K8S多维度问题排查手册[全]（作者: 七哥）

一、Pod 相关问题及排查：

1. **Pod 无法启动，如何查找原因?**

- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令查看该 Pod 的状态信息，检查容器的状态和事件信息，判断是否出现问题。
- 使用 `kubectl logs [pod_name] -n [namespace_name]` 命令查看该 Pod 容器的日志信息，判断是否有错误或异常信息。
- 使用 `kubectl get events --field-selector involvedObject.name=[pod_name] -n [namespace_name]` 查看Pod事件信息，是否有异常事件发生。

2. **Pod 无法连接到其他服务，如何排查?**

- 使用 `kubectl exec -it [pod_name] -n [namespace_name] -- /bin/bash` 命令进入该 Pod 所在的容器，尝试使用 `ping` 或 `telnet` 等命令测试与其他服务的网络连接情况。
- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令检查 Pod 的 `NetworkPolicy` 配置，判断是否阻止了该 Pod 访问其他服务。
- 使用 `kubectl describe service [service_name] -n [namespace_name]` 命令检查目标服务的配置和状态信息，判断是否存在故障。

3. **Pod 运行缓慢或异常，如何排查?**

- 使用 `kubectl top pod [pod_name] -n [namespace_name]` 命令查看该 Pod 的 CPU 和内存使用情况，判断是否存在性能瓶颈。
- 使用 `kubectl exec -it [pod_name] -n [namespace_name] -- /bin/bash` 命令进入该 Pod 所在的容器，使用 `top` 或 `htop` 命令查看容器内部进程的 CPU 和内存使用情况，找出可能存在的瓶颈。
- 使用 `kubectl logs [pod_name] -n [namespace_name]` 命令查看该 Pod 容器的日志信息，寻找可能的错误或异常信息。

4. **Pod 无法被调度到节点上运行，如何排查?**

- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令查看 Pod 的调度情况，判断是否存在资源不足、调度策略等问题。
- 使用 `kubectl get nodes` 和 `kubectl describe node [node_name]` 命令查看所有节点的资源使用情况，判断是否存在节点资源不足或故障的情况。
- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令检查 Pod 所需的标签和注释，以及节点的标签和注释，判断是否匹配。

5. **Pod 状态一直是 `Pending`，怎么办?**

- 使用 `kubectl get pods -n <namespace>` 命令检查 Pod 的状态和事件，确定 Pod 处于何种状态以及是否有任何错误或警告信息。
- 检查 Pod 的描述文件（YAML 或 JSON），确保各项字段（如镜像名称、资源请求、端口等）配置正确。
- 如果 Pod 需要特定类型的节点（如 GPU 节点），确认集群中是否有符合条件的节点可用。
- 检查 Pod 所需的资源配额（如 CPU、内存）是否已经达到上限，可以使用 `kubectl describe pod <pod-name> -n <namespace>` 查看详细信息。

- 检查 Pod 所需的存储卷是否可用，确保没有引发挂载错误。
- 如果是调度问题，可以通过以下方式解决：
    - 确保有足够的节点资源满足该 Pod 调度需求
    - 检查该节点的 taints 和 tolerations 是否与 Pod 的 selector 匹配
    - 调整 Pod 的调度策略，如使用 NodeSelector、Affinity 等

6. **Pod 无法访问外部服务，怎么办？**

- 查看 Pod 中的 DNS 配置是否正确
- 检查 Pod 所在的命名空间中是否存在 Service 服务
- 确认该 Pod 是否具有网络访问权限
- 查看 Pod 所在的节点是否有对外的访问权限
- 检查网络策略是否阻止了 Pod 对外的访问

7. **Pod 启动后立即退出，怎么办？**

- 查看该 Pod 的事件信息：`kubectl describe pod <pod-name>`
- 查看该 Pod 的日志：`kubectl logs <pod-name>`
- 检查容器镜像是否正确、环境变量是否正确、入口脚本是否正常
- 尝试在本地使用相同的镜像运行该容器，查看是否有报错信息，如执行 `docker run <image-name>`

8. **Pod 启动后无法正确运行应用程序，怎么办？**

- 查看 Pod 中的应用程序日志：`kubectl logs <pod-name>`
- 查看该 Pod 的事件信息：`kubectl describe pod <pod-name>`
- 检查应用程序的配置文件是否正确
- 检查应用程序的依赖是否正常
- 尝试在本地使用相同的镜像运行该容器，查看是否有报错信息，如执行 `docker run <image-name>`
- 确认该应用程序是否与 Pod 的资源限制相符

