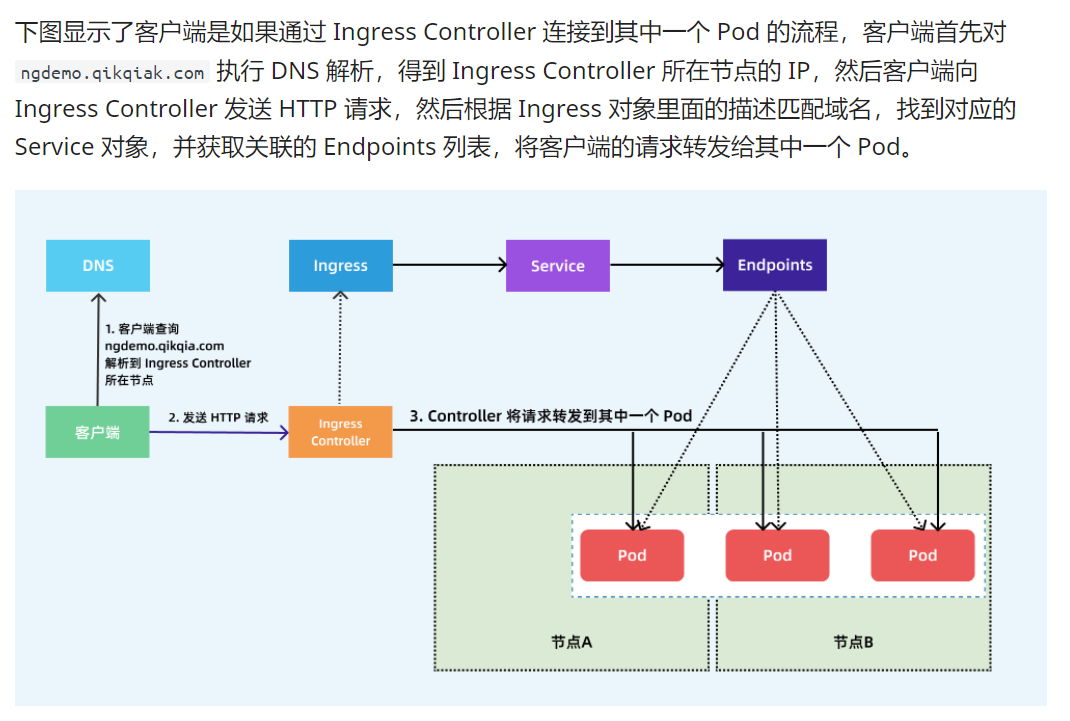
# K8s训练营

# <https://zhuanlan.zhihu.com/p/365759073>

<https://www.qikqiak.com/k8strain/network/ingress/nginx/>



平安现在主要使用 nginx-ingress-controller

# K8s资源利用率查询

通过 kubectl 查看 K8s 内节点、Pod 资源使用情况

[https://blog.csdn.net/yangchao1125/article/details/113243688https://blog.csdn.net/yangchao1125/article/details/113243688](https://blog.csdn.net/yangchao1125/article/details/113243688https:/blog.csdn.net/yangchao1125/article/details/113243688)

<https://segmentfault.com/a/1190000021493607>

## 容器环境下资源限制与jvm内存回收

<https://www.cnblogs.com/noah-luo/p/14721172.html>

<https://kubesphere.io/zh/blogs/deep-dive-into-the-k8s-request-and-limit/>

# 资源限制

<https://www.cnblogs.com/zhangrui153169/p/12988666.html>

# 端口映射

<https://www.lxlinux.net/2317.html>

<https://www.cnblogs.com/sinpo828/p/14103831.html>

# 滚动更新

滚动更新：

在金丝雀发布基础上的进一步优化改进，是一种自动化程度较高的发布方式，用户体验比较平滑，是目前成熟型技术组织所采用的主流发布方式。一次滚动式发布一般由若干个发布批次组成，每批的数量一般是可以配置的(可以通过发布模板定义)。例如，第一批1台(金丝雀)，第二批10%，第三批 50%，第四批100%。每个批次之间留观察间隔，通过手工验证或监控反馈确保没有问题再发下一批次，所以总体上滚动式发布过程是比较缓慢的 (其中金丝雀的时间一般会比后续批次更长，比如金丝雀10 分钟，后续间隔 2分钟)。

实战

1、命令行监测

打开一个窗口按如下操作：

kubectl get pods -l app=myapp -n test -w

将事先准备好的例子，从myapp:v1改成myapp:v3

保存退出，执行如下

kubectl apply -f deploy-demo.yaml

再在监测的那个窗口可以看到信息如下：

pending表示正在进行调度，ContainerCreating表示正在创建一个pod，running表示运行一个pod，running起来一个pod之后再Terminating(停掉)一个pod，以此类推，直到所有pod完成滚动升级

