云原生面试题

**什么是云原生？**

答案：

云原生是基于分布式云的一种技术体系和方法论，它以容器、微服务、DevOps等技术为基础，旨在构建和运行现代应用程序。云原生不仅涉及具体的技术，如容器化、微服务架构、自动化管理等，还包括一套方法论，如DevOps实践、持续交付等，旨在充分利用云平台的弹性、分布式特性，以及快速响应变化的能力。

**云原生的目的？**

答案：

云原生的目标是帮助组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。这些应用能够利用云原生的代表技术，如容器、微服务、服务网格等，构建容错性好、易于管理和便于观察的松耦合系统。结合可靠的自动化手段，云原生技术使工程师能够轻松地对系统做出频繁和可预测的重大变更，从而实现业务的快速迭代和响应市场变化。

**微服务架构的优势有哪些？**

答案：  
云原生应用通常采用微服务架构，将应用拆分为一组小而自治的服务单元，每个服务专注于独立的业务功能。这种架构方式取代了传统的单体应用架构，使得应用更加灵活和易于管理。

优势：

独立部署：每个微服务都可以独立部署和扩展，无需影响其他服务，提高了开发和部署的效率。

可维护性：微服务之间的解耦合降低了维护的难度，开发人员可以专注于特定服务的更新和维护。

容错性：一个服务的故障不会影响整个应用，其他服务仍然可以正常运行。

**Docker 是什么？它与虚拟机的区别是什么？**

答案：

Docker 是一种开源的容器化平台，可以使开发人员和运维人员轻松地创建、部署和运行应用程序。Docker 容器和虚拟机最大的区别在于，Docker 容器共享主机操作系统的内核，因此更加轻量级、快速和节省资源。

**什么是 Dockerfile？它的作用是什么？**

答案：

Dockerfile 是一种文本文件，包含一系列指令，用于构建 Docker 镜像。通过 Dockerfile，可以定义应用程序的运行环境、依赖关系、配置信息等，以及构建过程中需要执行的操作，例如安装软件、复制文件等。

**Docker 镜像和容器的区别是什么？**

答案：

Docker 镜像是应用程序和其依赖关系的静态快照，可以看作是一个只读的模板。Docker 容器是 Docker 镜像的一个实例，可以运行应用程序和修改应用程序的状态，例如添加、删除、修改文件。

**如何将 Docker 容器连接到其他容器或主机？**

答案：

Docker 提供了多种网络模式，包括桥接模式、主机模式、覆盖网络模式等，可以实现容器之间和容器与主机之间的网络连接。常用的方法包括使用 Docker Compose、Docker Swarm 和 Kubernetes 等容器编排工具。

**如何将数据持久化到 Docker 容器中？**

答案：

Docker 提供了多种方法，例如使用数据卷、绑定挂载主机目录、使用网络存储卷等，可以实现数据的持久化。其中，数据卷是最常用的方法，它可以在容器之间共享数据，并且可以在容器删除后仍然保留数据。

**Docker是什么？**

答案：

Docker是一个容器化平台，它以容器的形式将你的应用程序及所有的依赖项打包在一起，以确保你的应用程序在任何环境中无缝运行。

到目前为止，Docker看起来还很像一个典型的Linux虚拟化栈。实际上，Docker镜像的第二层是root文件系统rootfs，它位于引导文件系统之上。rootfs可以是一种或多种操作系统（如Debian或者 Ubuntu文件系统）。

在传统的Linux 引导过程中，root文件系统会最先以只读的方式加载，当引导结束并完成了完整性检查之后，它才会被切换为读写模式。但是在Docker中root文件系统永远只能是只读状态，并且Docker利用联合加载（union mount）技术又会在root文件系统层加载更多的只读文件系统。联合加载指的是一次同时加载多个文件系统，但是在外面看起来只能看到一个文件系统。联合加载会将各层文件系统叠加到一起，这样最终的文件系统会包含所有底层的文件和目录。

Docker将这样的文件系统称为镜像。一个镜像可以放到另一个镜像的顶部。位于下面的镜像称为父镜像（parent image），可以依次类推，直到镜像栈的最底部，最底部的镜像称为基础镜像（base image）。最后，当从一个镜像启动容器时，Docker会在该镜像的最顶层加载一个读写文件系统。如果想在Docker中运行的程序就是在这个读写层中执行的。

