面试基本信息

基础信息

- 面试岗位：运维工程师(k8s方向)、运维开发工程师、SRE工程师
- 工作经验：5年
- 薪资范围：年薪40万左右，上海
- 面试时间：2024年12月
- 面试公司:中大型公司
-

面试题汇总

注：☆表示多次出现过的高频面试题

Kubernetes

- 谈谈你对k8s的理解 ☆
- k8s集群架构是什么 ☆
- 简述Pod创建过程
- k8s中的资源限制和请求(requests/limits)有什么区别？
- 简述删除一个Pod流程
- 不同node上的Pod之间的通信过程
- pod创建Pending状态的原因
- 简述k8s中的认证和授权机制 ☆
- deployment和statefulset区别
- kube-proxy有什么作用
- StatefulSet和DaemonSet的使用场景分别是什么？☆
- kube-proxy怎么修改ipvs规则
- ipvs为什么比iptables效率高
- pod之间访问不通怎么排查☆
- k8s中Network Policy的实现原理
- 如何实现k8s集群的备份和恢复？
- 探针有哪些？探测方法有哪些？
- pod健康检查失败可能的原因和排查思路
- k8s的Service是什么☆
- metrics-server采集指标数据链路
- 谈谈你对k8s中ConfigMap和Secret的理解和使用场景 ☆
- k8s服务发现有哪些方式？
- pod几种常用状态
- Pod 生命周期的钩子函数
- k8s中的污点(Taints)和容忍(Tolerations)是什么？
- Calico和flannel区别☆
- calico网络原理、组组网方式
- Network Policy使用场景
- 如何处理k8s集群的性能问题？
- kubectl exec 实现的原理
- cgroup中限制CPU的方式有哪些
- kubeconfig存放内容
- pod DNS解析流程☆
- k8s中的存储类型有哪些？PV和PVC的工作原理是什么？
- traefik对比nginx ingress优点
- Harbor有哪些组件
- Harbor高可用怎么实现

- ETCD调优
- 如何实现k8s的滚动更新和回滚？
- 假设k8s集群规模上千，需要注意的问题有哪些？
- 节点NotReady可能的原因？会导致哪些问题？☆
- service和endpoints是如何关联的？
- ReplicaSet、Deployment功能是怎么实现的？
- scheduler调度流程
- HPA怎么实现的☆
- request limit底层是怎么限制的☆
- k8s中的RBAC是如何实现的？
- helm工作原理是什么？
- helm chart rollback实现过程是什么？
- velero备份与恢复流程是什么
- docker网络模式
- docker和container区别☆
- 如何排查k8s集群中的网络问题？
- 如何减小dockerfile生成镜像体积？
- k8s日志采集方案
- Pause容器的用途☆
- k8s证书过期怎么更新
- K8S QoS等级☆
- k8s中的Init容器有什么作用？
- k8s节点维护注意事项
- 如何确保k8s集群的安全性？☆
- Headless Service和ClusterIP区别☆
- Linux容器技术的基础原理
- Kubernetes Pod的常见调度方式
- kubernetes Ingress原理☆
- 如何进行k8s集群的升级？需要注意什么？☆
- Kubernetes各模块如何与API Server通信
- kubelet监控worker节点如何实现
- 容器时区不一致如何解决？

Kubernetes面试题答案

谈谈你对k8s的理解 ☆

Kubernetes（k8s）是一个开源的容器编排平台，主要功能包括：

1. 容器编排：自动化部署、扩展和管理容器化应用
2. 服务发现和负载均衡：通过Service抽象提供服务发现和负载均衡
3. 存储编排：自动挂载存储系统
4. 自动部署和回滚：支持渐进式更新和自动回滚
5. 自我修复：自动重启失败容器，替换和重新部署节点
6. 密钥和配置管理：管理敏感信息和配置文件

k8s集群架构是什么 ☆

Kubernetes集群主要分为控制平面（Master）和工作节点（Node）：

控制平面组件：

1. API Server：集群的统一入口，提供REST API
2. etcd：键值数据库，存储集群所有数据
3. Controller Manager：维护集群状态的控制器集合
4. Scheduler：负责Pod的调度

工作节点组件：

1. kubelet：管理节点上容器的生命周期
2. kube-proxy：维护节点的网络规则
3. Container Runtime：容器运行时（如Docker）