9. **Kubernetes 集群中的 Service 不可访问，怎么办？**

- 检查coreDNS服务是否可用；
- 查看dns配置文件是否正确（/etc/resolv.conf）；
- 业务层面svc的port是否正确；
- svc是否正确关联到后端的pod；
- 业务pod是否正常工作；
- CNI网络组件（flannel，calico）组件是否有问题；
- kube-proxy组件是否正常；
- 是否已经创建相关iptables规则或ipvs路由；
- **附：**Service工作流程图：

10. **Pod 启动后立即终止或 CrashLoopBackOff 状态：**

- 使用 `kubectl get pods -n <namespace>` 命令检查 Pod 的状态和事件，查看是否有任何错误或警告信息。
- 使用 `kubectl logs <pod-name> -n <namespace>` 命令查看 Pod 的日志输出，尤其关注最后几行的错误信息。
- 确认 Pod 的生命周期钩子（如 postStart、preStop）是否正确配置，是否有引发异常的操作。
- 确认 Pod 执行的命令或容器启动命令是否正确，是否会导致容器意外退出。
- 检查容器的资源使用情况是否超过 Pod 的资源限制，尤其是内存限制。

11. **Pod 内部服务无法访问或网络连接问题：**

- 使用 `kubectl get pods -n <namespace>` 命令检查 Pod 的状态和事件，查看是否有任何错误或警告信息。
- 确认 Pod 所属的 Service 是否已经创建，且与 Pod 使用的端口和协议匹配。
- 检查 Pod 内部的 DNS 配置，确保能够解析其他服务的域名。
- 使用 `kubectl exec <pod-name> -n <namespace> -- <command>` 命令进入 Pod 内部，手动测试容器之间的网络连通性。

12. **Pod 与存储卷之间的问题：**

- 使用 `kubectl get pods -n <namespace>` 命令检查 Pod 的状态和事件，查看是否有任何错误或警告信息。
- 确认存储卷是否已经正确地绑定到 Pod 上，可以使用 `kubectl describe pod <pod-name> -n <namespace>` 查看详细信息。
- 使用 `kubectl exec <pod-name> -n <namespace> -- <command>` 命令进入 Pod 内部，手动测试存储卷是否能够正常挂载和访问。
- 检查存储卷提供程序（如 NFS、AWS EBS）的配置是否正确，并确保其可用性。
- 确保存储卷访问模式（如 ReadWriteOnce、ReadOnlyMany）与应用程序的要求相匹配。

二、Node 相关问题及排查：

1. **Node 状态异常，如何排查？**

- 使用 `kubectl get nodes` 命令查看集群中所有节点的状态和信息，判断是否存在故障。
- 使用 `kubectl describe node [node_name]` 命令查看目标节点的详细信息，包括 CPU、内存、磁盘等硬件资源的使用情况，判断是否存在性能瓶颈。
- 使用 `kubectl get pods -o wide --all-namespaces` 命令查看集群中所有 Pod 的状态信息，判断是否有 Pod 运行在目标节点上导致资源紧张。

2. **Node 上运行的 Pod 无法访问网络，如何排查？**

- 使用 `kubectl describe node [node_name]` 命令查看目标节点的信息，检查节点是否正常连接到网络。
- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令查看 Pod 所运行的节点信息，判断是否因为节点状态异常导致网络访问失败。
- 使用 `kubectl logs [pod_name] -n [namespace_name]` 命令查看 Pod 容器的日志信息，寻找可能的错误或异常信息。

3. **Node 上的 Pod 无法访问存储，如何排查？**

- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令检查 Pod 的 `volumes` 配置信息，判断是否存在存储挂载失败的情况。
- 使用 `kubectl exec -it [pod_name] -n [namespace_name] -- /bin/bash` 命令进入 Pod 所在的容器，尝试使用 `ls` 和 `cat` 等命令访问挂载的文件系统，判断是否存在读写错误。
- 使用 `kubectl describe persistentvolumeclaim [pvc_name] -n [namespace_name]` 命令查看相关 PVC 配置和状态信息，判断是否存在故障。

4. **存储卷挂载失败，如何处理?**

- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令检查 Pod 的 `volumes` 配置信息，判断是否存在存储卷定义错误。
- 使用 `kubectl describe persistentvolumeclaim [pvc_name] -n [namespace_name]` 命令检查 PVC 的状态和信息，判断是否存在存储配额不足或存储资源故障等原因。
- 如果是 NFS 或 Ceph 等网络存储，需要确认网络连接是否正常，以及存储服务器的服务是否正常。