**如何进行 Docker 镜像的构建和发布？**

答案：

可以使用 Dockerfile 来构建 Docker 镜像，并且可以使用 Docker Hub、私有仓库或者其他镜像仓库来发布镜像。常用的命令包括 docker build、docker push 等。

**如何进行 Docker 容器的监控和日志管理？**

答案：

Docker 提供了多种工具，例如 Docker Stats、Docker Events、Docker Logs 等，可以实现容器的监控和日志管理。

**Docker Compose 是什么？**

答案：

Docker Compose 是 Docker 官方提供的容器编排工具，可以通过 YAML 文件定义多个容器组成的应用程序，并且可以统一管理容器的配置、网络、数据卷等。Docker Compose 可以通过命令行工具或者图形化界面进行操作。

**Docker 如何临时退出正在交互容器终端？**

答案：

使用快捷键Ctrl+p，后Ctrl+q，如果按Ctrl+c会使容器内的应用进程终止，进而会使容器终止。

**什么是联合文件系统**

答案：

联合文件系统指的是一次同时加载多个文件系统，但是在外面看起来只能看到一个文件系统。联合加载会将各层文件系统叠加到一起，这样最终的文件系统会包含所有底层的文件和目录。

**如何控制容器占用系统资源（CPU、内存）的份额？**

答案：

使用docker [container] create命令创建容器或使用docker [con­tainer] run创建并启动容器的时候，可以使用-c | - cpu -shares[=O]参数来调整容器使用CPU的权重；使用-ml-memory[=MEMORY]参数来调整容器使用内存的大小。

**什么是Docker的数据卷?**

答案：

数据卷是一个供容器使用的特殊目录，位于容器中。可将宿主机的目录挂载到数据卷上，对数据卷的修改操作立刻可见，并且更新数据不会影响镜像，从而实现数据在宿主机与容器之间的迁移。数据卷的使用类似于Linux下对目录进行的mount操作。

**列举工作中常用的几个 git 命令？**

答案：

新增文件的命令：git add file 或者 git add . 提交文件的命令：git commit –m 或者 git commit –a

查看工作区状况：git status –s

拉取合并远程分支的操作：git fetch/git merge 或者 git pull

查看提交记录命令：git reflog

**提交时发生冲突，你能解释冲突是如何产生的吗？你是如何解决的？**

答案：

开发过程中，我们都有自己的特性分支，所以冲突发生的并不多，但也碰到过。诸如公共类

的公共方法，我和别人同时修改同一个文件，他提交后我再提交就会报冲突的错误。

发生冲突，在 IDE 里面一般都是对比本地文件和远程分支的文件，然后把远程分支上文件的内容手工修改到本地文件，然后再提交冲突的文件使其保证与远程分支的文件一致，这样才会消除冲突，然后再提交自己修改的部分。特别要注意下，修改本地冲突文件使其与远程仓库的文件保持一致后，需要提交后才能消除冲突，否则无法继续提交。必要时可与同事交流，消除冲突。发生冲突，也可以使用命令。

通过 git stash 命令，把工作区的修改提交到栈区，目的是保存工作区的修改；

通过 git pull 命令，拉取远程分支上的代码并合并到本地分支，目的是消除冲突；

通过 git stash pop 命令，把保存在栈区的修改部分合并到最新的工作空间中；

**能不能说一下 git fetch 和 git pull 命令之间的区别？**

答案：

简单来说：git fetch branch 是把名为 branch 的远程分支拉取到本地；而 git pull branch 是在fetch 的基础上，把 branch 分支与当前分支进行 merge；因此 pull = fetch + merge。

