**简述什么是Kubernetes？**

* Kubernetes 是一个快速发展的平台，用于管理基于容器的应用程序及其相关的网络和存储组件。 Kubernetes 专注于应用程序工作负载，而不是底层基础设施组件。 Kubernetes 提供了一种声明式的部署方法，由一组强大的管理操作 API 提供支持。

**简述Kubernetes和Docker的关系？**

* Docker 提供容器的生命周期管理和，Docker 镜像构建运行时容器。它的主要优点是将将软件/应用程序运行所需的设置和依赖项打包到一个容器中，从而实现了可移植性等优点。
* Kubernetes 用于关联和编排在多个主机上运行的容器。

**简述Kubernetes中什么是Kubectl、Kubelet？**

* Kubectl 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。
* Kubelet 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。

**简述Kubernetes常见的部署方式？**

* 常见的Kubernetes部署方式有：
* kubeadm：也是推荐的一种部署方式；
* 二进制部署
* minikube：在本地轻松运行一个单节点 Kubernetes 群集的工具。

**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。

**简述Kubernetes master和worker节点的区别？**

* master：k8s集群的管理节点，负责管理集群，提供集群的资源数据访问入口。拥有Etcd存储服务（可选），运行Api Server进程，Controller Manager服务进程及Scheduler服务进程。
* node（worker）：Node（worker）是Kubernetes集群架构中运行Pod的服务节点，是Kubernetes集群操作的单元，用来承载被分配Pod的运行，是Pod运行的宿主机。运行docker eninge服务，守护进程kunelet及负载均衡器kube-proxy。

**什么是 Kubernetes Services**

* Service：Service定义了Pod的逻辑集合和访问该集合的策略，是真实服务的抽象。Service提供了一个统一的服务访问入口以及服务代理和发现机制，关联多个相同Label的Pod，用户不需要了解后台Pod是如何运行。

**简述kube-proxy作用？**

* kube-proxy 运行在所有节点上，它监听 apiserver 中 service 和 endpoint 的变化情况，创建路由规则以提供服务 IP 和负载均衡功能。简单理解此进程是Service的透明代理兼负载均衡器，其核心功能是将到某个Service的访问请求转发到后端的多个Pod实例上。

**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。

**简述kube-proxy ipvs原理？**

* 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。

**简述kube-proxy ipvs和iptables的异同？**

* 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更复杂的复制均衡算法（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。

**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。

**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：
* 1、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。
* 2、Apiserver收到指令后，通知给controller-manager创建一个资源对象。
* 3、Controller-manager通过api-server将pod的配置信息存储到ETCD数据中心中。
* 4、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。
* 5、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。

**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。

**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。

**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。

**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。

**简述什么是负载均衡器？**

* 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。

**简述Kubernetes RBAC及其特点？**

* RBAC是基于角色的访问控制，是一种基于个人用户的角色来管理对计算机或网络资源的访问的方法。
* 相对于其他授权模式，RBAC具有如下优势：
* 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。

**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。

**简述Kubernetes Calico网络组件？**

* Calico是一个基于BGP的纯三层的网络方案，与OpenStack、Kubernetes、AWS、GCE等云平台都能够良好地集成。
* 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。

**简述什么是**PersistentVolume**？**

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。

**简述Kubernetes PV生命周期内的阶段？**

* 某个PV在生命周期中可能处于以下4个阶段（Phaes）之一。
* 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。