在另外一个窗口执行kubectl get rs -n test，显示如下：

上面可以看到rs有两个，上面那个是升级之前的，已经被停掉，但是可以随时回滚

kubectl rollout history deployment myapp-deploy -n test

查看myapp-deploy这个控制器的滚动历史，显示如下：

需要回滚的话kubectl rollout undo作如下：

2、扩容到5个

cat deploy-demo.yaml

修改replicas数值是5

kubectl apply -f deploy-demo.yaml

kubectl get pods -n test

显示扩容成功了

3、修改maxSurge和maxUnavailable用来控制滚动更新的更新策略

修改更新策略最多不可用0个，也就是少不能少于5个，最大不能超过6个

kubectl patch deployment myapp-deploy -p '{"spec":{"strategy":{"rollingUpdate":{"maxSurge":1,"maxUnavailable":0}}}}' -n test

然后执行

kubectl describe deployment myapp-deploy -n test

查看myapp-deploy这个控制器的详细信息

上面可以看到RollingUpdateStrategy: 0 max unavailable, 1 max surge

这个rollingUpdate更新策略变成了刚才设定的，因为我们设定的pod副本数是5，0和1表示最少不能少于5个pod，最多不能超过6个pod

这个就是通过控制RollingUpdateStrategy这个字段来设置滚动更新策略的。

————————————————

版权声明：本文为CSDN博主「傅一一」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/weixin_42317626/article/details/113022352>

## **滚动更新与回滚**

1. 更新镜像至最新版本

[root@node-1 ~]# kubectl set image daemonsets fluentd-elasticsearch fluentd-elasticsearch=quay.io/fluentd\_elasticsearch/fluentd:latest -n kube-system

daemonset.extensions/fluentd-elasticsearch image updated

2. 查看滚动更新状态

[root@node-1 ~]# kubectl rollout status daemonset -n kube-system fluentd-elasticsearch

Waiting for daemon set "fluentd-elasticsearch" rollout to finish: 1 out of 3 new pods have been updated...

Waiting for daemon set "fluentd-elasticsearch" rollout to finish: 1 out of 3 new pods have been updated...

Waiting for daemon set "fluentd-elasticsearch" rollout to finish: 1 out of 3 new pods have been updated...

Waiting for daemon set "fluentd-elasticsearch" rollout to finish: 2 out of 3 new pods have been updated...

Waiting for daemon set "fluentd-elasticsearch" rollout to finish: 2 out of 3 new pods have been updated...

Waiting for daemon set "fluentd-elasticsearch" rollout to finish: 2 out of 3 new pods have been updated...

Waiting for daemon set "fluentd-elasticsearch" rollout to finish: 2 of 3 updated pods are available...

daemon set "fluentd-elasticsearch" successfully rolled out

3. 查看DaemonSet详情，可以看到DS滚动更新的过程：DaemonSet先将node上的pod删除然后再创建

<https://cloud.tencent.com/developer/article/1531128>

# K8s control manager

23. What is the Kubernetes controller manager?

The controller manager is a daemon that is used for embedding core control loops, garbage collection, and Namespace creation. It enables the running of multiple processes on the master node even though they are compiled to run as a single process.

24. What are the types of controller managers?

The primary controller managers that can run on the master node are the endpoints controller, service accounts controller, namespace controller, node controller, token controller, and replication controller.

# HeadLess-Service

在一个集群中有这个几个组件：pod-a，svc-b，pod-b1，pod-b2。当 pod-a 想访问 pod-b 中的应用程序时，先会把请求打到 svc-b，再由 svc-b 将请求随机转发到 pod-b1或 pod-b2。

如果有个需求：pod-a 需要同时连接到 pod-b1和 pod-b2 ，这时再采用 svc-b 转发显然已经不能满足需求了。那 pod-a 该如何获取到 pod-b1和 pod-b2 的 IP 地址呢？采用 handless service 就可以实现。

————————————————

版权声明：本文为CSDN博主「骑着蜗牛向前跑」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/yy_diego/article/details/109144442>

# statefulset

对于redis, mysql这种有状态的服务,我们使用statefulset方式为首选. 我们这边主要就是介绍statefulset这种方式, statefulset的设计原理模型:

1. 拓扑状态: 应用的多个实例之间不是完全对等的关系, 这个应用实例的启动必须按照某些顺序启动. 比如应用的主节点A要先于从节点B启动. 而如果你把A和B两个Pod删除掉, 他们再次被创建出来是也必须严格按照这个顺序才行. 并且, 新创建出来的Pod, 必须和原来的Pod的网络标识一样, 这样原先的访问者才能使用同样的方法, 访问到这个新的Pod.
2. 存储状态: 应用的多个实例分别绑定了不同的存储数据. 对于这些应用实例来说, Pod A第一次读取到的数据, 和隔了十分钟之后再次读取到的数据, 应该是同一份, 哪怕在此期间Pod A被重新创建过. 一个数据库应用的多个存储实例.