**git 跟其他版本控制器有啥区别？**

答案：

GIT 是分布式版本控制系统，其他类似于 SVN 是集中式版本控制系统。

分布式区别于集中式在于：每个节点的地位都是平等，拥有自己的版本库，在没有网络的情

况下，对工作空间内代码的修改可以提交到本地仓库，此时的本地仓库相当于集中式的远程

仓库，可以基于本地仓库进行提交、撤销等常规操作，从而方便日常开发。

**GitLab 在持续集成/持续部署(CI/CD)中的角色是什么？**

答案：

GitLab 在持续集成/持续部署(CI/CD)中扮演的角色非常关键，主要体现在以下几个方面：

1、自动化构建和测试： GitLab 可以自动化执行代码的构建和测试过程，确保代码提交后立即进行验证，这有助于及早发现和修复错误。

2、持续集成管道： GitLab 允许用户配置自定义的CI/CD管道（pipelines），通过定义不同的阶段和任务，实现代码从开发到部署的自动化流程。

3、部署和发布管理： GitLab 支持多环境部署，如开发、测试和生产环境，允许自动或手动触发部署过程，并对部署过程进行监控和管理。

4、集成第三方工具： GitLab 可以与各种第三方工具和服务集成（例如Docker、Kubernetes），以提高CI/CD流程的灵活性和效率。

**GitLab的权限管理机制有哪些特点？**

答案：

1、角色级别： GitLab定义了多个角色级别（如Guest、Reporter、Developer、Maintainer、Owner），每个角色都有不同的权限，以控制对项目的访问和操作。

2、群组和子群组： 可以在群组或子群组级别设置权限，实现对项目和成员的集中管理，简化权限的分配和管理。

3、项目特定权限： 在项目级别可以定义特定的权限设置，允许为不同项目指定不同的访问和操作权限。

4、保护分支： GitLab允许保护特定的分支，限制谁可以提交或推送到这些分支，以防止重要分支的未经授权修改。

**GitLab与GitHub的主要区别是什么？**

GitLab与GitHub的主要区别体现在以下几个方面：

1、集成的CI/CD： GitLab提供了内置的CI/CD服务，而GitHub则需要依赖外部服务（如GitHub Actions）来实现CI/CD。

2、自托管选项： GitLab提供了自托管的解决方案，允许企业在自己的服务器上部署GitLab实例。而GitHub虽然有Enterprise版本，但主要是云服务形式。

