Kubernetes 网络排错指南

本文介绍各种常见的网络问题以及排错方法，包括 Pod 访问异常、Service 访问异常以及网络安全策略异常等。

说到 Kubernetes 的网络，其实无非就是以下三种情况之一

- Pod 访问容器外部网络
- 从容器外部访问 Pod 网络
- Pod 之间相互访问

当然，以上每种情况还都分别包括本地访问和跨主机访问两种场景，并且一般情况下都是通过 Service 间接访问 Pod。

排查网络问题基本上也是从这几种情况出发，定位出具体的网络异常点，再进而寻找解决方法。网络异常可能的原因比较多，常见的有：

CNI 网络插件配置错误，导致多主机网络不通，比如

- IP 网段与现有网络冲突
- 插件使用了底层网络不支持的协议
- 忘记开启 IP 转发等
  - `sysctl net.ipv4.ip_forward`
  - `sysctl net.bridge.bridge-nf-call-iptables`

Pod 网络路由丢失，比如

- kubenet 要求网络中有 podCIDR 到主机 IP 地址的路由，这些路由如果没有正确配置会导致 Pod 网络通信等问题
- 在公有云平台上，kube-controller-manager 会自动为所有 Node 配置路由，但如果配置不当（如认证授权失败、超出配额等），也有可能导致无法配置路由

- 主机内或者云平台的安全组、防火墙或者安全策略等阻止了 Pod 网络，比如
  - 非 Kubernetes 管理的 iptables 规则禁止了 Pod 网络
  - 公有云平台的安全组禁止了 Pod 网络（注意 Pod 网络有可能与 Node 网络不在同一个网段）
  - 交换机或者路由器的 ACL 禁止了 Pod 网络

Flannel Pods 一直处于 Init:CrashLoopBackOff 状态

Flannel 网络插件非常容易部署，只要一条命令即可

```
kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

然而，部署完成后，Flannel Pod 有可能会碰到初始化失败的错误

```
$ kubectl -n kube-system get pod
NAME                          READY     STATUS                  RESTARTS   AGE
kube-flannel-ds-ckfdc         0/1       Init:CrashLoopBackOff   4          2m
kube-flannel-ds-jpp96         0/1       Init:CrashLoopBackOff   4          2m
```

查看日志会发现

```
$ kubectl -n kube-system  logs kube-flannel-ds-jpp96 -c  install-cni
cp: can't create '/etc/cni/net.d/10-flannel.conflist': Permission denied
```

这一般是由于 SELinux 开启导致的，关闭 SELinux 既可解决。有两种方法：

- 修改 `/etc/selinux/config` 文件方法：`SELINUX=disabled`
- 通过命令临时修改（重启会丢失）：`setenforce 0`

Pod 无法解析 DNS

如果 Node 上安装的 Docker 版本大于 1.12，那么 Docker 会把默认的 iptables FORWARD 策略改为 DROP。这会引发 Pod 网络访问的问题。解决方法则在每个 Node 上面运行 `iptables -P FORWARD ACCEPT`，比如

```
echo "ExecStartPost=/sbin/iptables -P FORWARD ACCEPT" >> /etc/systemd/system/docker.service.d/exec_start.conf
systemctl daemon-reload
systemctl restart docker
```

如果使用了 flannel/weave 网络插件，更新为最新版本也可以解决这个问题。

DNS 无法解析也有可能是 kube-dns 服务异常导致的，可以通过下面的命令来检查 kube-dns 是否处于正常运行状态

```
$ kubectl  get  pods --namespace=kube-system -l k8s-app=kube-dns
NAME                        READY       STATUS       RESTARTS     AGE
...
kube-dns-v19-ezoly    3/3           Running      0                  1h
...
```

如果 kube-dns 处于 CrashLoopBackOff 状态，那么可以参考 Kube-dns/Dashboard CrashLoopBackOff 排错 来查看具体排错方法。

如果 kube-dns Pod 处于正常 Running 状态，则需要进一步检查是否正确配置了 kube-dns 服务：