简述Pod创建过程

Pod的创建流程如下：

1. 用户通过kubectl或API创建Pod
2. API Server接收请求，将数据存入etcd
3. Scheduler通过API Server监听到未调度的Pod
4. Scheduler根据调度算法为Pod选择合适的Node
5. Scheduler将调度结果更新到API Server
6. 目标Node上的kubelet监听到Pod调度信息
7. kubelet通过Container Runtime创建容器
8. kubelet更新Pod状态到API Server

k8s中的资源限制和请求(requests/limits)有什么区别？

1. Requests：
    - 容器需要的最小资源量
    - 用于Pod调度，确保节点有足够资源
    - 不会限制容器使用更多资源
2. Limits：
    - 容器可以使用的最大资源量
    - 超过限制会被限制（CPU）或终止（内存）
    - 用于资源隔离和防止资源滥用

简述删除一个Pod流程

1. 用户发起删除Pod请求
2. API Server更新Pod的删除时间戳
3. Pod进入Terminating状态
4. kubelet开始Pod删除流程：
    - 执行PreStop钩子（如果配置）
    - 向容器主进程发送SIGTERM信号
    - 等待宽限期（默认30秒）
    - 如果容器仍在运行，发送SIGKILL信号
5. 清理Pod相关资源（卷等）
6. API Server从etcd中删除Pod对象

不同node上的Pod之间的通信过程

Pod间通信主要通过CNI插件实现，以Flannel为例：

1. Pod1发出请求，数据包先到达所在Node的flannel0虚拟网卡
2. flannel0将数据包封装成UDP包（VXLAN模式）
3. 通过物理网卡发送到目标Node
4. 目标Node的flannel0解封装数据包
5. 根据路由规则转发到目标Pod

pod创建Pending状态的原因

常见原因包括：

1. 资源不足：CPU、内存不满足requests要求
2. PV/PVC未就绪：存储卷问题
3. 调度失败：
    - 节点污点（Taints）与Pod不兼容
    - 节点标签（Label）不匹配
    - 节点资源不足
4. 镜像拉取失败
5. 网络插件未就绪

简述k8s中的认证和授权机制 ☆

认证（Authentication）：

1. 客户端证书认证
2. Bearer Token认证
3. ServiceAccount
4. WebHook认证

授权（Authorization）：

1. RBAC（基于角色的访问控制）
    - Role/ClusterRole：定义权限
    - RoleBinding/ClusterRoleBinding：将角色绑定到用户
2. Node授权
3. Webhook授权
4. ABAC（基于属性的访问控制）

deployment和statefulset区别

主要区别：

1. 身份标识：
    - Deployment的Pod名称随机
    - StatefulSet的Pod名称固定且有序
2. 存储：
    - Deployment的Pod共享存储
    - StatefulSet的Pod有独立存储
3. 扩缩容：
    - Deployment随机创建/删除
    - StatefulSet按顺序创建/删除
4. 网络：
    - Deployment使用Service负载均衡
    - StatefulSet可以使用Headless Service提供稳定网络标识

kube-proxy有什么作用

kube-proxy主要功能：

1. 实现Service的网络代理和负载均衡
2. 维护网络规则（iptables/ipvs）
3. 转发流量到后端Pod
4. 提供集群内服务发现和负载均衡
5. 支持三种代理模式：
    - userspace（已弃用）

    - iptables（默认）
    - ipvs（性能更好）

StatefulSet和DaemonSet的使用场景分别是什么？☆

StatefulSet使用场景：

1. 需要稳定持久化存储的应用（如数据库）
2. 需要稳定网络标识的应用
3. 有序部署、扩展的应用
4. 典型应用：
    - MySQL、MongoDB等数据库
    - ZooKeeper、etcd等有状态服务

DaemonSet使用场景：

1. 需要在每个节点上运行的守护进程
2. 节点监控和日志收集
3. 典型应用：
    - 日志收集：fluentd、logstash
    - 节点监控：node-exporter
    - 网络插件：calico、flannel

kube-proxy怎么修改ipvs规则

1. 监听Service和Endpoints变化
2. 创建ipvs虚拟服务器（virtual server）
3. 更新ipvs规则： # 查看ipvs规则 ipvsadm -ln # 修改默认调度算法 kubectl edit configmap kube-proxy -n kube-system # 重启kube-proxy kubectl delete pod -l k8s-app=kube-proxy -n kube-system

ipvs为什么比iptables效率高

1. 实现原理不同：
    - iptables：基于链表实现
    - ipvs：基于哈希表实现
2. 性能差异：
    - iptables规则是线性查找，时间复杂度O(n)
    - ipvs使用哈希表，时间复杂度O(1)
3. 支持更多负载均衡算法：
    - rr：轮询
    - wrr：加权轮询
    - lc：最小连接
    - wlc：加权最小连接
4. 同步性能更好：
    - iptables规则同步需要全量更新
    - ipvs支持增量更新