3、项目管理功能： GitLab除了版本控制，还提供了更丰富的项目管理工具，如看板、问题跟踪器等。GitHub虽然也有项目管理功能，但GitLab的功能更全面。

4、访问控制和权限管理： GitLab在访问控制和权限管理方面提供了更细粒度的设置，允许更复杂的权限配置。

**DevOps 有几个阶段？**

答案：

DevOps 生命周期有八个阶段：

计划：需要详细的应用类型计划，以便全面了解开发过程。

代码：程序根据最终用户的要求进行编码。

构建：通过合并前面阶段生成的代码来创建应用程序。

测试：这是应用程序创建过程中最重要的阶段。检查应用程序的功能，如果出现问题则重写它。

集成：来自不同程序员的多个代码被组合成一个代码。

部署：在云计算中发布代码供其他用户使用。新的更新都会经过彻底检查，以确保它们不会破坏高流量的网站。

操作：如果需要，对代码进行操作。

监控：应用程序的运行始终受到监控。最终消费者的需求被优先考虑，并做出相应的调整。

**描述 DevOps 的一些主要优势**

DevOps 的主要优点是：

持续交付软件

需要处理的复杂问题更少

更早识别并更快速修复缺陷

更快地交付功能

可靠的工作条件

增强团队合作和沟通

**什么是持续集成（CI）？**

答案：

开发人员进行编码，然后使用持续集成（CI）方法将其代码组合到主存储库中。每个集成都会通过自动构建过程进行检查，使团队能够比发布后更快地发现代码中的错误。

**解释Jenkins的主从架构。**

答案：

每次提交代码时，Jenkins master 都会从远程GitHub存储库检索更新的代码。

它将工作负载分散到所有Jenkins 从机上。

从机按照Jenkins 主机的指示执行、构建、测试并生成测试报告

**Jenkins 管道中的关键概念是什么？**

答案：

Pipeline：CD 管道是用户定义的模型。管道的代码指定如何构建、测试和交付应用程序。

Agent：它是Jenkins 环境的组件，可以运行管道。

Steps：指示Jenkins 在触发时执行的单个任务。

Stage：标识在整个管道（构建、测试、部署阶段）中执行的一组理论上独立的任务

**如何将 Jenkins 从一台服务器复制到另一台服务器？**

答案：

步骤 1：复制相关作业目录，将作业从一个 Jenkins 安装转移到另一个安装。

步骤 2：要复制现有作业，请克隆作业目录并为其指定新名称。

步骤 3：重命名目录以重命名现有作业。

**哪些是顶级的 DevOps 工具？您使用过哪些工具？**

答案：

最流行的 DevOps 工具是

Git：用于版本控制。

Jenkins：用于持续集成。

Selenium：用于连续测试。

Puppet、Chef、Ansible：用于配置管理和部署。

Prometheus：用于连续监控。

Docker：用于容器化。

**什么是CD?**

答案：

CD(Continuous Delivery， 中文意思持续交付)是在持续集成的基础上，将集成后的代码部署到更贴近真实运行环境(类生产环境)中。比如，我们完成单元测试后，可以把代码部署到连接数据库的 Staging 环境中更多的测试。如果代码没有问题，可以继续手动部署到生产环境。

**什么是 Jenkins pipeline**

答案：

Pipeline，简而言之，就是一台运行于 Jenkins 上的工作流框架，将原本独立运行于单个或者多个节点的任务连接起来，实现单个任务难以完成的复杂流程编排与可视化。

Jenkins Pipeline 是一组插件，让 Jenkins 可以实现持续交付管道的落地和实施。

**Jenkins pipelines 相对于 freestyle 项目有哪些优势？**

答案：

与自由式项目相比，Jenkins 管道具有多种优势，包括更好的可追溯性、版本控制、可重用性和可扩展性。管道可以将整个构建过程定义为代码，从而更轻松地管理和自动化复杂的工作流程。

**说一下 OpenStack 及其主要组件？**

答案：

OpenStack 是一系列开源软件，这些软件组成了一个云供给软件，也就是 OpenStack，

意即开源软件或项目栈。

下面是 OpenStack 的主要关键组件：

Nova – 用于在计算级别管理虚拟机，并在计算或管理程序级别执行其他计算任务。

Neutron – 为虚拟机、计算和控制节点提供网络功能。

Keystone – 为所有云用户和 OpenStack 云服务提供身份认证服务。换句话说，我们可以说Keystone 是一个提供给云用户和云服务访问权限的方法。

Horizon – 用于提供图形用户界面。使用图形化管理界面可以很轻松地完成各种日常操作任务。

Cinder – 用于提供块存储功能。通常来说 OpenStack 的 Cinder 中集成了 Chef 和 ScaleIO来共同为计算和控制节点提供块存储服务。

Swift – 用于提供对象存储功能。通常来说，Glance 管理的镜像是存储在对象存储空间的。

像 ScaleIO 这样的外部存储也可以提供对象存储，可以很容易的集成 Glance 服务。

Glance – 用于提供镜像服务。使用 Glance 的管理平台来上传和下载云镜像。

Heat – 用于提供编排服务或功能。使用 Heat 管理平台可以轻松地将虚拟机作为堆栈，并且根据需要可以将虚拟机扩展或收缩。

Ceilometer – 用于提供计量与监控功能。

**什么服务通常在控制节点上运行？**

答案：

以下服务通常在控制节点上运行：

认证服务（KeyStone）

镜像服务（Glance）

Nova 服务比如 Nova API、Nova Scheduler 和 Nova DB

块存储和对象存储服务

Ceilometer 服务

MariaDB / MySQL 和 RabbitMQ 服务

网络(Neutron)和网络代理的管理服务

编排服务（Heat）

**计算节点上虚拟机的默认地址是什么?**

答案：

虚拟机存储在计算节点的 /var/lib/nova/instances。

**Glance 镜像的默认地址是什么？**

答案：

因为 Glance 服务运行在控制节点上，所以 Glance 镜像都被存储在控制节点的/var/lib/glance/images 文件夹下。

**外部 OVS 桥（br-ex）的作用是什么？**

答案：

顾名思义，此网桥转发来往网络的流量，以允许外部访问实例。br-ex 连接物理接口比如 eth2，这样用户网络的浮动 IP 数据从物理网络接收并路由到用户网络端口。