```
$ kubectl  get  svc kube-dns --namespace=kube-system
NAME          CLUSTER-IP     EXTERNAL-IP     PORT(S)             AGE
kube-dns    10.0.0.10        <none>          53/UDP,53/TCP         1h

$ kubectl  get  ep kube-dns --namespace=kube-system
NAME          ENDPOINTS                                     AGE
kube-dns      10.180.3.17:53,10.180.3.17:53         1h
```

如果 kube-dns service 不存在，或者 endpoints 列表为空，则说明 kube-dns service 配置错误，可以重新创建 kube-dns service，比如

```
apiVersion: v1
kind: Service
metadata:
 name: kube-dns
 namespace: kube-system
 labels:
   k8s-app: kube-dns
   kubernetes.io/cluster-service: "true"
```

```
    kubernetes.io/name: "KubeDNS"
spec:
  selector:
    k8s-app: kube-dns
  clusterIP: 10.0.0.10
  ports:
  - name: dns
    port: 53
    protocol: UDP
  - name: dns-tcp
    port: 53
    protocol: TCP
```

Service 无法访问

访问 Service ClusterIP 失败时，可以首先确认是否有对应的 Endpoints

```
kubectl get endpoints <service-name>
```

如果该列表为空，则有可能是该 Service 的 LabelSelector 配置错误，可以用下面的方法确认一下

```
# 查询 Service 的 LabelSelector
kubectl get svc <service-name> -o jsonpath='{.spec.selector}'
# 查询匹配 LabelSelector 的 Pod
kubectl get pods -l key1=value1,key2=value2
```

如果 Endpoints 正常，可以进一步检查

- Pod 的 containerPort 与 Service 的 containerPort 是否对应
- 直接访问 `podIP:containerPort` 是否正常

再进一步，即使上述配置都正确无误，还有其他的原因会导致 Service 无法访问，比如

- Pod 内的容器有可能未正常运行或者没有监听在指定的 containerPort 上
- CNI 网络或主机路由异常也会导致类似的问题
- kube-proxy 服务有可能未启动或者未正确配置相应的 iptables 规则，比如正常情况下名为 `hostnames`的服务会配置以下 iptables 规则

```
$ iptables-save | grep hostnames
-A KUBE-SEP-57KPRZ3JQVENLNBR -s 10.244.3.6/32 -m comment --comment "default/hostnames:" -j MARK --set-xmark 0x00004000/0x00004000
-A KUBE-SEP-57KPRZ3JQVENLNBR -p tcp -m comment --comment "default/hostnames:" -m tcp -j DNAT --to-destination 10.244.3.6:9376
-A KUBE-SEP-WNBA2IHDGP2BOBGZ -s 10.244.1.7/32 -m comment --comment "default/hostnames:" -j MARK --set-xmark 0x00004000/0x00004000
-A KUBE-SEP-WNBA2IHDGP2BOBGZ -p tcp -m comment --comment "default/hostnames:" -m tcp -j DNAT --to-destination 10.244.1.7:9376
-A KUBE-SEP-X3P2623AGDH6CDF3 -s 10.244.2.3/32 -m comment --comment "default/hostnames:" -j MARK --set-xmark 0x00004000/0x00004000
-A KUBE-SEP-X3P2623AGDH6CDF3 -p tcp -m comment --comment "default/hostnames:" -m tcp -j DNAT --to-destination 10.244.2.3:9376
```

```
-A KUBE-SERVICES -d 10.0.1.175/32 -p tcp -m comment --comment "default/hostnames: cluster IP" -m tcp --dport 80 -j KUBE-SVC-NWV5X2332I4OT4T3
-A KUBE-SVC-NWV5X2332I4OT4T3 -m comment --comment "default/hostnames:" -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-WNBA2IHDGP2BOBGZ
-A KUBE-SVC-NWV5X2332I4OT4T3 -m comment --comment "default/hostnames:" -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-X3P2623AGDH6CDF3
-A KUBE-SVC-NWV5X2332I4OT4T3 -m comment --comment "default/hostnames:" -j KUBE-SEP-57KPRZ3JQVENLNBR
```

Pod 无法通过 Service 访问自己

这通常是 hairpin 配置错误导致的，可以通过 Kubelet 的 `--hairpin-mode` 选项配置，可选参数包括 "promiscuous-bridge"、"hairpin-veth" 和 "none"（默认为"promiscuous-bridge"）。

对于 hairpin-veth 模式，可以通过以下命令来确认是否生效

