### **1、简述ETCD及其特点?**

etcd是一个用于配置共享和服务发现的键值存储系统，能够为整个分布式集群存储关键数据，协助集群正常运转 服务端将配置信息存储在etcd中，客户端从etcd中得到配置信息，etcd监听配置信息的变化，发现配置变化通知到客户端 特点 - 安装、使用简单 - 数据分层存储在目录中，类似于文件系统 - watch机制 - 安装机制：支持ssl证书认证 - 高性能：etc支持2k/s的读操作 - 一致可靠：基于Raft共识算法实现数据存储、服务调用的一致性和高可用性 - Revision机制：每个key带有一个revision号，每次事物便加一。 -

### **2、简述ETCD适应的场景?**

* 服务发现
* 消息发布与订阅
* 负载均衡
* 分布式通知与协调
* 分布式锁
* 集群监控与leader选举

### **3、简述什么是Kubernetes?**

k8s是一个开源的容器管理工具，负责容器部署，容器扩缩容以及负载平衡。可以说k8s是一个多容器管理解决方案。

### **4、简述Kubernetes和Docker的关系?**

docker提供容器的生命周期管理、镜像构建运行时容器。k8s关联和编排容器在多个主机上互相通信。

### **5、简述Kubernetes中什么是Minikube、Kubectl、Kubelet?**

* 是一种可以在本地轻松运行k8s的工具。
* 是一个命令行工具，可以使用该工具控制Kubernetes集群管理器，如检查群集资源，创建、删除和更新组件，查看应用程序
* 是一个代理服务，它在每个节点上运行，并使从服务器与主服务器通信

### **6、简述Kubernetes常见的部署方式?**

* 二进制部署
* kubeadm
* 源码安装部署

### **7、简述Kubernetes如何实现集群管理?**

在集群管理方面，Kubernetes将集群中的机器划分为一个Master节点和一群工作节点Node。其中，在Master节点运行着集群管理相关的一组进程kube-apiserver、kube-controller-manager和kube-scheduler，这些进程实现了整个集群的资源管理、Pod调度、弹性伸缩、安全控制、系统监控和纠错等管理能力，并且都是全自动完成的

### **8、简述Kubernetes的优势、适应场景及其特点?**

优势 - 容器编排 - 轻量级 - 开源 - 弹性伸缩 - 负载均衡

场景 - 快速部署 - 快速扩展 - 快速对接新的应用功能 - 优化资源，提升资源利用率

特点 - 可移植 - 可扩展 - 自动化

### **9、简述Kubernetes的缺点或当前的不足之处?**

* 安装过程和配置相对困难复杂
* 管理服务相对繁琐
* 运行和编译需要很多时间
* 它比其他替代品更昂贵

### **10、简述Kubernetes相关基础概念**

|  |  |
| --- | --- |
| 概念 | 描述 |
| Pod（Pod组） | Pod是Kubernetes的最小调度单元，它可以包含一个或多个容器。容器在同一个Pod中共享网络和存储，它们之间可以通过localhost通信。 |
| Node（节点） | 节点是Kubernetes集群中的一个工作机器，可以是物理机器或虚拟机。每个节点负责运行Pod中的容器，并由Master节点进行管理。 |
| Cluster（集群） | 集群是由一组工作节点和一个Master节点组成的Kubernetes系统。Master节点负责整个集群的控制和管理。 |
| Service（服务） | 服务是一种抽象，定义了一组Pod及其访问方式。它可以确保Pod的稳定网络标识和动态路由。 |
| ReplicaSet | ReplicaSet是用于确保在集群中运行指定数量的Pod的控制器。当Pod数目发生变化时，ReplicaSet会启动或终止Pod，以维持所需的数量。 |
| Deployment（部署） | 部署是一种资源对象，它描述了应用程序的期望状态，并确保集群中的Pod数量符合这个状态。 |
| Namespace（命名空间） | 命名空间用于将集群划分为多个虚拟集群，每个命名空间中的资源相互隔离。 |
| ConfigMap和Secret | ConfigMap用于将配置数据提供给应用程序，而Secret用于存储敏感信息，如密码和API密钥。 |
| Service Discovery（服务发现） | Kubernetes通过服务发现机制允许容器应用程序找到和通信其他应用程序的服务。 |
| Ingress | Ingress是一种API对象，定义了从集群外部到集群内部服务的规则。 |
| Persistent Volumes和Persistent Volume Claims | Persistent Volumes（PV）提供了一种抽象，用于将存储资源与集群中的Pod分离开来。Persistent Volume Claims（PVC）是对Persistent Volumes的请求，用于绑定Pod中的存储资源。 |
| RBAC（Role-Based Access Control） | RBAC用于定义对Kubernetes资源的访问权限，以及哪些用户或服务账户可以执行哪些操作。 |

### **11、简述Kubernetes集群相关组件?**

* master
  + kube-controller-manager
  + kube-apiserver
  + kube-scheduler
  + etcd
* worker
  + kubelet
  + kube-proxy

### **12、简述Kubernetes RC的机制?**

Replication Controller用来管理Pod的副本，保证集群中存在指定数量的Pod副本。当定义了RC并提交至Kubernetes集群中之后，Master节点上的Controller Manager组件获悉，并同时巡检系统中当前存活的目标Pod，并确保目标Pod实例的数量刚好等于此RC的期望值，若存在过多的Pod副本在运行，系统会停止一些Pod，反之则自动创建一些Pod

### **13、简述kube-proxy作用?**

kube-proxy的作用主要是负责service的实现,具体来说,就是实现了内部从pod到service和外部的从node port向service的访问

### **14、简述kube-proxy iptables原理?**

Kubernetes从1.2版本开始，将iptables作为kube-proxy的默认模式。iptables模式下的kube-proxy不再起到Proxy的作用，其核心功能：通过API Server的Watch接口实时跟踪Service与Endpoint的变更信息，并更新对应的iptables规则，Client的请求流量则通过iptables的NAT机制“直接路由”到目标Pod

### **15、简述kube-proxy ipvs原理?**

答：IPVS 在 Kubernetes1.11 中升级为 GA 稳定版。IPVS 则专门用于高性能负载均衡，并使用更高效的数据结构（Hash 表），允许几乎无限的规模扩张，因此被 kube-proxy 采纳为最新模式。

在 IPVS 模式下，使用 iptables 的扩展 ipset，而不是直接调用 iptables 来生成规则链。iptables 规则链是一个线性的数据结构，ipset 则引入了带索引的数据结构，因此当规则很多时，也可以很高效地查找和匹配。