**云计算常见几种部署模式？**

答案：

私有云：云平台资源只给某个单位、或某部分用户内部使用。

公有云：云平台资源开放给社会公众服务。

社区云：云平台资源给几个固定的单位内使用。

混合云：两个或两个以上不同类型的云平台。

**OpenStack中的Domain，project，user，role，token 的概念和关系？**

答案：

Domain：表示 project 和 user 的集合，在公有云或者私有云中常常表示一个客户

Group：一个domain 中的部分用户的集合

Project：IT基础设施资源的集合，比如虚机，卷，镜像等

Role：角色，表示一个 user 对一个 project resource 的权限

Token：一个 user 对于某个目标（project 或者 domain）的一个有限时间段内的身份令牌

**OpenStack中根据权限不同网络可分为几种？**

答案：

Provider network：管理员创建的和物理网络有直接映射关系的虚拟网络。

Tenant network：租户普通用户创建的网络，物理网络对创建者透明，其配置由Neutron根据管理员在系统中的配置决定。

**Docker与k8s之间的关系是什么？**

Kubernetes（简称K8s）和Docker是两个不同的技术，但它们在容器化应用程序方面有着密切的关系。

Docker是一种开源的容器化平台，提供了一种轻量级的容器化解决方案，可以将应用程序及其依赖项打包成一个可移植的容器。这种容器化技术允许开发者将应用及其依赖打包到一个可移植的容器中，发布到任何流行的Linux机器上，实现虚拟化。

Kubernetes（K8s）是一个开源的容器集群管理系统，用于实现容器集群的自动化部署、自动扩缩容、维护等功能。K8s使用Docker容器进行编排，主要围绕pods进行工作，pods是K8s生态中最小的调度单位，可以包含一个或多个容器。K8s解决了裸跑Docker的若干痛点，如单机使用无法有效集群、管理成本攀升、没有有效的容灾自愈机制、没有预设编排模板等。

尽管K8s和Docker在技术上是不同的，但它们可以很好地协同工作。Docker用于构建、分发、运行容器，而K8s则是一个使用Docker容器进行编排的系统，两者共同构成了在容器化应用程序方面的强大工具。

**k8s中，pod的概念？**

答案：

在Kubernetes（简称K8s）中，Pod是其核心的抽象概念，它是Kubernetes系统中可以创建和管理的最小部署单元，同时也是容器化应用运行的基本实体。Pod代表了一组紧密相关的、共享资源的容器集合。简单来说，一个Pod可以包含一个或多个容器，这些容器共享网络命名空间（即拥有相同的IP地址和端口空间）、存储卷和Linux名称空间（如IPC、UTS），并且被一起管理和调度。

**Kubernetes Architecture的不同组件有哪些？**

答案：

Kubernetes Architecture主要有两个组件 - 主节点和工作节点。master和worker节点中包含许多内置组件。主节点具有kube-controller-manager，kube-apiserver，kube-scheduler等。而工作节点具有在每个节点上运行的kubelet和kube-proxy。

**组件Kube-proxy是做什么的？**

答案：

Kube-proxy可以在每个节点上运行，并且可以跨后端网络服务进行简单的TCP / UDP数据包转发。基本上，它是一个网络代理，它反映了每个节点上Kubernetes API中配置的服务。

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

答案：

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。

**什么是ETCD？**

答案：

Etcd是用Go编程语言编写的，是一个分布式键值存储，用于协调分布式工作。因此，Etcd存储Kubernetes集群的配置数据，表示在任何给定时间点的集群状态。

**Kubernetes有哪些不同类型的服务？**

答案：

Cluster IP 、Node Port、Load Balancer、External Name

**什么是Ingress网络，它是如何工作的？**

答案：

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。

**简述Kubernetes常见的部署方式？**

答案：

常见的Kubernetes部署方式有：

kubeadm：也是推荐的一种部署方式；

二进制：相对复杂

minikube：在本地轻松运行一个单节点 Kubernetes 群集的工具。

**简述Kubernetes Replica Set 和 Replication Controller 之间有什么区别？**

答案：

Replica Set 和 Replication Controller 类似，都是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RS 使用基于集合的选择器，而 Replication Controller 使用基于权限的选择器。