```
$ for intf in /sys/devices/virtual/net/cbr0/brif/*; do cat $intf/hairpin_mode; done
1
1
1
1
```

而对于 promiscuous-bridge 模式，可以通过以下命令来确认是否生效

```
$ ifconfig cbr0 |grep PROMISC
UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1460  Metric:1
```

无法访问 Kubernetes API

很多扩展服务需要访问 Kubernetes API 查询需要的数据（比如 kube-dns、Operator 等）。通常在 Kubernetes API 无法访问时，可以首先通过下面的命令验证 Kubernetes API 是正常的：

```
$ kubectl run curl --image=appropriate/curl -i -t --restart=Never --command -- sh
If you don't see a command prompt, try pressing enter.
/ #
/ # KUBE_TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)
/ # curl -sSk -H "Authorization: Bearer $KUBE_TOKEN" https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT/api/v1/namespaces/default/pods
{
"kind": "PodList",
"apiVersion": "v1",
"metadata": {
  "selfLink": "/api/v1/namespaces/default/pods",
  "resourceVersion": "2285"
},
"items": [
  ...
]
}
```

如果出现超时错误，则需要进一步确认名为 `kubernetes` 的服务以及 endpoints 列表是正常的：

```
$ kubectl get service kubernetes
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
```

```
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   25m
$ kubectl get endpoints kubernetes
NAME         ENDPOINTS           AGE
kubernetes   172.17.0.62:6443   25m
```

然后可以直接访问 endpoints 查看 kube-apiserver 是否可以正常访问。无法访问时通常说明 kube-apiserver 未正常启动，或者有防火墙规则阻止了访问。

但如果出现了 `403 - Forbidden` 错误，则说明 Kubernetes 集群开启了访问授权控制（如 RBAC），此时就需要给 Pod 所用的 ServiceAccount 创建角色和角色绑定授权访问所需要的资源。比如 CoreDNS 就需要创建以下 ServiceAccount 以及角色绑定：

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: coredns
  namespace: kube-system
  labels:
      kubernetes.io/cluster-service: "true"
      addonmanager.kubernetes.io/mode: Reconcile
---
# 2. cluster role
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
    addonmanager.kubernetes.io/mode: Reconcile
  name: system:coredns
rules:
- apiGroups:
  - ""
  resources:
  - endpoints
  - services
  - pods
  - namespaces
  verbs:
  - list
  - watch
---
# 3. cluster role binding
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
    addonmanager.kubernetes.io/mode: EnsureExists
  name: system:coredns
```

```
roleRef:
    apiGroup:  rbac.authorization.k8s.io
    kind:  ClusterRole
    name:  system:coredns
subjects:
-  kind:  ServiceAccount
    name:  coredns
    namespace:  kube-system
---
#  4.  use  created  service  account
apiVersion:  extensions/v1beta1
kind:  Deployment
metadata:
    name:  coredns
    namespace:  kube-system
    labels:
        k8s-app:  coredns
        kubernetes.io/cluster-service:  "true"
        addonmanager.kubernetes.io/mode:  Reconcile
        kubernetes.io/name:  "CoreDNS"
spec:
    replicas:  2
    selector:
        matchLabels:
            k8s-app:  coredns
    template:
        metadata:
            labels:
                k8s-app:  coredns
        spec:
            serviceAccountName:  coredns
            ...
```

原文链接：

https://zhuanlan.zhihu.com/p/34558421

看到这里，相信很多小伙伴也已经了解了K8S常见的网络问题及排错方法，其实，对于任何一门技术点的学习，其理论基础的掌握是第一步，继而实践操作是第二步，然后再自我总结方可全面掌握，那么，今天民工哥给大家带来好书福利了，且看下面的精彩介绍。