可以将 ipset 简单理解为一个 IP（段）的集合，这个集合的内容可以是 IP 地址、IP 网段、端口等，iptables 可以直接添加规则对这个“可变的集合”进行操作，这样做的好处在于可以大大减少 iptables 规则的数量，从而减少性能损耗。

### **16、简述kube-proxy ipvs和iptables的异同?**

答：iptables 与 IPVS 都是基于 Netfilter 实现的，但因为定位不同，二者有着本质的

差别：iptables 是为防火墙而设计的；IPVS 则专门用于高性能负载均衡，并使用更高效的数据结构（Hash 表），允许几乎无限的规模扩张。

与 iptables 相比，IPVS 拥有以下明显优势：

1、为大型集群提供了更好的可扩展性和性能；

2、支持比 iptables 更复杂的负载均衡算法（最小负载、最少连接、加权等）；

3、支持服务器健康检查和连接重试等功能；

4、可以动态修改 ipset 的集合，即使 iptables 的规则正在使用这个集合。

### **17、简述Kubernetes中什么是静态Pod?**

答：静态 pod 是由 kubelet 进行管理的仅存在于特定 Node的Pod，他们不能通过 API Server 进行管理，无法与 ReplicationController、Deployment 或者DaemonSet 进行关联，并且 kubelet 无法对他们进行健康检查。静态 Pod 总是由kubelet 进行创建，并且总是在 kubelet 所在的 Node 上运行

### **18、简述Kubernetes中Pod可能位于的状态?**

Pending API Server已经创建该Pod，且Pod内还有一个或多个容器的镜像没有创建，包括正在下载镜像的过程。 Running Pod内所有容器均已创建，且至少有一个容器处于运行状态、正在启动状态或正在重启状态。 Succeeded Pod内所有容器均成功执行退出，且不会重启。 Failed Pod内所有容器均已退出，但至少有一个容器退出为失败状态。 Unknown 由于某种原因无法获取该Pod状态，可能由于网络通信不畅导致。

### **19、简述Kubernetes创建一个Pod的主要流程?**

Kubernetes中创建一个Pod涉及多个组件之间联动，主要流程如下：

* 客户端提交Pod的配置信息（可以是yaml文件定义的信息）到kube-apiserver。
* Apiserver收到指令后，通知给controller-manager创建一个资源对象。
* Controller-manager通过api-server将pod的配置信息存储到ETCD数据中心中。
* Kube-scheduler检测到pod信息会开始调度预选，会先过滤掉不符合Pod资源配置要求的节点，然后开始调度调优，主要是挑选出更适合运行pod的节点，然后将pod的资源配置单发送到node节点上的kubelet组件上。
* Kubelet根据scheduler发来的资源配置单运行pod，运行成功后，将pod的运行信息返回给scheduler，scheduler将返回的pod运行状况的信息存储到etcd数据中心。

### **20、简述Kubernetes中Pod的重启策略?**

Pod的重启策略包括Always、OnFailure和Never，默认值为Always。

Always：当容器失效时，由kubelet自动重启该容器； OnFailure：当容器终止运行且退出码不为0时，由kubelet自动重启该容器； Never：不论容器运行状态如何，kubelet都不会重启该容器。 同时Pod的重启策略与控制方式关联，当前可用于管理Pod的控制器包括ReplicationController、Job、DaemonSet及直接管理kubelet管理（静态Pod）。 不同控制器的重启策略限制如下： RC和DaemonSet：必须设置为Always，需要保证该容器持续运行； Job：OnFailure或Never，确保容器执行完成后不再重启； kubelet：在Pod失效时重启，不论将RestartPolicy设置为何值，也不会对Pod进行健康检查。

### **21、简述Kubernetes中Pod的健康检查方式?**

对Pod的健康检查可以通过两类探针来检查：LivenessProbe和ReadinessProbe。

LivenessProbe探针：用于判断容器是否存活（running状态），如果LivenessProbe探针探测到容器不健康，则kubelet将杀掉该容器，并根据容器的重启策略做相应处理。若一个容器不包含LivenessProbe探针，kubelet认为该容器的LivenessProbe探针返回值用于是“Success”。

ReadineeProbe探针：用于判断容器是否启动完成（ready状态）。如果ReadinessProbe探针探测到失败，则Pod的状态将被修改。Endpoint Controller将从Service的Endpoint中删除包含该容器所在Pod的Eenpoint。

startupProbe探针：启动检查机制，应用一些启动缓慢的业务，避免业务长时间启动而被上面两类探针kill掉。

### **22、简述Kubernetes Pod的LivenessProbe探针的常见方式?**

ExecAction：在容器内执行一个命令，若返回码为0，则表明容器健康。

TCPSocketAction：通过容器的IP地址和端口号执行TCP检查，若能建立TCP连接，则表明容器健康。

HTTPGetAction：通过容器的IP地址、端口号及路径调用HTTP Get方法，若响应的状态码大于等于200且小于400，则表明容器健康。

### **23、简述Kubernetes Pod的常见调度方式?**

Kubernetes中，Pod通常是容器的载体，主要有如下常见调度方式：

Deployment或RC：该调度策略主要功能就是自动部署一个容器应用的多份副本，以及持续监控副本的数量，在集群内始终维持用户指定的副本数量。

NodeSelector：定向调度，当需要手动指定将Pod调度到特定Node上，可以通过Node的标签（Label）和Pod的nodeSelector属性相匹配。

NodeAffinity亲和性调度：亲和性调度机制极大的扩展了Pod的调度能力，目前有两种节点亲和力表达：

requiredDuringSchedulingIgnoredDuringExecution：硬规则，必须满足指定的规则，调度器才可以调度Pod至Node上（类似nodeSelector，语法不同）。

preferredDuringSchedulingIgnoredDuringExecution：软规则，优先调度至满足的Node的节点，但不强求，多个优先级规则还可以设置权重值。

Taints和Tolerations（污点和容忍）：

Taint：使Node拒绝特定Pod运行；

Toleration：为Pod的属性，表示Pod能容忍（运行）标注了Taint的Node。

### **24、简述Kubernetes初始化容器（init container）?**

init container的运行方式与应用容器不同，它们必须先于应用容器执行完成，当设置了多个init container时，将按顺序逐个运行，并且只有前一个init container运行成功后才能运行后一个init container。当所有init container都成功运行后，Kubernetes才会初始化Pod的各种信息，并开始创建和运行应用容器。