5. **Node 节点加入 Kubernetes 集群后无法被调度，怎么办?**

- 检查该节点的 taints 和 tolerations 是否与 Pod 的 selector 匹配
- 检查该节点的资源使用情况是否满足 Pod 的调度要求
- 确保该节点与 Kubernetes API server 的连接正常

6. **Kubernetes 集群中的 PersistentVolume 挂载失败，怎么办?**

- 检查 PersistentVolume 和 Pod 之间的匹配关系是否正确
- 检查 PersistentVolumeClaim 中的 storageClassName 是否与 PersistentVolume 的 storageClassName 匹配
- 检查节点存储配置和 PersistentVolume 的定义是否正确
- 自动供给层面的权限是否已经给到位

三、集群层面问题及排查：

1. **集群中很多 Pod 运行缓慢，如何排查?**

- 使用 `kubectl top pod -n [namespace_name]` 命令查看所有 Pod 的 CPU 和内存使用情况，判断是否存在资源瓶颈。
- 使用 `kubectl get nodes` 和 `kubectl describe node [node_name]` 命令查看所有节点的资源使用情况，判断是否存在单个节点资源紧张的情况。
- 使用 `kubectl logs [pod_name] -n [namespace_name]` 命令查看 Pod 容器的日志信息，寻找可能的错误或异常信息。

2. **集群中某个服务不可用，如何排查?**

- 使用 `kubectl get pods -n [namespace_name]` 命令查看相关服务的所有 Pod 的状态信息，判断是否存在故障。
- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令检查 Pod 的网络连接和存储访问等问题，寻找故障原因。
- 使用 `kubectl describe service [service_name] -n [namespace_name]` 命令查看服务的配置和状态信息，判断是否存在故障。

3. **集群中的 Node 和 Pod 不平衡，如何排查?**

- 使用 `kubectl get nodes` 和 `kubectl get pods -o wide --all-namespaces` 命令查看所有 Node 和 Pod 的状态信息，判断是否存在分布不均的情况。
- 使用 `kubectl top pod -n [namespace_name]` 命令查看所有 Pod 的 CPU 和内存使用情况，判断是否存在资源瓶颈导致 Pod 分布不均。
- 使用 `kubectl describe pod [pod_name] -n [namespace_name]` 命令查看 Pod 所运行的节点信息，并使用 `kubectl describe node [node_name]` 命令查看相关节点的状态信息，判断是否存在节点不平衡的情况。
- 使用 `kubectl describe pod / node [node_name]` 查看当前Pod / Node上是否有相关的亲和或反亲和策略导致固定调度。

4. **集群中某个节点宕机，如何处理?**

- 使用 `kubectl get nodes` 命令检查节点状态，找到异常节点。
- 使用 `kubectl drain [node_name] --ignore-daemonsets` 命令将节点上的 Pod 驱逐出去，并将其部署到其他节点上。添加 `--ignore-daemonsets` 参数可以忽略 DaemonSet 资源。
- 如果需要对节点进行维护或替换硬件：
  - 先将节点设置为不可以调度 `kubectl cordon [node_name]`
  - 再通过 `kubectl drain [node_name] --ignore-daemonsets` 命令将节点上的 Pod 驱逐出去，并将其部署到其他节点上。
  - 然后再次 `kubectl delete node [node_name]` 安全的进行节点下线。

5. **Kubernetes API Server 不可用，如何排查?**

- 使用 `kubectl cluster-info` 命令查看集群状态，判断是否存在 API Server 不可用的情况。
- 使用 `kubectl version` 命令查看集群版本，确认 Kubernetes API Server 和 kubelet 版本是否匹配。
- 使用 `systemctl status kube-apiserver` 命令检查 API Server 运行状态，确认是否存在故障或错误。
- 结合apiServer所在的节点查看系统层面的日志，进一步定位问题点。

6. **Kubernetes 命令执行失败，怎么办?**

- 检查 Kubernetes API server 是否可用：`kubectl cluster-info`
- 检查当前用户对集群的权限是否足够：`kubectl auth can-i <verb> <resource>`
- 检查 kubeconfig 文件中的登录信息是否正确：`kubectl config view`

7. **Kubernetes master 节点不可用，怎么办?**

- 检查 kube-apiserver、kube-scheduler、kube-controller-manager 是否都在运行状态
- 检查 etcd 存储系统是否可用
- 尝试重新启动 master 节点上的 kubelet 和容器运行时