<https://segmentfault.com/a/1190000022993205>

36. What is Kubelet?

The kubelet is a service agent that controls and maintains a set of pods by watching for pod specs through the Kubernetes API server. It preserves the pod lifecycle by ensuring that a given set of containers are all running as they should. The kubelet runs on each node and enables the communication between the master and slave nodes.

## What is Kube-proxy?

Kube-proxy is an implementation of a load balancer and network proxy used to support service abstraction with other networking operations. Kube-proxy is responsible for directing traffic to the right container based on IP and the port number of incoming requests.

* Acts as a load balancer and network proxy to perform service on a single worker node
* Manages pods on nodes, volumes, secrets, the creation of new containers, health check-ups, etc.
* A proxy service that runs on every node that makes services available to the external host

19. What is ‘Heapster’ in Kubernetes?

In this Kubernetes interview question, the interviewer would expect a thorough explanation. You can explain what it is and also it has been useful to you (if you have used it in your work so far!). A Heapster is a performance monitoring and metrics collection system for data collected by the Kublet. This aggregator is natively supported and runs like any other pod within a Kubernetes cluster, which allows it to discover and query usage data from all nodes within the cluster.

26. What are the different services within Kubernetes?

Different types of Kubernetes services include:

Cluster IP service

Node Port service

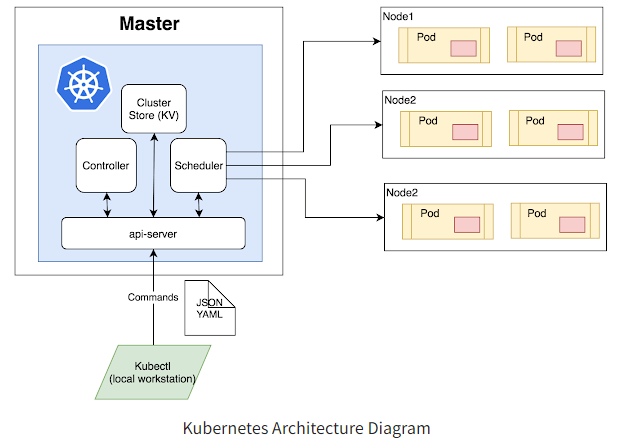
External Name Creation service and

Load Balancer service

## **[config map and secretq](https://www.knowledgehut.com/interview-questions/kubernetes" \l "collapse-beginner-2405)**

Config maps ideally stores application configuration in a plain text format whereas Secrets store sensitive data like password in an encrypted format. Both config maps and secrets can be used as volume and mounted inside a pod through a pod definition file.

# K8s架构



## the types of controller managers

Types of controller managers are: 1) endpoints controller, 2) service accounts controller, 3) node controller, 4) namespace controller, 5) replication controller, 6) token controller.

# 灵雀云k8s题目

<https://blog.51cto.com/yw666/2550027>

一、简述你知道的几种CNI网络插件，并详述其工作原理。K8s常用的CNI网络插件 （calico && flannel），简述一下它们的工作原理和区别。

1、calico根据iptables规则进行路由转发，并没有进行封包，解包的过程，这和flannel比起来效率就会快多

calico包括如下重要组件：Felix，etcd，BGP Client，BGP Route Reflector。下面分别说明一下这些组件。

Felix：主要负责路由配置以及ACLS规则的配置以及下发，它存在在每个node节点上。

etcd：分布式键值存储，主要负责网络元数据一致性，确保Calico网络状态的准确性，可以与kubernetes共用；

BGPClient(BIRD), 主要负责把 Felix写入 kernel的路由信息分发到当前 Calico网络，确保 workload间的通信的有效性；

BGPRoute Reflector(BIRD), 大规模部署时使用，摒弃所有节点互联的mesh模式，通过一个或者多个 BGPRoute Reflector 来完成集中式的路由分发

通过将整个互联网的可扩展 IP网络原则压缩到数据中心级别，Calico在每一个计算节点利用 Linuxkernel 实现了一个高效的 vRouter来负责数据转发，而每个vRouter通过 BGP协议负责把自己上运行的 workload的路由信息向整个Calico网络内传播，小规模部署可以直接互联，大规模下可通过指定的BGProute reflector 来完成。这样保证最终所有的workload之间的数据流量都是通过 IP包的方式完成互联的。