### **25、简述Kubernetes deployment升级过程?**

初始创建Deployment时，系统创建了一个ReplicaSet，并按用户的需求创建了对应数量的Pod副本。

当更新Deployment时，系统创建了一个新的ReplicaSet，并将其副本数量扩展到1，然后将旧ReplicaSet缩减为2。

之后，系统继续按照相同的更新策略对新旧两个ReplicaSet进行逐个调整。

最后，新的ReplicaSet运行了对应个新版本Pod副本，旧的ReplicaSet副本数量则缩减为0。

### **26、简述Kubernetes deployment升级策略?**

在Deployment的定义中，可以通过spec.strategy指定Pod更新的策略，目前支持两种策略：Recreate（重建）和RollingUpdate（滚动更新），默认值为RollingUpdate。

Recreate：设置spec.strategy.type=Recreate，表示Deployment在更新Pod时，会先杀掉所有正在运行的Pod，然后创建新的Pod。

RollingUpdate：设置spec.strategy.type=RollingUpdate，表示Deployment会以滚动更新的方式来逐个更新Pod。同时，可以通过设置spec.strategy.rollingUpdate下的两个参数（maxUnavailable和maxSurge）来控制滚动更新的过程。

### **27、简述Kubernetes DaemonSet类型的资源特性?**

DaemonSet资源对象会在每个Kubernetes集群中的节点上运行，并且每个节点只能运行一个pod，这是它和deployment资源对象的最大也是唯一的区别。因此，在定义yaml文件中，不支持定义replicas。

它的一般使用场景如下：

在去做每个节点的日志收集工作。

监控每个节点的的运行状态。

### **28、简述Kubernetes自动扩容机制?**

Kubernetes使用Horizontal Pod Autoscaler（HPA）的控制器实现基于CPU使用率进行自动Pod扩缩容的功能。HPA控制器周期性地监测目标Pod的资源性能指标，并与HPA资源对象中的扩缩容条件进行对比，在满足条件时对Pod副本数量进行调整。

HPA原理

Kubernetes中的某个Metrics Server（Heapster或自定义Metrics Server）持续采集所有Pod副本的指标数据。HPA控制器通过Metrics Server的API（Heapster的API或聚合API）获取这些数据，基于用户定义的扩缩容规则进行计算，得到目标Pod副本数量。

当目标Pod副本数量与当前副本数量不同时，HPA控制器就向Pod的副本控制器（Deployment、RC或ReplicaSet）发起scale操作，调整Pod的副本数量，完成扩缩容操作。

### **29、简述Kubernetes Service类型?**

通过创建Service，可以为一组具有相同功能的容器应用提供一个统一的入口地址，并且将请求负载分发到后端的各个容器应用上。其主要类型有：

ClusterIP：虚拟的服务IP地址，该地址用于Kubernetes集群内部的Pod访问，在Node上kube-proxy通过设置的iptables规则进行转发；

NodePort：使用宿主机的端口，使能够访问各Node的外部客户端通过Node的IP地址和端口号就能访问服务；

LoadBalancer：使用外接负载均衡器完成到服务的负载分发，需要在spec.status.loadBalancer字段指定外部负载均衡器的IP地址，通常用于公有云。

### **30、简述Kubernetes Service分发后端的策略?**

Service负载分发的策略有：RoundRobin和SessionAffinity

RoundRobin：默认为轮询模式，即轮询将请求转发到后端的各个Pod上。

SessionAffinity：基于客户端IP地址进行会话保持的模式，即第1次将某个客户端发起的请求转发到后端的某个Pod上，之后从相同的客户端发起的请求都将被转发到后端相同的Pod上。

### **31、简述Kubernetes Headless Service?**

在某些应用场景中，若需要人为指定负载均衡器，不使用Service提供的默认负载均衡的功能，或者应用程序希望知道属于同组服务的其他实例。Kubernetes提供了Headless Service来实现这种功能，即不为Service设置ClusterIP（入口IP地址），仅通过Label Selector将后端的Pod列表返回给调用的客户端。

### **32、简述Kubernetes外部如何访问集群内的服务?**

对于Kubernetes，集群外的客户端默认情况，无法通过Pod的IP地址或者Service的虚拟IP地址:虚拟端口号进行访问。通常可以通过以下方式进行访问Kubernetes集群内的服务：

映射Pod到物理机：将Pod端口号映射到宿主机，即在Pod中采用hostPort方式，以使客户端应用能够通过物理机访问容器应用。

映射Service到物理机：将Service端口号映射到宿主机，即在Service中采用nodePort方式，以使客户端应用能够通过物理机访问容器应用。

映射Sercie到LoadBalancer：通过设置LoadBalancer映射到云服务商提供的LoadBalancer地址。这种用法仅用于在公有云服务提供商的云平台上设置Service的场景。

### **33、简述Kubernetes ingress?**

Kubernetes的Ingress资源对象，用于将不同URL的访问请求转发到后端不同的Service，以实现HTTP层的业务路由机制。

Kubernetes使用了Ingress策略和Ingress Controller，两者结合并实现了一个完整的Ingress负载均衡器。使用Ingress进行负载分发时，Ingress Controller基于Ingress规则将客户端请求直接转发到Service对应的后端Endpoint（Pod）上，从而跳过kube-proxy的转发功能，kube-proxy不再起作用，全过程为：ingress controller + ingress 规则 ----> services。

同时当Ingress Controller提供的是对外服务，则实际上实现的是边缘路由器的功能。

### **34、简述Kubernetes镜像的下载策略?**

K8s的镜像下载策略有三种：Always、Never、IFNotPresent。

Always：镜像标签为latest时，总是从指定的仓库中获取镜像。

Never：禁止从仓库中下载镜像，也就是说只能使用本地镜像。

IfNotPresent：仅当本地没有对应镜像时，才从目标仓库中下载。默认的镜像下载策略是：当镜像标签是latest时，默认策略是Always；当镜像标签是自定义时（也就是标签不是latest），那么默认策略是IfNotPresent。

### **35、简述Kubernetes的负载均衡器?**

负载均衡器是暴露服务的最常见和标准方式之一。

根据工作环境使用两种类型的负载均衡器，即内部负载均衡器或外部负载均衡器。内部负载均衡器自动平衡负载并使用所需配置分配容器，而外部负载均衡器将流量从外部负载引导至后端容器。负载均衡器是暴露服务的最常见和标准方式之一。

根据工作环境使用两种类型的负载均衡器，即内部负载均衡器或外部负载均衡器。内部负载均衡器自动平衡负载并使用所需配置分配容器，而外部负载均衡器将流量从外部负载引导至后端容器。