**简述Kubernetes中Pod可能位于的状态？**

答案：

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 Pod的LivenessProbe探针的常见方式？**

答案：

kubelet定期执行LivenessProbe探针来诊断容器的健康状态，通常有以下三种方式：

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。

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

答案：

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。

**简述Kubernetes deployment升级过程？**

答案：

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。

**简述Kubernetes deployment升级策略？**

答案：

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。

**简述Kubernetes DaemonSet类型的资源特性？**

答案：

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。

**简述Kubernetes Headless Service？**

答案：

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。

**简述Kubernetes Secret作用？**

答案：

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。

**简述Kubernetes PV和PVC？**

答案：

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”。

**简述Kubernetes Metric Service？**

答案：

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。

**简述Kubernetes中Pod的重启策略?**

答案：

在Kubernetes中，可以通过定义Pod的重启策略来控制容器的重启行为。重启策略可以在Pod的定义文件中指定，有以下三种选项：

Always（始终重启）：无论容器退出的原因是什么，Kubernetes都会自动重启容器。这是默认的重启策略。

OnFailure（仅在失败时重启）：只有当容器以非零状态退出时，Kubernetes才会自动重启容器。如果容器以零状态退出（即正常退出），则不会触发重启。

Never（从不重启）：无论容器退出的原因是什么，Kubernetes都不会自动重启容器。

**简述Kubernetes Pod的常见调度方式?**

答案：

Kubernetes Pod的常见调度方式包括以下几种：

1）默认调度：Kubernetes默认的调度方式是通过调度器（Scheduler）将Pod分配到可用的节点上。调度器根据节点资源的可用性、亲和性和反亲和性规则等因素，选择一个合适的节点来运行Pod。

2）节点亲和性调度：使用节点亲和性调度，可以将Pod调度到具有特定标签或标签选择器匹配的节点上。这可以用于将Pod与特定类型的节点关联，例如将特定任务的Pod调度到具有特定硬件或软件配置的节点上。

3）Pod亲和性调度：使用Pod亲和性调度，可以将多个相关的Pod调度到同一个节点上。这可以用于将需要相互通信或共享资源的Pod部署在同一个节点上，以提高效率和性能。

4）Pod反亲和性调度：使用Pod反亲和性调度，可以避免将多个相关的Pod调度到同一个节点上。这可以用于确保关键任务或故障隔离的Pod在不同的节点上运行，以增加可靠性和稳定性。

5）资源限制调度：Kubernetes可以根据节点的资源限制和Pod的资源需求，进行调度决策。调度器会尽量将Pod调度到具有足够资源的节点上，以避免资源竞争和性能问题。

**镜像的下载策略有哪些？**

答案：

Always：始终尝试下载镜像，如果镜像已存在则重新拉取最新版本。

IfNotPresent：仅在本地不存在镜像时下载。

Never：仅使用本地已存在的镜像，不尝试下载。

**简述Kubernetes RBAC的作用?**

答案：

Kubernetes RBAC（Role-Based Access Control）是一种授权机制，用于管理和控制对Kubernetes集群中资源的访问权限。RBAC通过定义角色和角色绑定来控制用户、服务账户和组对资源的操作权限。

**简述Kubernetes Secret作用?**

答案：

Kubernetes Secret用于安全地存储和管理敏感信息，例如密码、API密钥、证书等。它提供了一种在Kubernetes集群中存储敏感数据的机制，确保这些数据不会以明文形式暴露给Pod中的容器。

**简述Kubernetes Secret有哪些使用方式?**

答案：

Kubernetes Secret有以下几种使用方式：

环境变量（Environment Variables）：可以将Secret中的值作为环境变量注入到Pod中的容器中。通过在Pod的配置中引用Secret的名称和键，容器可以直接访问Secret中的敏感数据。

卷挂载（Volume Mounts）：可以将Secret作为一个卷挂载到Pod中的容器中。通过在Pod的配置中定义一个Volume，并将Secret挂载到该Volume上，容器可以通过文件系统的方式访问Secret中的数据。

镜像拉取凭证（Image Pull Secrets）：可以使用Secret来存储用于拉取私有容器镜像的凭证信息。通过将Secret指定为Pod或Deployment的镜像拉取凭证，Kubernetes可以使用这些凭证来拉取私有镜像。