pod之间访问不通怎么排查☆

排查步骤：

1. 检查网络策略（NetworkPolicy）

    kubectl get networkpolicy

2. 检查DNS解析

kubectl exec -it <pod> -- nslookup <service-name>

3. 检查Service配置

   kubectl get svc kubectl describe svc <service-name>

4. 检查Pod状态和网络

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

5. 检查网络插件状态

   kubectl get pods -n kube-system | grep calico

6. 使用工具排查

   - ping/telnet测试连通性
   - tcpdump抓包分析
   - curl测试服务访问

k8s中Network Policy的实现原理

1. 基本原理：

   - 通过网络插件实现（如Calico）
   - 使用iptables或ipvs规则控制流量

2. 规则类型：

   - Ingress：入站规则
   - Egress：出站规则

3. 选择器：

   - podSelector：选择Pod
   - namespaceSelector：选择命名空间
   - ipBlock：选择IP范围

4. 实现流程：

   - API Server接收NetworkPolicy对象
   - 网络控制器监听变化
   - 转换为具体网络规则
   - 下发到各节点执行

如何实现k8s集群的备份和恢复？

主要备份内容：

1. etcd数据

备份 ETCDCTL_API=3 etcdctl snapshot save snapshot.db # 恢复 ETCDCTL_API=3 etcdctl snapshot restore snapshot.db

1. 使用工具：

   - Velero：备份集群资源和持久卷

    - etcd-operator：自动化etcd备份
    - Rancher Backup Operator：备份Rancher资源

2. 定期备份策略：
    - 创建CronJob定时备份
    - 设置备份保留期
    - 备份文件异地存储

探针有哪些？探测方法有哪些？

三种探针类型：

1. livenessProbe：
    - 检查容器是否运行
    - 失败时重启容器
2. readinessProbe：
    - 检查容器是否就绪
    - 失败时从Service负载均衡中移除
3. startupProbe：
    - 检查容器是否启动完成
    - 失败时重启容器

探测方法：

1. exec：执行命令
2. httpGet：发送HTTP GET请求
3. tcpSocket：TCP端口检查

pod健康检查失败可能的原因和排查思路

常见原因：

1. 应用程序问题：
    - 服务未启动
    - 程序崩溃
    - 端口配置错误
2. 资源问题：
    - 内存不足
    - CPU负载过高
    - 磁盘空间不足

排查思路：

1. 查看Pod状态和事件

kubectl describe pod <pod-name>

1. 查看容器日志

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

1. 检查探针配置
2. 进入容器排查

kubectl exec -it <pod-name> -- /bin/sh

k8s的Service是什么？☆

Service是k8s中的服务抽象，主要功能：

1. 提供固定访问入口
2. 实现负载均衡
3. 服务发现

类型：

1. ClusterIP：集群内访问
2. NodePort：对外暴露服务
3. LoadBalancer：使用云服务负载均衡
4. ExternalName：外部服务映射

工作原理：

1. 创建Service时生成虚拟IP
2. kube-proxy维护转发规则
3. 通过标签选择器关联Pod
4. 自动负载均衡

metrics-server采集指标数据链路

数据采集流程：

1. kubelet通过cAdvisor采集容器指标
2. metrics-server定期从kubelet获取数据
3. 聚合数据并存储在内存中
4. 通过API提供指标查询

数据类型：

1. 容器指标（CPU、内存）
2. 节点指标
3. Pod指标

使用场景：

1. HPA自动扩缩容
2. kubectl top命令
3. 资源监控

谈谈你对k8s中ConfigMap和Secret的理解和使用场景 ☆

ConfigMap：

1. 用途：
    - 存储非敏感配置信息
    - 环境变量配置

- 配置文件管理

2. 使用方式：
    - 环境变量注入
    - 挂载为文件
    - 命令行参数
3. 特点：
    - 可以动态更新
    - 支持大容量配置
    - 明文存储