2、Flannel的工作原理：

Flannel实质上是一种“覆盖网络(overlay network)”，也就是将TCP数据包装在另一种网络包里面进行路由转发和通信，目前已经支持UDP、VxLAN、AWS VPC和GCE路由等数据转发方式。

默认的节点间数据通信方式是UDP转发。

工作原理：

数据从源容器中发出后，经由所在主机的docker0虚拟网卡转发到flannel0虚拟网卡（先可以不经过docker0网卡，使用cni模式），这是个P2P的虚拟网卡，flanneld服务监听在网卡的另外一端。

Flannel通过Etcd服务维护了一张节点间的路由表，详细记录了各节点子网网段 。

源主机的flanneld服务将原本的数据内容UDP封装后根据自己的路由表投递给目的节点的flanneld服务，数据到达以后被解包，然后直接进入目的节点的flannel0虚拟网卡，然后被转发到目的主机的docker0虚拟网卡，最后就像本机容器通信一下的有docker0路由到达目标容器。

flannel在进行路由转发的基础上进行了封包解包的操作，这样浪费了CPU的计算资源。

二、Worker节点宕机，简述Pods驱逐流程。

1、概述：

在 Kubernetes 集群中，当节点由于某些原因（网络、宕机等）不能正常工作时会被认定为不可用状态（Unknown 或者 False 状态），当时间超过了 pod-eviction-timeout 值时，那么节点上的所有 Pod 都会被节点控制器计划删除。

2、Kubernetes 集群中有一个节点生命周期控制器：node\_lifecycle\_controller.go。它会与每一个节点上的 kubelet 进行通信，以收集各个节点已经节点上容器的相关状态信息。当超出一定时间后不能与 kubelet 通信，那么就会标记该节点为 Unknown 状态。并且节点生命周期控制器会自动创建代表状况的污点，用于防止调度器调度 pod 到该节点。

3、那么 Unknown 状态的节点上已经运行的 pod 会怎么处理呢？节点上的所有 Pod 都会被污点管理器（taint\_manager.go）计划删除。而在节点被认定为不可用状态到删除节点上的 Pod 之间是有一段时间的，这段时间被称为容忍度。你可以通过下面的方式来配置容忍度的时间长短：

   tolerations:

      - key: node.kubernetes.io/not-ready

        operator: Exists

        effect: NoExecute

        tolerationSeconds: 180

      - key: node.kubernetes.io/unreachable

        operator: Exists

        effect: NoExecute

        tolerationSeconds: 180

如果在不配置的情况下，Kubernetes 会自动给 Pod 添加一个 key 为 node.kubernetes.io/not-ready 的容忍度 并配置 tolerationSeconds=300，同样，Kubernetes 会给 Pod 添加一个 key 为 node.kubernetes.io/unreachable 的容忍度 并配置 tolerationSeconds=300。

4、当到了删除 Pod 时，污点管理器会创建污点标记事件，然后驱逐 pod 。这里需要注意的是由于已经不能与 kubelet 通信，所以该节点上的 Pod 在管理后台看到的是处于灰色标记，但是此时如果去获取 pod 的状态其实还是处于 Running 状态。每种类型的资源都有相应的资源控制器（Controller），例如：deployment\_controller.go、stateful\_set\_control.go。每种控制器都在监听资源变化，从而做出相应的动作执行。deployment 控制器在监听到 Pod 被驱逐后会创建一个新的 Pod 出来，但是 Statefulset 控制器并不会创建出新的 Pod，原因是因为它可能会违反 StatefulSet 固有的至多一个的语义，可能出现具有相同身份的多个成员，这将可能是灾难性的，并且可能导致数据丢失。

三、简述你知道的K8s中几种Controller控制器，并详述其工作原理。简述 ingress-controller 的工作机制

1、deployment：适合无状态的服务部署

适合部署无状态的应用服务，用来管理pod和replicaset，具有上线部署、副本设定、滚动更新、回滚等功能，还可提供声明式更新，例如只更新一个新的Image

编写yaml文件，并创建nginx服务pod资源。

[root@master test]# vim nginx-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

  name: nginx-deployment

  labels:

    app: nginx

spec:

  replicas: 3

  selector:

    matchLabels:

      app: nginx

  template:

    metadata:

      labels:

        app: nginx

    spec:

      containers:

      - name: nginx1

        image: nginx:1.15.4

        ports:

        - containerPort: 80

查看控制器参数：可以使用describe或者edit两种方式

[root@master test]# kubectl describe deploy nginx-deployment

[root@master test]# kubectl edit deploy nginx-deployment/

2、StatefullSet：适合有状态的服务部署

适合部署有状态应用

解决Pod的独立生命周期，保持Pod启动顺序和唯一性

稳定，唯一的网络标识符，持久存储（例如：etcd配置文件，节点地址发生变化，将无法使用）

有序，优雅的部署和扩展、删除和终止（例如：mysql主从关系，先启动主，再启动从）

有序，滚动更新

应用场景：例如数据库

无状态服务的特点：

deployment 认为所有的pod都是一样的，不用考虑顺序的要求

不用考虑在哪个node节点上运行

可以随意扩容和缩容

有状态服务的特点：

实例之间有差别，每个实例都有自己的独特性，元数据不同，例如etcd，zookeeper

实例之间不对等的关系，以及依靠外部存储的应用

常规的service服务和无头服务的区别

service：一组Pod访问策略，提供cluster-IP群集之间通讯，还提供负载均衡和服务发现

Headless service 无头服务，不需要cluster-IP，直接绑定具体的Pod的IP，无头服务经常用于statefulset的有状态部署

创建无头服务的service资源和dns资源

由于有状态服务的IP地址是动态的，所以使用无头服务的时候要绑定dns服务

3、DaemonSet：一次部署，所有的node节点都会部署，例如一些典型的应用场景：

运行集群存储 daemon，例如在每个Node上运行 glusterd、ceph

在每个Node上运行日志收集 daemon，例如 fluentd、 logstash

在每个Node上运行监控 daemon，例如 Prometheus Node Exporter

在每一个Node上运行一个Pod

新加入的Node也同样会自动运行一个Pod

应用场景：监控，分布式存储，日志收集等

4、Job：一次性的执行任务

一次性执行任务，类似Linux中的job

应用场景：如离线数据处理，视频解码等业务

5、Cronjob：周期性的执行任务

周期性任务，像Linux的Crontab一样

应用场景：如通知，备份等

使用cronjob要慎重，用完之后要删掉，不然会占用很多资源

6、ingress-controller的工作机制

诞生

通常情况下，service和pod的IP仅可在集群内部访问

k8s提供了service方式：NodePort 来提供对外的服务，外部的服务可以通过访问Node节点ip+NodePort端口来访问集群内部的资源，外部的请求先到达service所选中的节点上，然后负载均衡到每一个节点上.

NodePort虽然提供了对外的方式但也有很大弊端：

由于service的实现方式：user\_space 、iptebles、 3 ipvs、方式这三种方式只支持在4层协议通信，不支持7层协议，因此NodePort不能代理https服务.

NodePort 需要暴露service所属每个node节点上端口，当需求越来越多，端口数量过多，导致维护成本过高，并且集群不好管理。

原理

Ingress也是Kubernetes API的标准资源类型之一，它其实就是一组基于DNS名称（host）或URL路径把请求转发到指定的Service资源的规则。用于将集群外部的请求流量转发到集群内部完成的服务发布。我们需要明白的是，Ingress资源自身不能进行“流量穿透”，仅仅是一组规则的集合，这些集合规则还需要其他功能的辅助，比如监听某套接字，然后根据这些规则的匹配进行路由转发，这些能够为Ingress资源监听套接字并将流量转发的组件就是Ingress Controller。

Ingress 控制器不同于Deployment 等pod控制器的是，Ingress控制器不直接运行为kube-controller-manager的一部分，它仅仅是Kubernetes集群的一个附件，类似于CoreDNS，需要在集群上单独部署。

ingress controller通过监视api server获取相关ingress、service、endpoint、secret、node、configmap对象，并在程序内部不断循环监视相关service是否有新的endpoints变化，一旦发生变化则自动更新nginx.conf模板配置并产生新的配置文件进行reload

四、简述k8s的调度机制

1、Scheduler工作原理：

请求及Scheduler调度步骤：

节点预选(Predicate)：排除完全不满足条件的节点，如内存大小，端口等条件不满足。

节点优先级排序(Priority)：根据优先级选出最佳节点

节点择优(Select)：根据优先级选定节点

2、具体步骤：

首先用户通过 Kubernetes 客户端 Kubectl 提交创建 Pod 的 Yaml 的文件，向Kubernetes 系统发起资源请求，该资源请求被提交到

Kubernetes 系统中，用户通过命令行工具 Kubectl 向 Kubernetes 集群即 APIServer 用 的方式发送“POST”请求，即创建 Pod 的请求。

APIServer 接收到请求后把创建 Pod 的信息存储到 Etcd 中，从集群运行那一刻起，资源调度系统 Scheduler 就会定时去监控 APIServer

通过 APIServer 得到创建 Pod 的信息，Scheduler 采用 watch 机制，一旦 Etcd 存储 Pod 信息成功便会立即通知APIServer，

APIServer会立即把Pod创建的消息通知Scheduler，Scheduler发现 Pod 的属性中 Dest Node 为空时（Dest Node=””）便会立即触发调度流程进行调度。