TLS证书和私钥（TLS Certificates and Private Keys）：可以使用Secret存储TLS证书和私钥，用于在Pod之间进行安全的通信。通过将Secret配置为Ingress或Service的TLS证书，可以实现HTTPS的终止和加密通信。

**简述service服务的作用？**

答案：

Service（服务）：Service是一种抽象的逻辑概念，用于将一组Pod暴露为一个稳定的网络终结点。Service为Pod提供了一个虚拟的IP地址和DNS名称，可以通过这些标识符来访问Pod。

**简述什么是endpoint端点？**

答案：

Endpoint是一个与Service关联的实际网络终结点，它代表了Service所引用的一组Pod的网络地址。Endpoint会动态地根据Pod的状态进行更新，确保Service始终指向正确的Pod。

**简述Ingress的作用？**

答案：

Ingress是一种配置规则，用于将外部流量路由到集群中的Service。Ingress控制器会根据Ingress规则将流量转发到相应的Service，从而实现集群外部的访问。

**简述Kubernetes PV生命周期内的阶段?**

答案：

Kubernetes PV（Persistent Volume）的生命周期可以分为以下几个阶段：

Available（可用）：在这个阶段，PV已经被创建并且可供使用。但是，尚未与任何PVC（Persistent Volume Claim）进行绑定，因此它可以被任何符合要求的PVC请求绑定。

Bound（已绑定）：当PV与一个PVC成功绑定时，它进入绑定阶段。在这个阶段，PV只能由与之绑定的PVC使用，并且不能再被其他PVC绑定。绑定后，PVC可以使用PV中的持久化存储。

Released（已释放）：当PVC与PV解除绑定时，PV进入释放阶段。在这个阶段，PV不再与任何PVC绑定，但它仍然保留着以前绑定的PVC的信息。这是为了防止数据丢失，以便在需要时重新绑定。

Failed（失败）：如果PV的绑定过程中发生错误或绑定失败，它将进入失败阶段。这可能是由于PVC的要求与PV的属性不匹配，或者底层存储出现故障等原因导致的。在这种情况下，需要检查并修复问题，然后重新绑定PV。

Released（已释放）或Terminating（终止中）：当PV被删除时，它会进入释放或终止阶段。在这个阶段，PV将被清理，并且与之关联的存储资源将被释放。一旦PV完成删除过程，它将不再存在。

**简述Kubernetes所支持的存储供应模式?**

答案：

Kubernetes支持多种存储供应模式，以满足不同应用场景和需求。以下是Kubernetes支持的存储供应模式的简要描述：

主机路径（HostPath）：使用主机节点上的本地文件系统作为存储。适用于单节点测试或开发环境。

空白存储（EmptyDir）：在Pod所在节点上创建临时目录，用作Pod中容器之间共享数据的临时存储。

持久卷（PersistentVolume）：将外部存储卷（如云存储、网络存储等）挂载到Pod中。PersistentVolume是独立于Pod的存储资源，可以在多个Pod之间共享。

持久卷声明（PersistentVolumeClaim）：用于请求和使用PersistentVolume的抽象。Pod通过声明PersistentVolumeClaim来请求所需的存储资源。

本地存储（Local）：将本地存储设备挂载到Pod中，适用于需要高性能和低延迟的应用。

网络存储（Network Storage）：使用网络存储解决方案，如NFS（Network File System）或Ceph等，将存储卷挂载到Pod中。

云存储（Cloud Storage）：使用云服务提供商的存储解决方案，如AWS EBS（Elastic Block Store）或Azure Disk等。

分布式存储（Distributed Storage）：使用分布式存储系统，如GlusterFS或Ceph，将存储卷挂载到Pod中，以实现高可用性和扩展性。

**简述什么是Kubernetes Metric Service?**

答案：

Kubernetes Metric Service（Kubelet Metrics Service）是Kubernetes中的一个组件，用于收集和暴露节点上的资源使用情况和性能指标。它通过kubelet进程在每个节点上运行，并提供了一组API来获取节点的度量数据。

