原因为：
节点上同时启动多个使用 `hostNetwork` 监听相同宿主机端口的 Pod，但只有一个Pod 会监听成功，其余 Pod 监听失败且不会退出，继续进行适配健康检查。从而使得 kubelet 发送健康检查探测报文给 Pod 时，发送给监听失败（即没有监听端口）的 Pod，导致健康检查失败。

检查 SYN backlog 设置是否过小

现象描述

SYN backlog 大小即 SYN 队列大小，如果短时间内新建连接比较多，而 SYN backlog 设置过小，就会导致新建连接失败。使用 `netstat -s | grep TCPBacklogDrop` 可查看由于 backlog 满了导致丢弃的新连接数量。

解决方法

如果已确认由于 backlog 满了导致的丢包，则建议调高 backlog 值，相关内核参数为 `net.ipv4.tcp_max_syn_backlog`。

Pod 处于 CrashLoopBackOff 状态

最近更新时间：2024-03-25 15:41:13

本文档介绍可能导致 Pod 处于 CrashLoopBackOff 状态的几种情形，以及如何通过排查步骤定位异常原因。请按照以下步骤依次进行排查，定位问题后恢复正确配置即可。

现象描述

Pod 处于 `CrashLoopBackOff` 状态，说明该 Pod 在正常启动过后异常退出过，此状态下 Pod 的 restartPolicy 如果不是 Never 就可能会被重启拉起，且 Pod 的 `RestartCounts` 通常大于0。可首先参考 通过 Exit Code 定位 Pod 异常退出原因 查看对应容器进程的退出状态码，缩小异常问题范围。

可能原因

- 容器进程主动退出
- 系统 OOM
- cgroup OOM
- 节点内存碎片化
- 健康检查失败

排查步骤

检查容器进程是否主动退出

容器进程主动退出时，退出状态码通常在0 – 128之间，导致异常的原因可能是业务程序 Bug，也可能是其他原因。请参考 容器进程主动退出 进一步定位异常问题。

检查是否发生系统 OOM

问题分析

如果发生系统 OOM，Pod 中容器退出状态码为137，表示其被 `SIGKILL` 信号停止，同时内核将会出现以下报错信息。

```
Out of memory: Kill process ...
```

该异常是由于节点上部署了其他非 K8S 管理的进程消耗了较多的内存，或是 kubelet 的 `--kube-reserved` 和 `--system-reserved` 所分配的内存太小，没有足够的空间运行其他非容器进程。

> **说明**
> 节点上所有 Pod 的实际内存占用总量不会超过 `/sys/fs/cgroup/memory/kubepods` 中的 cgroup 值（`cgroup = capacity - "kube-reserved" - "system-reserved"`）。通常情况下，如果预留空间设置合理，且节点上其他非容器进程（例如 kubelet、dockerd、kube-proxy 及 sshd 等）内存占用没有超过 kubelet 配置的预留空间，是不会发生系统 OOM 的。

解决方法

为确保不再发生此类问题，您可以根据实际需求对预留空间进行合理的调整。

检查是否发生 cgroup OOM

现象描述

如果是因 cgroup OOM 而停止的进程，可看到 Pod 事件下 `Reason` 为 `OOMKilled`，说明容器实际占用的内存已超过 limit，同时内核日志会报 `Memory cgroup out of memory` 错误信息。

解决方法

请根据需求调整 limit。

节点内存碎片化

如果节点出现内存碎片化严重、缺少大页内存问题，即使总体剩余内存较多，但仍会出现申请内存失败的情况。请参考 内存碎片化 进行异常定位及解决。

健康检查失败

请参考 Pod 健康检查失败 进一步定位异常问题。

Pod 无限重启且流量异常

最近更新时间：2024-03-25 18:18:32

故障现象

Pod 突然不断重启，期间有流量进入，这部分流量异常。

原因

1. Pod 之前所在节点异常，重建漂移到了其它节点去启动。
2. Pod 重建后由于基础镜像中依赖的一个服务有问题导致启动较慢，因为同时配置了 ReadinessProbe 与 LivenessProbe，大概率是启动时所有健康检查都失败，达到 LivenessProbe 失败次数阈值，又被重启。
3. Pod 配置了 preStop 实现优雅终止，被重启前会先执行 preStop，优雅终止的时长较长，preStop 期间 ReadinessProbe 还会继续探测。
4. 探测方式使用的 TCP 探测，进程优雅终止过程中 TCP 探测仍然会成功（没完全退出前端口监听仍然存在），但实际此时进程已不会处理新请求了。
5. LivenessProbe 结果不会影响 Pod Ready 状态，是否 Ready 主要取决于 ReadinessProbe 结果，由于 preStop 期间 ReadinessProbe 是成功的，Pod 就变 Ready 了。
6. Pod Ready 但实际无法处理请求，业务就会异常。