而这一个创建Pod对象，在调度的过程当中有3个阶段：节点预选、节点优选、节点选定，从而筛选出最佳的节点

节点预选：基于一系列的预选规则对每个节点进行检查，将那些不符合条件的节点过滤，从而完成节点的预选

节点优选：对预选出的节点进行优先级排序，以便选出最合适运行Pod对象的节点

节点选定：从优先级排序结果中挑选出优先级最高的节点运行Pod，当这类节点多于1个时，则进行随机选择

3、k8s的调用工作方式

Kubernetes调度器作为集群的大脑，在如何提高集群的资源利用率、保证集群中服务的稳定运行中也会变得越来越重要Kubernetes的资源分为两种属性。

可压缩资源（例如CPU循环，Disk I/O带宽）都是可以被限制和被回收的，对于一个Pod来说可以降低这些资源的使用量而不去杀掉Pod。

不可压缩资源（例如内存、硬盘空间）一般来说不杀掉Pod就没法回收。未来Kubernetes会加入更多资源，如网络带宽，存储IOPS的支持。

五、简述kube-proxy的三种工作模式和原理

1、userspace 模式

该模式下kube-proxy会为每一个Service创建一个监听端口。发向Cluster IP的请求被Iptables规则重定向到Kube-proxy监听的端口上，Kube-proxy根据LB算法选择一个提供服务的Pod并和其建立链接，以将请求转发到Pod上。

该模式下，Kube-proxy充当了一个四层Load balancer的角色。由于kube-proxy运行在userspace中，在进行转发处理时会增加两次内核和用户空间之间的数据拷贝，效率较另外两种模式低一些；好处是当后端的Pod不可用时，kube-proxy可以重试其他Pod。

2、iptables 模式

为了避免增加内核和用户空间的数据拷贝操作，提高转发效率，Kube-proxy提供了iptables模式。在该模式下，Kube-proxy为service后端的每个Pod创建对应的iptables规则，直接将发向Cluster IP的请求重定向到一个Pod IP。

该模式下Kube-proxy不承担四层代理的角色，只负责创建iptables规则。该模式的优点是较userspace模式效率更高，但不能提供灵活的LB策略，当后端Pod不可用时也无法进行重试。

3、该模式和iptables类似，kube-proxy监控Pod的变化并创建相应的ipvs rules。ipvs也是在kernel模式下通过netfilter实现的，但采用了hash table来存储规则，因此在规则较多的情况下，Ipvs相对iptables转发效率更高。除此以外，ipvs支持更多的LB算法。如果要设置kube-proxy为ipvs模式，必须在操作系统中安装IPVS内核模块。

六、k8s每个Pod中有一个特殊的Pause容器，能否去除，简述原因

pause container作为pod里其他所有container的parent container，主要有两个职责：

是pod里其他容器共享Linux namespace的基础

扮演PID 1的角色，负责处理僵尸进程

在Linux里，当父进程fork一个新进程时，子进程会从父进程继承namespace。目前Linux实现了六种类型的namespace，每一个namespace是包装了一些全局系统资源的抽象集合，这一抽象集合使得在进程的命名空间中可以看到全局系统资源。命名空间的一个总体目标是支持轻量级虚拟化工具container的实现，container机制本身对外提供一组进程，这组进程自己会认为它们就是系统唯一存在的进程。