Secret：

1. 用途：
    - 存储敏感信息
    - 证书管理
    - 密码凭证
2. 类型：
    - Opaque：通用密钥
    - kubernetes.io/tls：TLS证书
    - kubernetes.io/dockerconfigjson：镜像仓库认证
3. 特点：
    - Base64编码
    - 内存存储
    - 按需挂载

k8s服务发现有哪些方式？

1. 环境变量：
    - Pod启动时自动注入Service信息
    - 格式：SERVICE_NAME_SERVICE_HOST
2. DNS：
    - CoreDNS服务
    - 格式：service-name.namespace.svc.cluster.local
3. Service：
    - ClusterIP
    - Headless Service
4. API服务发现：
    - 通过API Server查询Service
    - 客户端自行实现负载均衡

pod几种常用状态

1. Pending：
    - Pod已创建，等待调度
    - 下载镜像中
2. Running：
    - 所有容器已创建
    - 至少一个容器运行中
3. Succeeded：
    - 所有容器成功终止
    - 不会重启
4. Failed：
    - 所有容器已终止
    - 至少一个容器失败退出
5. Unknown：
    - Pod状态无法获取
    - 通常是节点通信问题
6. CrashLoopBackOff：
    - 容器反复重启
    - 可能是配置错误

Pod 生命周期的钩子函数

1. postStart：
    - 容器创建后立即执行
    - 与容器ENTRYPOINT并行执行
    - 执行失败会重启容器
2. preStop：
    - 容器终止前执行
    - 阻塞式执行
    - 用于优雅关闭

执行方式：

1. Exec：执行命令
2. HTTP：发送HTTP请求

使用场景：

1. 资源清理
2. 通知其他服务
3. 优雅关闭应用

k8s中的污点(Taints)和容忍(Tolerations)是什么？

污点(Taints)：

1. 作用：
    - 拒绝不符合条件的Pod调度到节点
    - 驱逐不符合条件的Pod
2. 效果：
    - NoSchedule：不调度
    - PreferNoSchedule：尽量不调度
    - NoExecute：驱逐现有Pod
3. 设置方法：

kubectl taint nodes node1 key=value:NoSchedule

容忍(Tolerations)：

1. 作用：
    - 允许Pod调度到有特定污点的节点
    - 防止Pod被驱逐
2. 配置示例：

tolerations: - key: "key" operator: "Equal" value: "value" effect: "NoSchedule"

Calico和flannel区别☆

Flannel：

1. 特点：
    - 简单易用
    - 只专注网络连通性
    - 性能一般
2. 网络模式：
    - VXLAN
    - host-gw
    - UDP（已弃用）

Calico：

1. 特点：
    - 性能更好
    - 支持网络策略
    - 可扩展性强
2. 网络模式：
    - BGP
    - IPIP
    - VXLAN

3. 额外功能：
    - 网络策略实现
    - 流量加密
    - 流量监控

calico网络原理、组网方式

组件：

1. Felix：
    - 路由配置
    - ACL规则管理
    - 接口管理
2. BIRD：
    - BGP客户端
    - 路由信息交换
3. etcd：
    - 存储网络元数据
    - 配置信息

网络模式：

1. BGP模式：
    - 直接路由
    - 性能最好
    - 要求网络支持BGP
2. IPIP模式：
    - 隧道封装
    - 跨网段支持
    - 性能略低
3. VXLAN模式：
    - 类似Flannel
    - 兼容性好

Network Policy使用场景

1. 安全隔离：
    - 限制Pod间通信
    - 实现网络分段
    - 多租户隔离
2. 常见场景：
    - 限制特定命名空间间的访问
    - 只允许特定端口访问
    - 限制外部访问
    - 实现零信任网络

3. 配置示例：

apiVersion: networking.k8s.io/v1 kind: NetworkPolicy metadata: name: test-network-policy spec: podSelector: matchLabels: role: db policyTypes: - Ingress ingress: - from: - podSelector: matchLabels: role: frontend ports: - protocol: TCP port: 6379

如何处理k8s集群的性能问题？

1. 资源监控：
    - 使用Prometheus+Grafana
    - 监控节点和Pod资源使用
    - 设置告警阈值
2. 性能优化：
    - 合理设置资源限制
    - 优化镜像大小
    - 使用本地存储
    - 调整kubelet参数
3. 常见问题处理：
    - CPU瓶颈：
        - 检查limit设置
        - 优化应用代码
    - 内存问题：
        - 排查内存泄漏
        - 调整JVM参数
    - 网络问题：
        - 使用IPVS模式
        - 优化CNI配置

kubectl exec 实现的原理

1. 执行流程：
    - kubectl发送API请求
    - apiserver验证请求
    - apiserver创建流式连接
    - kubelet执行命令
    - 返回结果
2. 具体步骤：
    - 创建SPDY连接
    - 建立双向数据流
    - 转发stdin/stdout/stderr
    - 处理终端resize事件
3. 涉及组件：
    - kubectl
    - apiserver
    - kubelet
    - container runtime

cgroup中限制CPU的方式有哪些

1. cpu.shares：
    - 相对权重
    - 默认值1024
    - 按比例分配CPU
2. cpu.cfs_period_us：
    - CPU使用周期
    - 默认100ms
3. cpu.cfs_quota_us：
    - 周期内最大使用时间
    - -1表示不限制
4. cpuset.cpus：
    - 限制使用的CPU核心
    - 如"0-3,5,7"
5. 示例：