### **36、简述Kubernetes各模块如何与API Server通信?**

Kubernetes API Server作为集群的核心，负责集群各功能模块之间的通信。集群内的各个功能模块通过API Server将信息存入etcd，当需要获取和操作这些数据时，则通过API Server提供的REST接口（用GET、LIST或WATCH方法）来实现，从而实现各模块之间的信息交互。

如kubelet进程与API Server的交互：每个Node上的kubelet每隔一个时间周期，就会调用一次API Server的REST接口报告自身状态，API Server在接收到这些信息后，会将节点状态信息更新到etcd中。

如kube-controller-manager进程与API Server的交互：kube-controller-manager中的Node Controller模块通过API Server提供的Watch接口实时监控Node的信息，并做相应处理。

如kube-scheduler进程与API Server的交互：Scheduler通过API Server的Watch接口监听到新建Pod副本的信息后，会检索所有符合该Pod要求的Node列表，开始执行Pod调度逻辑，在调度成功后将Pod绑定到目标节点上。

### **37、简述Kubernetes Scheduler作用及实现原理?**

Kubernetes Scheduler是负责Pod调度的重要功能模块，Kubernetes Scheduler在整个系统中承担了“承上启下”的重要功能，“承上”是指它负责接收Controller Manager创建的新Pod，为其调度至目标Node；“启下”是指调度完成后，目标Node上的kubelet服务进程接管后继工作，负责Pod接下来生命周期。

Kubernetes Scheduler的作用是将待调度的Pod（API新创建的Pod、Controller Manager为补足副本而创建的Pod等）按照特定的调度算法和调度策略绑定（Binding）到集群中某个合适的Node上，并将绑定信息写入etcd中。

在整个调度过程中涉及三个对象，分别是待调度Pod列表、可用Node列表，以及调度算法和策略。

Kubernetes Scheduler通过调度算法调度为待调度Pod列表中的每个Pod从Node列表中选择一个最适合的Node来实现Pod的调度。随后，目标节点上的kubelet通过API Server监听到Kubernetes Scheduler产生的Pod绑定事件，然后获取对应的Pod清单，下载Image镜像并启动容器。

### **38、简述Kubernetes Scheduler使用哪两种算法将Pod绑定到worker节点?**

预选（Predicates）：输入是所有节点，输出是满足预选条件的节点。kube-scheduler根据预选策略过滤掉不满足策略的Nodes。如果某节点的资源不足或者不满足预选策略的条件则无法通过预选。如“Node的label必须与Pod的Selector一致”。

优选（Priorities）：输入是预选阶段筛选出的节点，优选会根据优先策略为通过预选的Nodes进行打分排名，选择得分最高的Node。例如，资源越富裕、负载越小的Node可能具有越高的排名。

### **39、简述Kubernetes kubelet的作用?**

在Kubernetes集群中，在每个Node（又称Worker）上都会启动一个kubelet服务进程。该进程用于处理Master下发到本节点的任务，管理Pod及Pod中的容器。每个kubelet进程都会在API Server上注册节点自身的信息，定期向Master汇报节点资源的使用情况，并通过cAdvisor监控容器和节点资源。

### **40、简述Kubernetes kubelet监控Worker节点资源是使用什么组件来实现的?**

kubelet使用cAdvisor对worker节点资源进行监控。在 Kubernetes 系统中，cAdvisor 已被默认集成到 kubelet 组件内，当 kubelet 服务启动时，它会自动启动 cAdvisor 服务，然后 cAdvisor 会实时采集所在节点的性能指标及在节点上运行的容器的性能指标。

### **41、简述Kubernetes如何保证集群的安全性?**

* 最小权限原则
* 用户权限：划分普通用户和管理员的角色
* API Server的认证授权
* API Server的授权管理
* 敏感数据引入Secret机制
* AdmissionControl（准入机制）

### **42、简述Kubernetes准入机制?**

|  |  |
| --- | --- |
| Kubernetes准入机制 | 描述 |
| 准入控制插件 |  |
| - AlwaysAdmit | 总是通过请求，不执行检查。 |
| - AlwaysDeny | 总是拒绝请求，不执行检查。 |
| - NamespaceLifecycle | 确保命名空间存在，并根据策略允许或拒绝创建。 |
| - ResourceQuota | 实施资源配额，限制在命名空间内的资源使用。 |
| - PodSecurityPolicy | 强制执行容器安全策略。 |
| - ServiceAccount | 确保Pod使用有效的ServiceAccount。 |
| Webhook准入控制 |  |
| - Webhook机制 | 将自定义的准入控制插件集成到API服务器中。 |
| 动态准入控制配置 |  |
| - 动态配置 | 在Kubernetes 1.9版本及以上，引入了动态准入控制配置，管理员可以动态配置准入控制插件。 |
| Mutating和Validating准入控制 |  |
| - MutatingAdmissionWebhook | 在资源被持久化之前对其进行修改。例如，可以动态地注入sidecar容器。 |
| - ValidatingAdmissionWebhook | 对资源进行验证，确保其符合特定规则。例如，可以强制执行命名约定或其他策略。 |
| 默认准入控制配置 |  |
| - 启用/禁用默认插件 | 集群管理员可以通过启用或禁用默认的准入控制插件来调整系统的默认行为。 |

### **43、简述Kubernetes RBAC及其特点（优势）?**

* 对集群中的资源和非资源权限均有完整的覆盖。
* 整个RBAC完全由几个API对象完成， 同其他API对象一样， 可以用kubectl或API进行操作。
* 可以在运行时进行调整，无须重新启动API Server。

### **44、简述Kubernetes Secret作用?**

Secret对象，主要作用是保管私密数据，比如密码、OAuth Tokens、SSH Keys等信息。将这些私密信息放在Secret对象中比直接放在Pod或Docker Image中更安全，也更便于使用和分发。

### **45、简述Kubernetes Secret有哪些使用方式?**

在创建Pod时，通过为Pod指定Service Account来自动使用该Secret。

通过挂载该Secret到Pod来使用它。

在Docker镜像下载时使用，通过指定Pod的spc.ImagePullSecrets来引用它。

### **46、简述Kubernetes PodSecurityPolicy机制?**

Kubernetes PodSecurityPolicy是为了更精细地控制Pod对资源的使用方式以及提升安全策略。在开启PodSecurityPolicy准入控制器后，Kubernetes默认不允许创建任何Pod，需要创建PodSecurityPolicy策略和相应的RBAC授权策略（Authorizing Policies），Pod才能创建成功。