总结

1. Pod 慢启动 + 存活探测 导致被无限重启。需要延长 `initialDelaySeconds` 或 StartProbe 来保护慢启动容器。
2. TCP 探测方式不能完全真实反映业务健康状态，导致在优雅终止过程中，ReadinessProbe 探测成功让流量放进来而业务却不会处理，导致流量异常。需要使用更好的探测方式，建议业务提供 HTTP 探活接口，使用 HTTP 探测业务真实健康状态。

容器进程主动退出

最近更新时间：2025-06-17 14:50:51

本文档介绍可能导致容器进程主动退出的几种场景，以及如何通过排查步骤定位异常原因。请按照以下步骤依次进行排查，定位问题后恢复正确配置即可。

现象描述

容器进程主动退出（不是被外界中断停止）时，退出状态码通常在0-128之间。根据规定，正常退出时状态码为0，状态码为1-127则说明为程序发生异常导致其主动退出。例如，当检测到程序启动参数和条件不满足要求，或者程序运行过程中发生 panic 但没有捕获处理就会导致程序主动退出。可首先参考 通过 Exit Code 定位 Pod 异常退出原因 查看对应容器进程的退出状态码，缩小异常问题范围。

可能原因

- 程序本身运行异常
- DNS 无法解析
- 程序配置有误

排查方法

检查容器退出前日志

可通过以下命令查看容器退出前日志，进而排查进程退出原因。

```
# 格式
kubectl logs -p -n <namespace> <pod name>
# 示例
kubectl logs -p -n mynamespace mypod
```

检查 DNS 是否无法解析

若程序依赖集群 DNS 服务，则解析失败将导致程序报错并主动退出。例如，程序启动时需要连接数据库，且数据库使用 Service 名称或外部域名都需要 DNS 解析，若解析失败将引发程序报错并主动退出，导致容器进程主动退出。解析失败的可能原因如下：

- 集群网络存在异常，Pod 无法连接集群 DNS 服务。
- 集群 DNS 服务故障，无法响应解析请求。
- Service 或域名地址配置有误，致使其无法解析。

检查程序配置

程序配置有错误也可能引起容器进程主动退出，可能原因如下：

- 配置文件格式错误，程序启动时解析配置失败从而报错退出。
- 配置内容不符合规范。例如，配置中某个必选字段未填写，导致配置校验不通过，程序报错主动退出。

为数据盘设置文件系统卷标

最近更新时间：2025-01-02 14:36:42

本文向您介绍如何为数据盘设置文件系统卷标。

使用背景

NVMe 数据盘的盘符由 NVMe 盘的驱动生成，如果驱动加载顺序不一致可能会导致盘符变化。因此，由于操作系统和物理机的限制，云服务器 CVM 部分普通机型（如黑石、高 IO 型、高性能 HCC 等）挂载多块 NVMe 数据盘时，重启系统后盘符可能会错乱。更多内容可参考 Linux 云服务器重启后云硬盘未自动挂载。

现象描述

1. 操作路径

通过 添加已有节点 将上述机型作为 Worker 节点添加至 TKE 集群中。

2. 问题描述

上述机型通过指定盘符完成挂盘操作可能不符合预期，即 NVMe 盘无法挂到指定位置。
例如，当前待添加节点所在机型存在两块 NVMe 盘，即 NVMe.0 和 NVMe.1：

- 重启前，存在映射关系 /dev/sda → NVMe.0、/dev/sdb → NVMe.1。
- 重启后，映射可能会错乱为 /dev/sda → NVMe.1、/dev/sdb → NVMe.0。

处理步骤

为解决底层物理机对盘符飘移的兼容问题，TKE 支持通过**“磁盘卷标 Label”**的方式指定数据盘挂载，操作步骤如下：

1. 添加已有节点前，可以在机器上为数据盘添加 Label，该 Label 简称为**卷标**，执行命令如下：
   - ext 文件系统：`e2label /dev/sdz label1`
   - xfs 文件系统：`xfs_admin -L label1 /dev/sdz`
2. 在 容器服务控制台 **添加已有节点**页面可直接**使用卷标 Label 来指定使用的数据盘**，如下图所示：

> **说明：**
> - 要确保数据盘对应的 Label 保持唯一性，否则会导致系统初始化失败。
> - 由于卷标 Label 和文件系统具有绑定关系，因此被指定的数据盘不会格式化。
> - 若高 IO 型、高性能 HCC 类机型需要对 NVMe 类型数据盘进行挂载，建议通过上述指定卷标 Lable 的方式单独添加进集群，不要和其他机型一起操作。

原理说明

在添加已有节点指定卷标 Label 挂盘时，安装组件会根据 Label 找到数据盘，并将数据盘挂载到挂载点上。同时将数据盘的 UUID 写入到 fstab。Label 和 UUID 都是记录在文件系统元信息上的标记，盘符的变化不会影响到 Label、UUID 和文件系统内容的对应关系。