限制CPU使用率为50% echo 50000 > /sys/fs/cgroup/cpu/container/cpu.cfs_quota_us
echo 100000 > /sys/fs/cgroup/cpu/container/cpu.cfs_period_us

kubeconfig存放内容

1. 主要配置：
    - clusters：集群信息
    - users：用户认证信息
    - contexts：集群和用户的关联关系
    - current-context：当前使用的上下文
2. 示例结构：

apiVersion: v1 kind: Config clusters: - name: kubernetes cluster: server: https://kubernetes:6443 certificate-authority-data: ... users: - name: admin user: client-certificate-data: ... client-key-data: ... contexts: - name: kubernetes-admin@kubernetes context: cluster: kubernetes user: admin current-context: kubernetes-admin@kubernetes

pod DNS解析流程 ☆

1. 解析流程：
    - Pod -> CoreDNS
    - CoreDNS查找本地缓存
    - 未命中则查询kubernetes API
    - 返回解析结果
2. DNS策略（dnsPolicy）：
    - ClusterFirst：优先使用集群DNS

- Default：使用节点DNS
- None：自定义DNS
- ClusterFirstWithHostNet：hostNetwork下使用集群DNS

3. 常见格式：
   - 服务：service-name.namespace.svc.cluster.local
   - Pod：pod-ip.namespace.pod.cluster.local

k8s中的存储类型有哪些？PV和PVC的工作原理是什么？

1. 存储类型：
   - emptyDir：临时存储
   - hostPath：主机目录
   - configMap：配置文件
   - secret：加密数据
   - PersistentVolume：持久化存储
2. PV（PersistentVolume）：
   - 管理员创建的存储资源
   - 独立于Pod生命周期
   - 支持多种后端存储
3. PVC（PersistentVolumeClaim）：
   - 用户存储请求
   - 指定存储大小和访问模式
   - 自动绑定合适的PV
4. 工作流程：
   - 管理员创建StorageClass
   - 用户创建PVC
   - 系统根据StorageClass动态创建PV
   - PVC绑定到PV
   - Pod使用PVC

traefik对比nginx ingress优点

1. 动态配置：
   - 自动发现服务
   - 实时更新配置
   - 无需重启
2. 多协议支持：
   - HTTP/HTTPS
   - TCP/UDP
   - gRPC
   - WebSocket
3. 监控和可视化：
   - 内置Dashboard
   - Prometheus指标
   - 健康检查

4. 中间件功能：
    - 认证
    - 限流
    - 重试
    - 熔断

Harbor有哪些组件

1. 核心组件：
    - Core：核心API和认证
    - Registry：镜像存储
    - Database：元数据存储
    - Redis：会话管理
    - Job Service：任务处理
    - Log Collector：日志收集
2. 可选组件：
    - Notary：镜像签名
    - Clair：漏洞扫描
    - Chartmuseum：Helm仓库

Harbor高可用怎么实现

1. 存储高可用：
    - 共享存储（NFS/S3）
    - 数据库主从复制
    - Redis集群
2. 负载均衡：
    - nginx/haproxy前端代理
    - 多个Harbor节点
    - 会话保持
3. 组件冗余：
    - 多副本部署
    - 自动故障转移
    - 健康检查

ETCD调优

1. 性能优化：
    - 使用SSD存储
    - 调整内存配置
    - 优化网络延迟
2. 参数调整：

--quota-backend-bytes：空间配额 --auto-compaction-retention：压缩保留时间 --snapshot-count：快照触发阈值 --heartbeat-interval：心跳间隔 --election-timeout：选举超时

1. 监控指标：
    - 磁盘IOPS
    - 网络延迟
    - 内存使用
    - 请求延迟

如何实现k8s的滚动更新和回滚？

1. 滚动更新策略：

spec: strategy: type: RollingUpdate rollingUpdate: maxSurge: 25% maxUnavailable: 25%

1. 更新命令：

kubectl set image deployment/app container=image:v2 # 或 kubectl edit deployment app