### **47、简述Kubernetes PodSecurityPolicy机制能实现哪些安全策略?**

* 特权模式：privileged是否允许Pod以特权模式运行。
* 宿主机资源：控制Pod对宿主机资源的控制，如hostPID：是否允许Pod共享宿主机的进程空间。
* 用户和组：设置运行容器的用户ID（范围）或组（范围）。
* 提升权限：AllowPrivilegeEscalation：设置容器内的子进程是否可以提升权限，通常在设置非root用户（MustRunAsNonRoot）时进行设置。
* SELinux：进行SELinux的相关配置。

### **48、简述Kubernetes网络模型?**

Kubernetes网络模型中每个Pod都拥有一个独立的IP地址，并假定所有Pod都在一个可以直接连通的、扁平的网络空间中。所以不管它们是否运行在同一个Node（宿主机）中，都要求它们可以直接通过对方的IP进行访问。设计这个原则的原因是，用户不需要额外考虑如何建立Pod之间的连接，也不需要考虑如何将容器端口映射到主机端口等问题。

同时为每个Pod都设置一个IP地址的模型使得同一个Pod内的不同容器会共享同一个网络命名空间，也就是同一个Linux网络协议栈。这就意味着同一个Pod内的容器可以通过localhost来连接对方的端口。

在Kubernetes的集群里，IP是以Pod为单位进行分配的。一个Pod内部的所有容器共享一个网络堆栈（相当于一个网络命名空间，它们的IP地址、网络设备、配置等都是共享的）。

### **49、简述Kubernetes CNI模型?**

CNI提供了一种应用容器的插件化网络解决方案，定义对容器网络进行操作和配置的规范，通过插件的形式对CNI接口进行实现。CNI仅关注在创建容器时分配网络资源，和在销毁容器时删除网络资源。在CNI模型中只涉及两个概念：容器和网络。

容器（Container）：是拥有独立Linux网络命名空间的环境，例如使用Docker或rkt创建的容器。容器需要拥有自己的Linux网络命名空间，这是加入网络的必要条件。

网络（Network）：表示可以互连的一组实体，这些实体拥有各自独立、唯一的IP地址，可以是容器、物理机或者其他网络设备（比如路由器）等。

对容器网络的设置和操作都通过插件（Plugin）进行具体实现，CNI插件包括两种类型：CNI Plugin和IPAM（IP Address Management）Plugin。CNI Plugin负责为容器配置网络资源，IPAM Plugin负责对容器的IP地址进行分配和管理。IPAM Plugin作为CNI Plugin的一部分，与CNI Plugin协同工作。

### **50、简述Kubernetes网络策略?**

Network Policy的主要功能是对Pod间的网络通信进行限制和准入控制，设置允许访问或禁止访问的客户端Pod列表。Network Policy定义网络策略，配合策略控制器（Policy Controller）进行策略的实现。

### **51、简述Kubernetes网络策略原理?**

Network Policy的工作原理主要为：policy controller需要实现一个API Listener，监听用户设置的Network Policy定义，并将网络访问规则通过各Node的Agent进行实际设置（Agent则需要通过CNI网络插件实现）。

### **52、简述Kubernetes中flannel的作用?**

它能协助Kubernetes，给每一个Node上的Docker容器都分配互相不冲突的IP地址。

它能在这些IP地址之间建立一个覆盖网络（Overlay Network），通过这个覆盖网络，将数据包原封不动地传递到目标容器内。

### **53、简述Kubernetes Calico网络组件实现原理?**

Calico是一个基于BGP的纯三层的网络方案 Calico在每个计算节点都利用Linux Kernel实现了一个高效的vRouter来负责数据转发。每个vRouter都通过BGP协议把在本节点上运行的容器的路由信息向整个Calico网络广播，并自动设置到达其他节点的路由转发规则。

Calico保证所有容器之间的数据流量都是通过IP路由的方式完成互联互通的。Calico节点组网时可以直接利用数据中心的网络结构（L2或者L3），不需要额外的NAT、隧道或者Overlay Network，没有额外的封包解包，能够节约CPU运算，提高网络效率。

### **54、简述Kubernetes共享存储的作用?**

Kubernetes对于有状态的容器应用或者对数据需要持久化的应用，因此需要更加可靠的存储来保存应用产生的重要数据，以便容器应用在重建之后仍然可以使用之前的数据。因此需要使用共享存储。

### **55、简述Kubernetes数据持久化的方式有哪些?**

EmptyDir（空目录）：没有指定要挂载宿主机上的某个目录，直接由Pod内保部映射到宿主机上。类似于docker中的manager volume。只需要临时将数据保存在磁盘上，比如在合并/排序算法中；作为两个容器的共享存储。同个pod里面的不同容器，共享同一个持久化目录，当pod节点删除时，volume的数据也会被删除。 Hostpath：将宿主机上已存在的目录或文件挂载到容器内部。类似于docker中的bind mount挂载方式 PersistentVolume（简称PV）：如基于NFS服务的PV，也可以基于GFS的PV。它的作用是统一数据持久化目录，方便管理。

### **56、简述Kubernetes PV和PVC?**

PV是对底层网络共享存储的抽象，将共享存储定义为一种“资源”。

PVC则是用户对存储资源的一个“申请”。

### **57、简述Kubernetes PV生命周期内的阶段?**

* Available：可用状态，还未与某个PVC绑定。
* Bound：已与某个PVC绑定。
* Released：绑定的PVC已经删除，资源已释放，但没有被集群回收。
* Failed：自动资源回收失败。

### **58、简述Kubernetes所支持的存储供应模式?**

Kubernetes支持两种资源的存储供应模式：静态模式（Static）和动态模式（Dynamic）。

静态模式：集群管理员手工创建许多PV，在定义PV时需要将后端存储的特性进行设置。

动态模式：集群管理员无须手工创建PV，而是通过StorageClass的设置对后端存储进行描述，标记为某种类型。此时要求PVC对存储的类型进行声明，系统将自动完成PV的创建及与PVC的绑定。

### **59、简述Kubernetes CSI模型?**

Kubernetes CSI是Kubernetes推出与容器对接的存储接口标准，存储提供方只需要基于标准接口进行存储插件的实现，就能使用Kubernetes的原生存储机制为容器提供存储服务。CSI使得存储提供方的代码能和Kubernetes代码彻底解耦，部署也与Kubernetes核心组件分离，显然，存储插件的开发由提供方自行维护，就能为Kubernetes用户提供更多的存储功能，也更加安全可靠。