在Linux里，父进程fork的子进程会继承父进程的命名空间。与这种行为相反的一个系统命令就是unshare：

七、简述pod中readness和liveness的区别和各自应用场景

存活性探针（liveness probes）和就绪性探针（readiness probes）

用户通过 Liveness 探测可以告诉 Kubernetes 什么时候通过重启容器实现自愈；

Readiness 探测则是告诉 Kubernetes 什么时候可以将容器加入到 Service 负载均衡池中，对外提供服务。语法是一样的。

主要的探测方式支持http探测，执行命令探测，以及tcp探测。

1、执行命令探测：

kubelet是根据执行命令的退出码来决定是否探测成功。当执行命令的退出码为0时，认为执行成功，否则为执行失败。如果执行超时，则状态为Unknown。

2、http探测：

http探测是通过kubelet请求容器的指定url，并根据response来进行判断。

当返回的状态码在200到400(不含400)之间时，也就是状态码为2xx和3xx是，认为探测成功。否则认为失败

3、tcp探测

tcp探测是通过探测指定的端口。如果可以连接，则认为探测成功，否则认为失败。

探测失败的可能原因

执行命令探测失败的原因主要可能是容器未成功启动，或者执行命令失败。当然也可能docker或者docker-shim存在故障。

由于http和tcp都是从kubelet自node节点上发起的，向容器的ip进行探测。

所以探测失败的原因除了应用容器的问题外，还可能是从node到容器ip的网络不通。

八、Pod启动失败如何解决以及常见的原因有哪些

1、一般查看系统资源是否满足，然后就是查看pod日志看看原因

2、describe通过这个参数查看pod失败的原因

3、还有就是看组件日志，apiserver等组件日志，有没有异常

4、访问不到看看网络插件状态和日志，还有就是dns状态和日志

九、简述K8s中label的几种应用场景

一些Pod有Label（enter image description here）。一个Label是attach到Pod的一对键/值对，用来传递用户定义的属性。

比如，你可能创建了一个"tier"和“app”标签，通过Label（tier=frontend, app=myapp）来标记前端Pod容器，

使用Label（tier=backend, app=myapp）标记后台Pod。

然后可以使用Selectors选择带有特定Label的Pod，并且将Service或者Replication Controller应用到上面。

10.简述你知道的Jenkins Pipeline中脚本语法中的几个关键字

pipeline 是jenkins2.X 最核心的特性， 帮助jenkins 实现从CI 到 CD与 DevOps的转变

pipeline 提供一组可扩展的工具， 通过 pipeline domain specific language syntax 可以到达pipeline as code 目的

pipiline as code :  jenkinsfile 存储在项目的 源代码库

为什么要使用pipeline

1. 代码： pipeline 以代码的形式实现，通过被捡入源代码控制， 使团队能够编译，审查和迭代其cd流程

2 可连续性： jenkins 重启 或者中断后都不会影响pipeline job

3.停顿： pipeline 可以选择停止并等待人工输入或者批准，然后在继续pipeline运行

4.多功能：  pipeline 支持现实世界的复杂CD要求， 包括fork、join子进程，循环和并行执行工作的能力

5.可扩展： pipeline 插件支持其DSL的自动扩展以及其插件集成的多个选项。

块(Blocks{})

　　-　　由大括号括起来的语句： 如　Pipeline{}, Sections{}, parameters{}, script{}

章节(Sections)

　　-　　通常包括一个或者多个指令或步骤　如 agent，post，stages，steps

指令(Directives)

　　-　　environment, options, parameters, triggers, stage, tools, when

步骤(steps)

　　-　　执行脚本式pipeline, 如script{}

十一、Docker 的网络通信模式。

1、host 模式，使用 --net=host 指定。

和宿主机共用一个 Network Namespace。容器将不会虚拟出自己的网卡，配置自己的 IP 等，而是使用宿主机的 IP 和端口。

2、container 模式，使用 --net=container:NAMEorID 指定。

指定新创建的容器和已经存在的一个容器共享一个 Network Namespace，而不是和宿主机共享。

3、none 模式，使用 --net=none 指定。

告诉 docker 将新容器放到自己的网络堆栈中，但是不要配置它的网络。

4、bridge 模式，使用 --net=bridge 指定，默认设置。

bridge 模式是 Docker 默认的网络设置，此模式会为每一个容器分配 Network Namespace、设置 IP 等，并将一个主机上的 Docker 容器连接到一个虚拟网桥上。

十二、K8s有哪些重要组件，简述一下集群部署的流程。

1:k8s集群的安装

1.1 k8s集群的架构

master节点:etcd,api-server,scheduler,controller-manager

node节点:kubelet,kube-proxy

etcd的作用:数据库

api-server:核心服务

controller-manager: 控制器管理 rc

scheduler: 创建新pod,选择合适的节点

kubelet: 调用docker来创建容器

kube-proxy: 对外提供用户访问,对内提供一个负载均衡器

1.6 :所有节点配置flannel网络

跨节点容器间的通信

a:安装etcd

b:安装配置启动flannel

c:重启docker生效

1.7 :配置master为docker镜像私有仓库

a:速度快

b:保护隐私

2:什么是k8s,k8s有什么功能?

2.1 k8s的核心功能

自愈:

弹性伸缩:

服务自动发现和负载均衡

滚动升级和一键回滚

密码和配置文件管理

2.3 k8s的安装方式

yum

源码编译

二进制   生产使用

kubeadm  生产使用

minikube

2.4 k8s的应用场景

微服务:

更高的并发,更高的可用性,更快代码更新

缺点: 管理复杂度上升

docker--k8s--弹性伸缩

3:k8s常用的资源

3.1创建pod资源

k8s最小资源单位

pod资源至少包含两个容器,基础容器pod+业务容器

3.2 ReplicationController资源

保证指定数量的pod运行

pod和rc是通过标签来关联

rc滚动升级和一键回滚

部署流程：

修改IP地址、主机和host解析

master节点安装etcd

master节点安装kubernetes

node节点安装kubernetes

所有节点配置flannel网络

配置master为镜像仓库

十三、你所用的到的日志分析工具有哪些以及它们如何与K8s集群通讯。

背景：

k8s 上面会跑大量的pod/容器。 特别是 集群相关的控制面 容器， 业务容器，有时候我们要对容器日志进行 监控与分析处理，这时一套好用的日志系统就格外的重要。