1. 回滚操作：

查看历史 kubectl rollout history deployment/app # 回滚到上一版本 kubectl rollout undo deployment/app # 回滚到指定版本 kubectl rollout undo deployment/app --to-revision=2

1. 更新过程监控：

kubectl rollout status deployment/app

假设k8s集群规模上千，需要注意的问题有哪些？

1. 性能优化：
    - etcd性能调优
    - API Server水平扩展
    - 使用ipvs模式
    - 节点标签优化
2. 资源管理：
    - 合理设置资源限制
    - 使用节点亲和性
    - 优化调度策略
    - Pod打散部署
3. 监控告警：
    - 全面监控覆盖
    - 及时告警通知
    - 日志收集优化
    - 性能指标监控

4. 运维管理：
    - 自动化运维
    - 备份恢复策略
    - 升级策略
    - 故障自愈

节点NotReady可能的原因？会导致哪些问题？☆

可能原因：

1. 网络问题：
    - CNI插件故障
    - 网络配置错误
    - 网络连接中断
2. 系统问题：
    - 内存不足
    - 磁盘空间满
    - 系统负载过高
3. kubelet问题：
    - kubelet服务异常
    - 证书过期
    - 配置错误

导致的问题：

1. Pod调度：
    - 新Pod无法调度到该节点
    - 已有Pod可能被驱逐
2. 服务影响：
    - 服务可用性下降
    - 负载均衡异常
    - 集群容量减少

service和endpoints是如何关联的？

1. 关联机制：
    - Service通过selector选择Pod
    - 自动创建对应的Endpoints
    - 持续监控Pod变化更新Endpoints
2. 工作流程：

Service定义 apiVersion: v1 kind: Service spec: selector: app: myapp ports: - port: 80

1. Endpoints更新：
    - Pod创建时添加endpoint
    - Pod删除时移除endpoint
    - Pod IP变化时更新endpoint

2. 手动管理：
    - 可以不使用selector
    - 手动创建Endpoints
    - 用于外部服务接入

ReplicaSet、Deployment功能是怎么实现的？

1. ReplicaSet实现：
    - 监控Pod数量
    - 根据模板创建Pod
    - 保持期望副本数
    - 标签选择器匹配
2. Deployment实现：
    - 创建ReplicaSet
    - 管理更新策略
    - 控制版本记录
    - 支持回滚操作
3. 控制器模式：
    - 观察当前状态
    - 对比期望状态
    - 执行调整操作
    - 持续循环检查

scheduler调度流程

1. 预选（Predicates）：
    - 节点筛选
    - 资源检查
    - 亲和性规则
    - 污点容忍
2. 优选（Priorities）：
    - 节点评分
    - 负载均衡
    - 资源分布
    - 亲和性优先级
3. 选择节点：
    - 综合评分
    - 选择最优节点
    - 绑定Pod
4. 扩展点：
    - 自定义调度器
    - 调度插件
    - 优先级定制

HPA怎么实现的☆

1. 工作原理：
    - 定期获取指标数据
    - 计算期望副本数
    - 调整Deployment/RS副本数
    - 自动扩缩容
2. 支持的指标：
    - CPU使用率
    - 内存使用率
    - 自定义指标
    - 外部指标
3. 配置示例：

apiVersion: autoscaling/v2 kind: HorizontalPodAutoscaler spec: scaleTargetRef: apiVersion: apps/v1 kind: Deployment name: php-apache minReplicas: 1 maxReplicas: 10 metrics: - type: Resource resource: name: cpu target: type: Utilization averageUtilization: 50

request limit底层是怎么限制的☆

1. CPU限制：
    - requests：通过cpu.shares实现
    - limits：通过cpu.cfs_quota_us实现
    - 使用CFS调度器
2. 内存限制：
    - requests：最小保证量
    - limits：通过memory.limit_in_bytes实现
    - OOM Kill机制
3. 实现机制：
    - cgroups控制
    - kubelet执行
    - 容器运行时配合

k8s中的RBAC是如何实现的？

1. 基本概念：
    - Role：权限规则集合
    - RoleBinding：角色绑定
    - ClusterRole：集群级角色
    - ClusterRoleBinding：集群级绑定
2. 权限控制：
    - API资源级别
    - 命名空间级别
    - 操作动词控制