CSI包括CSI Controller和CSI Node：

CSI Controller的主要功能是提供存储服务视角对存储资源和存储卷进行管理和操作。

CSI Node的主要功能是对主机（Node）上的Volume进行管理和操作。

### **60、简述Kubernetes Worker节点加入集群的过程?**

1、在该Node上安装Docker、kubelet和kube-proxy服务；

2、然后配置kubelet和kubeproxy的启动参数，将Master URL指定为当前Kubernetes集群Master的地址，最后启动这些服务；

3、通过kubelet默认的自动注册机制，新的Worker将会自动加入现有的Kubernetes集群中；

4、Kubernetes Master在接受了新Worker的注册之后，会自动将其纳入当前集群的调度范围。

### **61、简述Kubernetes Pod如何实现对节点的资源控制?**

Kubernetes集群里的节点提供的资源主要是计算资源，计算资源是可计量的能被申请、分配和使用的基础资源。当前Kubernetes集群中的计算资源主要包括CPU、GPU及Memory。CPU与Memory是被Pod使用的，因此在配置Pod时可以通过参数CPU Request及Memory Request为其中的每个容器指定所需使用的CPU与Memory量，Kubernetes会根据Request的值去查找有足够资源的Node来调度此Pod。

通常，一个程序所使用的CPU与Memory是一个动态的量，确切地说，是一个范围，跟它的负载密切相关：负载增加时，CPU和Memory的使用量也会增加。

### **62、简述Kubernetes Requests和Limits如何影响Pod的调度?**

当一个Pod创建成功时，Kubernetes调度器（Scheduler）会为该Pod选择一个节点来执行。对于每种计算资源（CPU和Memory）而言，每个节点都有一个能用于运行Pod的最大容量值。调度器在调度时，首先要确保调度后该节点上所有Pod的CPU和内存的Requests总和，不超过该节点能提供给Pod使用的CPU和Memory的最大容量值。

### **63、简述Kubernetes Metric Service?**

在Kubernetes从1.10版本后采用Metrics Server作为默认的性能数据采集和监控，主要用于提供核心指标（Core Metrics），包括Node、Pod的CPU和内存使用指标。

对其他自定义指标（Custom Metrics）的监控则由Prometheus等组件来完成。

### **64、简述Kubernetes中，如何使用EFK实现日志的统一管理**

在Kubernetes集群环境中，通常一个完整的应用或服务涉及组件过多，建议对日志系统进行集中化管理，通常采用EFK实现。

EFK是 Elasticsearch、Fluentd 和 Kibana 的组合，其各组件功能如下：

Elasticsearch：是一个搜索引擎，负责存储日志并提供查询接口；

Fluentd：负责从 Kubernetes 搜集日志，每个node节点上面的fluentd监控并收集该节点上面的系统日志，并将处理过后的日志信息发送给Elasticsearch；

Kibana：提供了一个 Web GUI，用户可以浏览和搜索存储在 Elasticsearch 中的日志。

通过在每台node上部署一个以DaemonSet方式运行的fluentd来收集每台node上的日志。Fluentd将docker日志目录/var/lib/docker/containers和/var/log目录挂载到Pod中，然后Pod会在node节点的/var/log/pods目录中创建新的目录，可以区别不同的容器日志输出，该目录下有一个日志文件链接到/var/lib/docker/contianers目录下的容器日志输出。

### **65、简述Kubernetes如何进行优雅的节点关机维护?**

由于Kubernetes节点运行大量Pod，因此在进行关机维护之前，建议先使用kubectl drain将该节点的Pod进行驱逐，然后进行关机维护。

### **66、简述Kubernetes集群联邦?**

Kubernetes集群联邦可以将多个Kubernetes集群作为一个集群进行管理。因此，可以在一个数据中心/云中创建多个Kubernetes集群，并使用集群联邦在一个地方控制/管理所有集群。

### **67、简述Helm及其优势?**

|  |  |
| --- | --- |
| Helm及其优势 | 描述 |
| 什么是Helm | Helm是一个用于简化Kubernetes应用程序部署、更新和管理的包管理工具。 |
| Chart | Helm使用Chart来定义、安装和升级Kubernetes应用程序。Chart是一个包含了所有部署Kubernetes应用所需信息的打包文件。 |
| 优势 |  |
| - 模板引擎 | Helm使用Go语言的模板引擎，允许用户动态生成Kubernetes资源配置，实现配置的可重用性和参数化。 |
| - 版本控制 | Helm允许用户版本控制Chart，轻松管理不同应用程序版本的发布和回滚。 |
| - 包管理 | 通过Helm，用户可以轻松共享和重用应用程序定义，避免重复劳动。 |
| - 依赖管理 | Helm支持定义Chart之间的依赖关系，简化了多组件应用程序的部署。 |
| - 社区支持 | Helm拥有活跃的社区，用户可以从社区中获取Chart，共享经验和解决问题。 |
| - 插件系统 | Helm提供插件系统，允许用户扩展其功能，满足特定需求。 |
| - 安全发布 | Helm允许用户将应用程序作为一个整体发布，确保一致性和可重复性。 |
| - Charts存储库 | 用户可以配置Charts存储库，从中获取和发布Charts，便于管理和分发。 |
| - 自定义配置值 | Helm允许通过values文件自定义Chart的配置，提高了可配置性和灵活性。 |

### **69、容器和主机部署应用的区别是什么?**

容器的中心思想就是秒级启动；一次封装、到处运行 容器部署可以将各个服务进行隔离，互不影响

### **70、请你说一下kubenetes针对pod资源对象的健康监测机制?**

* livenessProbe
* readinessProbe
* startupProbe

### **71、如何控制滚动更新过程?**

maxSurge：　此参数控制滚动更新过程，副本总数超过预期pod数量的上限。可以是百分比，也可以是具体的值。默认为1。

（上述参数的作用就是在更新过程中，值若为3，那么不管三七二一，先运行三个pod，用于替换旧的pod，以此类推）

maxUnavailable： 此参数控制滚动更新过程中，不可用的Pod的数量。

（这个值和上面的值没有任何关系，举个例子：我有十个pod，但是在更新的过程中，我允许这十个pod中最多有三个不可用，那么就将这个参数的值设置为3，在更新的过程中，只要不可用的pod数量小于或等于3，那么更新过程就不会停止）。

### **75、Service这种资源对象的作用是什么?**

用来给相同的多个pod对象提供一个固定的统一访问接口，常用于服务发现和服务访问。

### **76、版本回滚相关的命令?**

|  |  |
| --- | --- |
| 命令 | 描述 |
| kubectl rollout history deployment/<deployment-name> | 查看滚动升级历史记录 |
| kubectl rollout history deployment/<deployment-name> --revision=<revision-number> | 查看特定版本的详细信息 |
| kubectl rollout undo deployment/<deployment-name> | 回滚到上一个版本 |
| kubectl rollout undo deployment/<deployment-name> --to-revision=<revision-number> | 回滚到特定版本 |