> **说明：**
> **如果添加已有节点并指定挂载数据盘后，又来到节点上再次格式化磁盘，则需要同时更新 fstab，否则可能会导致系统重启失败。**

授权腾讯云售后运维排障

最近更新时间：2024-03-25 15:41:13

默认情况下腾讯云无法登录集群进行问题排障，如果您需要腾讯云售后协助进行运维排障，请参考以下步骤授予腾讯云运维权限。您有权随时吊销回收授予腾讯云的运维排障权限。

通过控制台授予腾讯云权限

1. 登录 容器服务控制台 。
2. 在集群管理中选择需要腾讯云协助的集群。
3. 在集群详情页，选择**授权管理 > 授权腾讯云运维**。
4. 在**集群RBAC设置**中，选择赋予腾讯云的操作权限。如下图所示：

5. 设置完成后，您可在 我的工单 中查看问题处理进度。

> **注意**
> 默认情况下腾讯云无法登录集群进行问题排障，如果您需要腾讯云售后协助进行运维排障，您可以授予腾讯云指定的运维权限，同时您有权随时吊销回收授予腾讯云的运维排障权限。
> 您可以通过删除相关资源(ClusterRoleBinding/tkeopsaccount-ClusterRole、ServiceAccount/tkeopsaccount、Sercet/tkeopsaccount-token-xxxx)吊销腾讯云运维权限。

通过 Kubernetes API 授予腾讯云权限

您可以通过创建以下 Kubernetes 资源授予腾讯云指定权限。

ServiceAccount 授予腾讯云访问集群凭证

```
kind: ServiceAccount
apiVersion: v1
metadata:
  name: tkeopsaccount
  namespace: kube-system
  labels:
    cloud.tencent.com/tke-ops-account: tkeops
```

ClusterRoleBinding/RoleBing 授予腾讯云的操作权限规则

> **说明**
> 1. 名称和 label 需按如下规则创建。
> 2. roleRef 可替换为您期望授权腾讯云的权限。

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
```

```
metadata:
  annotations:
    cloud.tencent.com/tke-ops-account: tkeops
  labels:
    cloud.tencent.com/tke-ops-account: tkeops
  name: tkeopsaccount-ClusterRole
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tke:admin
subjects:
- kind: ServiceAccount
  name: tkeopsaccount
  namespace: kube-system
```

（可选）ClusterRole/Role 授予腾讯云的操作权限

如集群内有相关 ClusterRole/Role 可直接使用 ClusterRoleBinding/RoleBinding 关联。通过控制台授权，将自动创建策略，无需单独创建。

管理员权限

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
  name: tke:admin
rules:
- apiGroups:
  - '*'
  resources:
  - '*'
  verbs:
  - '*'
- nonResourceURLs:
  - '*'
  verbs:
  - '*'
```

只读权限

```
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  labels:
    cloud.tencent.com/tke-rbac-generated: "true"
  name: tke:ro
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - pods/attach
  - pods/exec
  - pods/portforward
  - pods/proxy
  verbs:
  - get
  - list
```

```
  - watch
- apiGroups:
  - ""
  resources:
  - configmaps
  - endpoints
  - persistentvolumeclaims
  - replicationcontrollers
  - replicationcontrollers/scale
  - secrets
  - serviceaccounts
  - services
  - services/proxy
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - nodes
  - persistentvolumes
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  - replicationcontrollers/status
  - pods/log
  - pods/status
  - componentstatuses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - apps
  resources:
  - daemonsets
  - deployments
  - deployments/rollback
  - deployments/scale
  - replicasets
  - replicasets/scale
  - statefulsets
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - autoscaling
  resources:
  - horizontalpodautoscalers
  verbs:
  - get
  - list
  - watch
- apiGroups:
```

```
  - storage.k8s.io
  resources:
  - storageclasses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - batch
  resources:
  - cronjobs
  - jobs
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - extensions
  - networking.k8s.io
  resources:
  - daemonsets
  - deployments
  - deployments/rollback
  - deployments/scale
  - ingresses
  - replicasets
  - replicasets/scale
  - replicationcontrollers/scale
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - servicecatalog.k8s.io
  resources:
  - clusterserviceclasses
  - clusterserviceplans
  - clusterservicebrokers
  - serviceinstances
  - servicebindings
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - policy
  resources:
  - poddisruptionbudgets
  verbs:
  - get
  - list
- apiGroups:
  - networking.istio.io
  - config.istio.io
  - rbac.istio.io
  - authentication.istio.io
  - security.istio.io
  - install.istio.io
  resources:
  - '*'
  verbs:
```

```
  - get
  - list
  - watch
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - networking.tke.cloud.tencent.com
  resources:
  - '*'
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - cloud.tencent.com
  resources:
  - '*'
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ccs.cloud.tencent.com
  resources:
  - '*'
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - cls.cloud.tencent.com
  resources:
  - '*'
  verbs:
  - get
  - list
  - watch
```