3. 配置示例：

apiVersion: rbac.authorization.k8s.io/v1 kind: Role metadata: namespace: default name: pod-reader rules: - apiGroups: [""] resources: ["pods"] verbs: ["get", "list", "watch"] --- apiVersion: rbac.authorization.k8s.io/v1 kind: RoleBinding metadata: name: read-pods namespace: default subjects: - kind: User name: jane apiGroup: rbac.authorization.k8s.io roleRef: kind: Role name: pod-reader apiGroup: rbac.authorization.k8s.io

helm工作原理是什么？

1. 核心概念：
    - Chart：应用包
    - Release：Chart的运行实例
    - Repository：Chart仓库
    - Values：配置值
2. 工作流程：
    - 读取Chart
    - 合并values
    - 模板渲染
    - 生成k8s资源
    - 应用到集群
3. 组件：
    - helm CLI：客户端工具
    - Chart Repository：存储仓库
    - Kubernetes API：部署目标

helm chart rollback实现过程是什么？

1. 回滚流程：
    - 获取历史版本信息
    - 读取历史配置
    - 重新渲染模板
    - 应用新配置
2. 命令操作：

查看历史版本 helm history release-name # 回滚操作 helm rollback release-name revision-number # 查看回滚状态 helm status release-name

1. 注意事项：
    - 保留历史版本数量配置
    - 数据持久化处理
    - 配置兼容性检查

velero备份与恢复流程是什么

1. 备份流程：
    - 创建备份任务

- 获取资源快照
- 备份etcd数据
- 存储到对象存储

2. 恢复流程：
    - 选择备份点
    - 恢复资源对象
    - 恢复持久化数据
    - 验证恢复结果
3. 配置示例：

apiVersion: velero.io/v1 kind: Backup metadata: name: backup-1 spec: includedNamespaces: - default storageLocation: default volumeSnapshotLocations: - default

docker网络模式

1. bridge模式（默认）：
    - 创建虚拟网桥
    - NAT转发
    - 容器间通信
2. host模式：
    - 共享主机网络
    - 直接使用主机端口
    - 性能最好
3. container模式：
    - 共享其他容器网络
    - 适用于监控容器
4. none模式：
    - 无网络配置
    - 完全隔离
    - 自定义网络

docker和container区别☆

1. 架构差异：
    - Docker：完整容器解决方案
    - Container：OCI规范实现
2. 功能对比： Docker：
    - 镜像构建
    - 容器运行时
    - 网络管理
    - 存储管理

Container：

- 容器运行时
- OCI兼容
- 轻量级

3. 使用场景： Docker：

- 开发环境
- 单机部署
- 完整生态

Container：

- 云原生环境
- 编排系统
- 轻量部署

如何排查k8s集群中的网络问题？

1. 排查步骤：

- 检查Pod网络状态
- 验证DNS解析
- 检查网络策略
- 排查CNI插件

2. 常用命令：

检查Pod网络 kubectl exec -it <pod> -- ping <target> kubectl exec -it <pod> -- nslookup <service> kubectl exec -it <pod> -- curl <endpoint> # 检查网络策略 kubectl get networkpolicy kubectl describe networkpolicy <name> # 检查CNI kubectl get pods -n kube-system | grep calico

1. 常见问题：
   - DNS解析失败
   - Service访问异常
   - 跨节点通信问题
   - 网络策略配置错误

如何减小dockerfile生成镜像体积？

1. 基础优化：

- 使用轻量级基础镜像
- 多阶段构建
- 合并RUN命令
- 清理缓存文件

2. 最佳实践：

多阶段构建示例 FROM golang:1.17 as builder WORKDIR /app COPY . . RUN go build -o main FROM alpine:3.14 COPY --from=builder /app/main / CMD ["/main"]

1. 其他技巧：
   - 使用.dockerignore
   - 选择合适的基础镜像

- 删除不必要的依赖
- 压缩静态文件

k8s日志采集方案

1. 常用方案：
    - DaemonSet部署采集器
    - Sidecar容器
    - Node级别采集
2. 流行工具：
    - EFK/ELK Stack
    - Loki
    - Filebeat
    - Fluentd
3. 架构设计：
    - 日志源
    - 采集器
    - 传输层
    - 存储层
    - 展示层

Pause容器的用途☆

1. 主要功能：
    - 作为Pod网络命名空间的基础
    - 启用Pod内容器共享网络
    - 维持Pod网络生命周期
2. 技术实现：
    - 创建网络命名空间
    - 分配IP地址
    - 路由设置
    - 端口映射
3. 重要性：
    - Pod网络基础设施
    - 容器通信基础
    - 保持Pod存活

k8s证书过期怎么更新

1. 检查证书：

kubeadm certs check-expiration # 更新所有证书 kubeadm certs renew all # 更新特定证书 kubeadm certs renew apiserver kubeadm certs renew apiserver-kubelet-client