### **77、标签与标签选择器的作用是什么?**

|  |  |
| --- | --- |
| 名词 | 描述 |
| 标签（Label） | 标签是键值对，用于附加元数据到Kubernetes资源对象（如Pod、Service、Node等）。 |
| 标签选择器（Label Selector） | 标签选择器用于选择带有特定标签的资源对象，以便进行筛选和操作。 |
| 作用 | 通过标签，可以对资源对象进行分类、组织和识别，提供更灵活的资源管理和调度方式。 |
|  | 标签选择器允许用户根据标签的键值对对资源进行查询和选择，用于实现有选择性的操作。 |
| 示例 | 一个Pod可以有标签app=web，允许使用标签选择器选择所有具有该标签的Pod进行管理。 |
|  | Service可以使用标签选择器将请求路由到具有特定标签的Pod，实现服务发现和负载均衡。 |

### **78、常用的标签分类有哪些?**

|  |  |
| --- | --- |
| 类别 | 描述 |
| 环境 | 用于区分不同环境，如environment=production。 |
| 应用程序 | 用于标识属于哪个应用程序，如app=web。 |
| 版本 | 用于标识应用程序或组件的版本，如version=1.2.3。 |
| 部门 | 用于区分不同部门的资源，如department=finance。 |
| 客户 | 用于标识属于哪个客户，如customer=companyA。 |
| 组织 | 用于标识属于哪个组织，如organization=dev-team。 |
| 状态 | 用于标识资源的状态，如status=ready。 |
| 系统 | 用于标识系统级别的资源，如system=true。 |

### **79、有几种查看标签的方式?**

|  |  |
| --- | --- |
| 方式 | 命令/操作 |
| 查看单个资源的标签 | kubectl get <资源类型> <资源名称> -o yaml |
| 查看所有资源的标签 | kubectl get <资源类型> --show-labels |
| 查看特定标签的资源 | kubectl get <资源类型> -l <标签选择器> |
| 查看所有命名空间中的标签 | kubectl get all --all-namespaces --show-labels |
| 使用kubectl describe查看资源详情 | kubectl describe <资源类型> <资源名称> |

### **80、添加、修改、删除标签的命令?**

|  |  |  |
| --- | --- | --- |
| 操作 | 命令 | 示例 |
| 添加标签 | kubectl label <资源类型> <资源名称> <键>=<值> | kubectl label pod nginx-app app=web |
| 修改标签 | kubectl label --overwrite <资源类型> <资源名称> <键>=<新值> | kubectl label --overwrite pod nginx-app app=backend |
| 删除标签 | kubectl label <资源类型> <资源名称> <键>- | kubectl label pod nginx-app app- |

### **82、说说你对Job这种资源对象的了解?**

|  |  |
| --- | --- |
| 概念 | 描述 |
| 任务类型 | Job主要用于管理一次性任务或批处理任务 |
| Pod管理 | 创建一个或多个Pod，Pod包含执行任务所需的容器 |
| 任务完成策略 | 有两种策略，Parallel允许多个Pod并行运行，Complete确保所有Pod成功完成 |
| 重试机制 | 支持任务的重试，启动新的Pod替代失败的Pod |
| 成功条件 | 配置成功的条件，如成功运行的Pod数量或成功完成的总任务数 |
| 示例 | 一个简单的Job定义包括容器镜像、命令和重试策略 |

apiVersion: batch/v1kind: Jobmetadata: name: example-jobspec: template: metadata: name: example-job-pod spec: containers: - name: example-container image: example-image command: ["echo", "Hello, Kubernetes!"] backoffLimit: 4

### **85、删除一个Pod会发生什么事情?**

答：Kube-apiserver会接受到用户的删除指令，默认有30秒时间等待优雅退出，超过30秒会被标记为死亡状态，此时Pod的状态Terminating，kubelet看到pod标记为Terminating就开始了关闭Pod的工作；

关闭流程如下：

pod从service的endpoint列表中被移除； 如果该pod定义了一个停止前的钩子，其会在pod内部被调用，停止钩子一般定义了如何优雅的结束进程； 进程被发送TERM信号（kill -14） 当超过优雅退出的时间后，Pod中的所有进程都会被发送SIGKILL信号（kill -9）。

### **87、k8s是怎么进行服务注册的?**

Pod启动后会加载当前环境所有Service信息，以便不同Pod根据Service名进行通信。

### **88、k8s集群外流量怎么访问Pod?**

可以通过Service的NodePort方式访问，会在所有节点监听同一个端口，比如：30000，访问节点的流量会被重定向到对应的Service上面。

### **90、Kubernetes与Docker Swarm的区别如何?**

Kubernetes有助于管理更复杂的软件应用容器；Docker Swarm只能编排简单的Docker容器 支持自动扩缩容；不支持自动扩缩容 滚动更新回滚；滚动更新不能自动回滚 仅能与同一pod的容器共享存储卷；可以和其他任何容器共享存储卷

### **94、什么是Container Orchestration（容器编排）?**

意味着各个容器中的所有服务协同工作以满足单个服务器的需求

### **95、Container Orchestration需要什么?**

|  |  |
| --- | --- |
| 组件/功能 | 描述 |
| 编排引擎 | 管理和调度容器的核心组件，负责实现自动化部署和伸缩 |
| 编排策略 | 决定容器如何分布、伸缩和调度的规则和算法 |
| 服务发现 | 动态发现和管理容器化应用程序中的服务和网络细节 |
| 负载均衡 | 将流量分发到不同的容器实例，确保应用程序高可用性 |
| 自动伸缩 | 根据应用程序负载和需求自动调整容器实例数量 |
| 健康检查 | 定期检查容器实例的健康状态，以确保高可用性 |
| 日志管理 | 收集、存储和管理容器产生的日志信息 |
| 配置管理 | 管理应用程序配置的中心化系统，支持动态配置更新 |
| 安全性和权限 | 提供容器级别和集群级别的安全性和访问控制 |
| 存储编排 | 管理和调度容器中的数据存储，支持持久化和共享存储 |
| 版本控制 | 管理应用程序和容器镜像的版本，支持回滚和升级 |
| 跨节点通信 | 实现容器之间的通信，跨节点通信保障分布式应用正常运行 |
| 故障恢复 | 处理节点故障、容器故障，确保应用程序的高可用性 |