8. **Kubernetes 集群绕过了 LoadBalancer，直接访问 Pod，怎么办?**

- 检查 Service 和 Pod 的通信是否使用了 ClusterIP 类型的 Service
- 确认该 Service 的 selector 是否匹配到了正确的 Pod

9. **Kubernetes 集群中的 Deployment 自动更新失败，怎么办?**

- 检查更新策略是否设置正确，如 `rollingUpdate` 或 `recreate`

- 检查 Kubernetes API server 和 kubelet 之间的连接是否正常
- 检查 Pod 的定义是否正确

10. **Kubernetes 集群中的状态检查错误，怎么办？**

- 检查节点日志和事件信息，并确认错误类型
- 确认该状态检查是否与 kubelet 的版本兼容
- 尝试升级 kubelet 和容器运行时等组件

11. **Kubernetes 集群中的授权配置有误，怎么办？**

- 检查 RoleBinding 和 ClusterRoleBinding 定义是否正确
- 检查用户或服务账号所绑定的角色是否正确
- 检查 kubeconfig 文件中的用户和访问权限是否正确

12. **Kubernetes 集群无法连接 etcd 存储系统，怎么办？**

- 检查 etcd 存储系统是否正常运行
- 检查 kube-apiserver 配置文件中 etcd 的连接信息是否正确
- 尝试手动连接 etcd 集群，如执行 `etcdctl cluster-health`

四、Pod常遇状态异常排查：

一般来说，无论 Pod 处于什么异常状态，都可以执行以下命令来查看 Pod 的状态：

```
$ kubectl get pod <pod-name> -o yaml       查看 Pod 的配置是否正确
$ kubectl describe pod <pod-name> -n命名空间          查看 Pod 的事件
$ kubectl logs <pod-name> [-c <container-name>]      查看容器日志
```

如上这些事件和日志通常都会有助于排查 Pod 发生的问题。

1、Pod 一直处于 Pending 状态

Pending 说明 Pod 还没有调度到某个 Node 上面。可以通过 `kubectl describe pod <pod-name>` 命令查看到当前 Pod 的事件，进而判断为什么没有调度。

可能的原因包括：

- 资源不足，集群内所有的 Node 都不满足该 Pod 请求的 CPU、内存、GPU 等资源；
- HostPort 已被占用，通常推荐使用 Service 对外开放服务端口；

2、Pod 一直处于 Waiting 或 ContainerCreating 状态

首先还是通过 `kubectl describe pod <pod-name>` 命令查看到当前 Pod 的事件。可能的原因包括：

- 镜像拉取失败，比如：
  - 配置了错误的镜像；
  - Kubelet 无法访问镜像（国内环境访问 gcr.io 需要特殊处理）；
  - 私有镜像的密钥配置错误；
  - 镜像太大，拉取超时（可以适当调整 kubelet 的 `--image-pull-progress-deadline` 和 `--runtime-request-timeout` 选项）；
- CNI 网络错误，一般需要检查 CNI 网络插件的配置，比如：
  - 无法配置 Pod 网络；

    - 无法分配 IP 地址;
- 容器无法启动，需要检查是否打包了正确的镜像或者是否配置了正确的容器参数;

3、Pod 处于 ImagePullBackOff 状态

这通常是镜像名称配置错误或者私有镜像的密钥配置错误导致。

这种情况可以使用 `docker pull <image>` 来验证镜像是否可以正常拉取。

如果是私有镜像，需要首先创建一个 docker-registry 类型的 Secret

```
$ kubectl create secret docker-registry my-secret --docker-
server=DOCKER_REGISTRY_SERVER --docker-username=DOCKER_USER --docker-
password=DOCKER_PASSWORD --docker-email=DOCKER_EMAIL
```

然后在容器中引用这个 Secret:

```
spec:
  containers:
  - name: private-reg-container
    image: <your-private-image>
  imagePullSecrets:
  - name: my-secret
```

4、Pod 一直处于 CrashLoopBackOff 状态

`CrashLoopBackOff` 状态说明容器曾经启动了，但又异常退出了。此时可以先查看一下容器的日志

```
$ kubectl logs <pod-name>
$ kubectl logs --previous <pod-name>
```

这里可以发现一些容器退出的原因，比如:

- 容器进程退出;
- 健康检查失败退出;

此时如果还未发现线索，还可以到容器内执行命令来进一步查看退出原因

```
$ kubectl exec cassandra -- cat /var/log/cassandra/system.log
```

如果还是没有线索，那就需要 SSH 登录该 Pod 所在的 Node 上，查看 Kubelet 或者 Docker 的日志进一步排查了

查询 pod 在哪台 Node:

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

5、Pod 处于 Error 状态

通常处于 Error 状态说明 Pod 启动过程中发生了错误。常见的原因包括:

- 依赖的 ConfigMap、Secret 或者 PV 等不存在;
- 请求的资源超过了管理员设置的限制，比如超过了 LimitRange 等;
- 违反集群的安全策略，比如违反了 PodSecurityPolicy 等;

- 容器无权操作集群内的资源，比如开启 RBAC 后，需要为 ServiceAccount 配置角色绑定；

6、Pod 处于 Terminating 或 Unknown 状态

Kubernetes 不会因为 Node 失联而删除其上正在运行的 Pod，而是将其标记为 Terminating 或 Unknown 状态。想要删除这些状态的 Pod 有三种方法：

- 从集群中删除该 Node。使用公有云时，kube-controller-manager 会在 VM 删除后自动删除对应的 Node。而在物理机部署的集群中，需要管理员手动删除 Node（如 `kubectl delete node <node-name>`。
- Node 恢复正常。Kubelet 会重新跟 kube-apiserver 通信确认这些 Pod 的期待状态，进而再决定删除或者继续运行这些 Pod。
- 用户强制删除。用户可以执行 `kubectl delete pods <pod> --grace-period=0 --force` 强制删除 Pod。除非明确知道 Pod 的确处于停止状态（比如 Node 所在 VM 或物理机已经关机），否则不建议使用该方法。特别是 StatefulSet 管理的 Pod，强制删除容易导致脑裂或者数据丢失等问题。

五、Kubernetes故障排查指南-分析容器退出状态码

5.1、Pod status 状态解释

```
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：镜像正在重试拉取
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
m.internalLifecycle.PreStartContainer：执行hook报错
RunContainerError：启动容器失败
PostStartHookError：执行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
```


5.2、容器 Exit Code

5.2.1、容器退出状态码的区间

- 必须在 0-255 之间
- 0 表示正常退出
- 外界中断将程序退出的时候状态码区间在 129-255，(操作系统给程序发送中断信号，比如 kill -9 是 SIGKILL，Ctrl+c 是 SIGINT)
- 一般程序自身原因导致的异常退出状态区间在 1-128 (这只是一般约定，程序如果一定要用129-255的状态码也是可以的)
  注意：有时我们会看到代码中有 exit(-1)，这时会自动做一个转换，最终输出的结果还是会在 0-255 之间。

转换公式如下，code 表现退出的状态码：

当指定的退出时状态码为负数，转换公式如下：

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

当指定的退出时状态码为正数，转换公式如下：

```
code % 256
```

5.2.2、常见的容器退出状态码解释

EXIT CODE 0

- 退出代码0表示特定容器没有附加前台进程
- 该退出代码是所有其他后续退出代码的例外
- 如果开发人员想要在容器完成其工作后自动停止其容器，则使用此退出代码。比如：kubernetes job 在执行完任务后正常退出码为0

EXIT CODE 1

- 程序错误，或者Dockerfile中引用不存在的文件，如 entrypoint 中引用了错误的包
- 程序错误可以很简单，例如 “除以0”，也可以很复杂，比如空引用或者其他程序 crash

EXIT CODE 137

- 表明容器收到了 SIGKILL 信号，进程被杀掉，对应kill -9
- 引发 SIGKILL 的是docker kill。这可以由用户或由docker守护程序来发起，手动执行：docker kill
- 137 比较常见，如果 pod 中的limit 资源设置较小，会运行内存不足导致 OOMKilled，此时state 中的 “OOMKilled” 值为true，你可以在系统的 dmesg -T 中看到 oom 日志

EXIT CODE 139

- 表明容器收到了 SIGSEGV 信号，无效的内存引用，对应kill -11
- 一般是代码有问题，或者 docker 的基础镜像有问题

EXIT CODE 143

- 表明容器收到了 SIGTERM 信号，终端关闭，对应kill -15
- 一般对应 docker stop 命令
- 有时docker stop也会导致Exit Code 137。发生在与代码无法处理 SIGTERM 的情况下，docker进程等待十秒钟然后发出 SIGKILL 强制退出。

不常用的一些 EXIT CODE

- Exit Code 126: 权限问题或命令不可执行
- Exit Code 127: Shell脚本中可能出现错字且字符无法识别的情况
- Exit Code 1 或 255：因为很多程序员写异常退出时习惯用 exit(1) 或 exit(-1)，-1 会根据转换规则转成 255。这个一般是自定义 code，要看具体逻辑。

对课程感兴趣&想获取更多隐形福利添加如下微信：👇