1. 手动更新步骤：
    - 备份现有证书
    - 生成新证书

- 替换证书文件
- 重启相关组件

2. 注意事项：

- 更新时间窗口选择
- 集群组件重启顺序
- 证书备份保存
- 多master节点同步

K8S QoS等级☆

1. Guaranteed：

- requests = limits
- CPU和内存都必须设置
- 最高优先级
- 不会被OOM Kill

2. Burstable：

- requests < limits
- 至少设置一个资源的requests
- 中等优先级
- 可能被OOM Kill

3. BestEffort：

- 未设置requests和limits
- 最低优先级
- 最先被OOM Kill

k8s节点维护注意事项

1. 前置准备：

- 检查节点状态
- 备份重要数据
- 评估影响范围
- 制定回滚方案

2. 维护步骤：

标记节点不可调度 kubectl cordon <node> # 驱逐节点上的Pod kubectl drain <node> --ignore-daemonsets # 维护完成后恢复调度 kubectl uncordon <node>

1. 注意事项：
   - Pod驱逐策略
   - 存储卷处理
   - 服务可用性
   - 监控告警调整

Headless Service和ClusterIP区别☆

1. Headless Service：

- 不分配ClusterIP
- 直接返回Pod IP

    - DNS解析到所有Pod
    - 适用于有状态应用

2. ClusterIP Service：
    - 分配虚拟IP
    - 提供负载均衡
    - DNS解析到ClusterIP
    - 适用于无状态应用

3. 配置区别：

Headless Service spec: clusterIP: None # ClusterIP Service spec: type: ClusterIP

Linux容器技术的基础原理

1. 核心技术：
    - Namespace：资源隔离
        - PID namespace：进程隔离
        - Network namespace：网络隔离
        - Mount namespace：文件系统隔离
        - UTS namespace：主机名隔离
        - IPC namespace：进程间通信隔离
        - User namespace：用户隔离
    - Cgroups：资源限制
        - CPU限制
        - 内存限制
        - IO限制
        - 网络带宽限制
2. 安全机制：
    - SELinux/AppArmor
    - Capabilities
    - Seccomp
3. 存储机制：
    - 联合文件系统（UnionFS）
    - 写时复制（Copy-on-Write）
    - 数据卷（Volume）

Kubernetes Pod的常见调度方式

1. 自动调度：
    - 默认调度器
    - 资源需求
    - 节点选择器
    - 亲和性规则
2. 手动调度：

spec: nodeName: node1 # 直接指定节点

1. 高级调度：
    - Pod亲和性/反亲和性
    - 节点亲和性
    - 污点和容忍
    - Pod拓扑分布约束

kubernetes Ingress原理☆

1. 工作原理：
    - 监听Ingress资源变化
    - 生成负载均衡配置
    - 转发外部流量到Service
    - 提供七层负载均衡
2. 主要功能：
    - 路径转发
    - TLS终止
    - 名称虚拟主机
    - 流量控制
3. 配置示例：

apiVersion: networking.k8s.io/v1 kind: Ingress metadata: name: example-ingress spec: rules: - host: example.com http: paths: - path: / pathType: Prefix backend: service: name: web port: number: 80

Kubernetes各模块如何与API Server通信

1. 组件通信方式：
    - REST API
    - Watch机制
    - 证书认证
    - webhook
2. 通信流程：
    - 组件启动时配置API Server地址
    - 建立TLS安全连接
    - 通过API进行操作
    - Watch资源变化
3. 认证方式：
    - 客户端证书
    - Bearer Token
    - ServiceAccount
    - Webhook Token

kubelet监控worker节点如何实现

1. 监控内容：
    - 节点状态
    - 容器健康
    - 资源使用
    - 存储状态
2. 实现机制：
    - cAdvisor采集容器指标
    - 定期健康检查
    - 资源使用统计
    - 状态报告给API Server
3. 监控方式：
    - 主动健康检查
    - 被动状态收集
    - 事件上报
    - 指标采集

容器时区不一致如何解决？

1. 挂载主机时区文件：

volumeMounts: - name: tz-config mountPath: /etc/localtime volumes: - name: tz-config hostPath: path: /etc/localtime

1. 设置环境变量：

env: - name: TZ value: Asia/Shanghai

1. 构建镜像时设置：

FROM ubuntu RUN ln -sf /usr/share/zoneinfo/Asia/Shanghai /etc/localtime