### **97、Kubernetes如何简化容器化部署?**

由于典型应用程序将具有跨多个主机运行的容器集群，因此所有这些容器都需要相互通信。因此，要做到这一点，你需要一些能够负载平衡，扩展和监控容器的东西。由于Kubernetes与云无关并且可以在任何公共/私有提供商上运行，因此必须是您简化容器化部署的选择。

### **98、对Kubernetes的集群了解多少？**

|  |  |
| --- | --- |
| 概念 | 描述 |
| Master节点 | Kubernetes集群的控制中心，负责整个集群的管理和控制。主要组件包括kube-apiserver、kube-controller-manager、kube-scheduler和etcd等。 |
| Node节点 | 集群中的工作节点，负责运行应用程序容器。每个Node节点上都运行有kubelet和kube-proxy，与Master节点协同工作。 |
| Pod | Kubernetes中最小的部署单元，包含一个或多个容器。Pod共享网络和存储资源，它们在同一节点上运行，并可以直接通过localhost通信。 |
| Service | 服务是一种抽象，定义了一组Pod及其访问方式。它确保了Pod的稳定网络标识和动态路由。 |
| Namespace | 命名空间用于将集群划分为多个虚拟集群，每个命名空间中的资源相互隔离。 |
| ConfigMap和Secret | ConfigMap用于将配置数据提供给应用程序，而Secret用于存储敏感信息，如密码和API密钥。 |

### **99、什么是Google容器引擎?**

Google Container Engine（GKE）是Docker容器和集群的开源管理平台。这个基于Kubernetes的引擎仅支持在Google的公共云服务中运行的群集。

### **100、什么是Heapster?**

Heapster是由每个节点上运行的Kubelet提供的集群范围的数据聚合器。此容器管理工具在Kubernetes集群上本机支持，并作为pod运行，就像集群中的任何其他pod一样。因此，它基本上发现集群中的所有节点，并通过机上Kubernetes代理查询集群中Kubernetes节点的使用信息。

### **107、kube-apiserver和kube-scheduler的作用是什么?**

kube-apiserver遵循横向扩展架构，是主节点控制面板的前端。这将公开Kubernetes主节点组件的所有API，并负责在Kubernetes节点和Kubernetes主组件之间建立通信。 kube-scheduler负责工作节点上工作负载的分配和管理。因此，它根据资源需求选择最合适的节点来运行未调度的pod，并跟踪资源利用率。它确保不在已满的节点上调度工作负载。

### **108、你能简要介绍一下Kubernetes控制管理器吗?**

多个控制器进程在主节点上运行，但是一起编译为单个进程运行，即Kubernetes控制器管理器。因此，Controller Manager是一个嵌入控制器并执行命名空间创建和垃圾收集的守护程序。它拥有责任并与API服务器通信以管理端点。 常见控制器： - node controller - hpa controller - replication controller - service account & token controller - endpoints controller

### **112、什么是Ingress网络，它是如何工作的?**

Ingress网络是一组规则，充当Kubernetes集群的入口点。这允许入站连接，可以将其配置为 通过可访问的URL，负载平衡流量或通过提供基于名称的虚拟主机从外部提供服务。因此， Ingress是一个API对象，通常通过HTTP管理集群中服务的外部访问，是暴露服务的最有效方 式。

现在，让我以一个例子向您解释Ingress网络的工作。

有2个节点具有带有Linux桥接器的pod和根网络命名空间。除此之外，还有一个名为 flannel0（网络插件）的新虚拟以太网设备被添加到根网络中。

现在，假设我们希望数据包从pod1流向pod 4。

因此，数据包将pod1的网络保留在eth0，并进入veth0的根网络。 然后它被传递给cbr0，这使得ARP请求找到目的地，并且发现该节点上没有人具有目的地IP地址。

因此，桥接器将数据包发送到flannel0，因为节点的路由表配置了flannel0。 现在，flannel守护程序与Kubernetes的API服务器通信，以了解所有pod IP及其各自的节 点，以创建pods IP到节点IP的映射。

网络插件将此数据包封装在UDP数据包中，其中额外的标头将源和目标IP更改为各自的节点， 并通过eth0发送此数据包。

现在，由于路由表已经知道如何在节点之间路由流量，因此它将数据包发送到目标节点2。 数据包到达node2的eth0并返回到flannel0以解封装并在根网络命名空间中将其发回。 同样，数据包被转发到Linux网桥以发出ARP请求以找出属于veth1的IP。 数据包最终穿过根网络并到达目标Pod4。

### **113、您对云控制器管理器有何了解?**

Cloud Controller Manager负责持久存储，网络路由，从核心Kubernetes特定代码中抽象出特定于云的代码，以及管理与底层云服务的通信。它可能会分成几个不同的容器，具体取决于您运行的是哪个云平台，然后它可以使云供应商和Kubernetes代码在没有任何相互依赖的情况下开发。因此，云供应商开发他们的代码并在运行Kubernetes时与Kubernetes云控制器管理器连接。 - node controller - route controller - volume controller - service controller

### **114、什么是Container资源监控?**

|  |  |
| --- | --- |
| 概念 | 描述 |
| 容器资源监控 | 监控容器运行时的资源使用情况，包括CPU利用率、内存使用、网络IO等。用于性能分析、故障排查和资源优化。 |
| 监控指标 | 常见的容器监控指标包括CPU利用率、内存使用量、磁盘IO、网络IO、容器启动时间等。 |
| 监控工具 | 一些常用的容器监控工具包括Prometheus、Grafana、cAdvisor等。它们提供可视化和警报功能，帮助管理员实时监控和管理容器。 |
| 数据收集 | 监控工具通过在容器内运行代理或直接访问容器运行时API，收集实时的性能数据。 |
| 可视化和报警 | 容器监控工具通常提供仪表盘和报警功能，让管理员能够直观地查看容器集群的状态，并在发生异常时及时收到通知。 |

### **115、Replica Set和Replication Controller之间有什么区别?**

Replica Set 和 Replication Controller几乎完全相同。它们都确保在任何给定时间运行指定数量的pod副本。不同之处在于复制pod使用的选择器。Replica Set使用基于集合的选择器，而Replication Controller使用基于权限的选择器。

### **117、使用Kubernetes时可以采取哪些最佳安全措施?**

* 日志记录生产环境中的所有内容
* 定期对环境应用安全更新
* 实施网络分割
* 为资源制定严格的策略/规则
* 实施持续安全漏洞扫描
* 提供对 k8s 节点的有限直接访问
* 定义资源限制
* 使用私有镜像仓库