Kubernetes Metric Service收集的指标包括CPU使用率、内存使用量、网络流量、磁盘IO等。这些指标可以帮助运维人员和开发人员监控和调优集群的性能，并进行自动化的水平扩展和负载均衡。

**容器和主机部署应用的区别是什么？**

答案：

容器的中心思想就是秒级启动；一次封装、到处运行；这是主机部署应用无法达到的效果，但同时也更应该注重容器的数据持久化问题。

另外，容器部署可以将各个服务进行隔离，互不影响，这也是容器的另一个核心概念。

**什么是ReplicaSet？**

答案：

ReplicaSet是Kubernetes的控制器之一，用于确保在集群中运行指定数量的Pod副本。如果Pod的数量少于指定的副本数，ReplicaSet将创建新的Pod副本；如果Pod的数量多于指定的副本数，ReplicaSet将删除多余的Pod。

**什么是Deployment？**

答案：

Deployment是Kubernetes的控制器之一，用于声明性地管理Pod副本集。它允许定义Pod模板、副本数和更新策略，使得应用程序的部署和更新变得简单可控。

**什么是命名空间（Namespace）？**

答案：

命名空间是一种在Kubernetes集群中创建多个虚拟集群的机制。它可以用于隔离和管理不同的应用程序、团队或环境。

**什么是Kubernetes的水平自动扩展HPA（Horizontal Pod Autoscaling）？**

答案：

水平自动扩展是Kubernetes的功能之一，根据应用程序的负载自动调整Pod副本数。它基于CPU利用率或自定义指标来进行自动扩展。

**什么是DaemonSet？**

答案：

DaemonSet是Kubernetes的控制器之一，用于在集群中的每个节点上运行一个Pod副本。它适用于在集群中的每个节点上运行系统级别的守护进程。

**什么是亲和性（Affinity）和反亲和性（Anti-Affinity）？**

答案：

亲和性和反亲和性是Pod调度的约束条件。通过使用亲和性，可以将Pod调度到指定的节点；通过使用反亲和性，可以避免将Pod调度到指定的节点。

**什么是Init容器（Init Container）？**

答案：

Init容器是Pod中的一个额外容器，用于在主应用程序容器启动之前运行初始化任务。它可以用于数据准备、配置下载等任务。

**什么是Kubernetes中的水平和垂直扩展？**

答案：

水平扩展（Horizontal Scaling）指的是增加Pod副本数来处理更多的负载。垂直扩展（Vertical Scaling）指的是增加或减少单个Pod的资源限制。

**什么是Helm？**

答案：

Helm是Kubernetes的包管理工具，用于简化应用程序的部署和管理。它允许定义和版本化应用程序的Charts（图表），并通过Helm命令进行安装、升级和删除。

**什么是Kubernetes的状态管理器（StatefulSet）？**

答案：

StatefulSet是Kubernetes的控制器之一，用于管理有状态应用程序的部署。它确保Pod的稳定网络标识和有序启动、停止，适用于数据库、队列等有状态应用程序。

**如何将应用程序部署到 Kubernetes？**

答案：

可以使用Deployment、StatefulSet或DaemonSet等资源对象来部署应用程序。

**如何水平扩展 Deployment？**

答案：

可以通过更改Deployment的副本数来水平扩展应用程序。例如，使用kubectl scale命令或更改Deployment的replicas字段。

**什么是 Sidecar 容器？**

答案：

Sidecar容器是与主应用程序容器共同运行的辅助容器，用于提供额外的功能或服务。

**什么是 CNI（Container Network Interface）？**

答案：

CNI是一种规范，用于定义容器运行时和网络插件之间的接口，以实现容器网络的配置和管理。

**Kubernetes 有哪些特点？**

答案：

可移植: 支持公有云，私有云，混合云，多重云（multi-cloud）

可扩展: 模块化，插件化，可挂载，可组合

自动化: 自动部署，自动重启，自动复制，自动伸缩/扩展

**请简要介绍一下Kubernetes？**

答案：

Kubernetes是一种开源的容器编排平台，可以用于自动化部署、扩展和管理容器化应用程序。Kubernetes允许您在不同的环境中轻松部署容器化应用程序，包括物理机、虚拟机、公有云和私有云等。