而成熟的日志解决方案有哪些呢？ 以前的ELK ， k8s主推的基于云原生的EFK ，这里的F是 CNCF认证的子项目 fluentd ，

fluentd 是 ruby的项目 ，由于个人体验习惯与yaml配置，以及项目特点， 选择了filebeat ，所以 整套日志系统使用的技术包括（elastic search fidlebeat kibana）

官方项目地址：

https://github.com/elastic/beats/tree/master/filebeat

https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-getting-started.html

配置文件：

https://github.com/chenpfeisoo/EFK-logging-deployment

该配置文件中对于采集器有fluentd ，也有filebeat ，看个人喜好部署

注意事项：

对于filebeat 我们会在comfigmap中配置 对采集做配置

比如 多行显示：

官方推荐

      multiline.pattern: '^\['

      multiline.negate: true

      multiline.match: after

java应用：

  include\_lines: ['Caused by']

  multiline:

      pattern: '^\['

      negate:  true

      match:   after

十四、Kubelet 与 kubeproxy 作用。Kubeproxy 的三种代理模式和各自的原理以及它们的区别。

kubelet

kubelet 进程用于处理master 下发的任务, 管理pod 中的容器, 注册 自身所在的节点.

kube-proxy 运行机制解析

kube-proxy 本质上,类似一个反向代理. 我们可以把每个节点上运行的 kube-proxy 看作 service 的透明代理兼LB.

Service是k8s中资源的一种，也是k8s能够实现减少运维工作量，甚至免运维的关键点，我们公司的运维都要把服务搭在我们集群里，接触过的人应该都能体会到其方便之处。Service能将pod的变化屏蔽在集群内部，同时提供负载均衡的能力，自动将请求流量分布到后端的pod，这一功能的实现靠的就是kube-proxy的流量代理，一共有三种模式，userspace、iptables以及ipvs。

1、userspace

为每个service在node上打开一个随机端口（代理端口）

建立iptables规则，将clusterip的请求重定向到代理端口

到达代理端口（用户空间）的请求再由kubeproxy转发到后端pod。

这里为什么需要建iptables规则，因为kube-proxy 监听的端口在用户空间，所以需要一层 iptables 把访问服务的连接重定向给 kube-proxy 服务，这里就存在内核态到用户态的切换，代价很大，因此就有了iptables。

2、iptables

kube-proxy不再负责转发，数据包的走向完全由iptables规则决定，这样的过程不存在内核态到用户态的切换，效率明显会高很多。但是随着service的增加，iptables规则会不断增加，导致内核十分繁忙（等于在读一张很大的没建索引的表）。

3、ipvs

用ipset存储iptables规则，这样规则的数量就能够得到有效控制，而在查找时就类似hash表的查找。

十五、介绍一下你所用到的中间件，其中那些是部署到容器中的。

略

十六、Iptables 四个表五个链

raw表：确定是否对该数据包进行状态跟踪

mangle表：为数据包设置标记

nat表：修改数据包中的源、目标IP地址或端口

filter表：确定是否放行该数据包（过滤）

INPUT：处理入站数据包

OUTPUT：处理出站数据包

FORWARD：处理转发数据包

POSTROUTING链：在进行路由选择后处理数据包

PREROUTING链：在进行路由选择前处理数据包

十七、还有一些有关dashboard 和prometheus 的他们有时会穿插着问一下

grafana顶多再多两年 现在都往统一平台走呢 比如rancher作为目前最好用的k8s ui 工具 他怎么可能不做一个监控出来替代grafana？k8s 的原生监控就成熟的不得了了 并且以后dashboard都不用自己编辑。

推荐3大方向：云原生 混合云 SRE 都跟着google走

云原生概念： https://jimmysong.io/kubernetes-handbook/cloud-native/cloud-native-definition.html

先明确清楚你的监控需求：要监控的对象有哪些？机器数量和监控指标有多少？需要具备什么样的告警功能？

监控是一项长期建设的事情，一开始就想做一个 All In One 的监控解决方案，我觉得没有必要。从成本角度考虑，在初期直接使用开源的监控方案即可，先解决有无问题。

从系统成熟度上看，Zabbix 属于老牌的监控系统，资料多，功能全面且稳定，如果机器数量在几百台以内，不用太担心性能问题，另外，采用数据库分区、SSD 硬盘、Proxy 架构、Push 采集模式都可以提高监控性能。

Zabbix 在服务器监控方面占绝对优势，可以满足 90% 以上的监控场景，但是应用层的监控似乎并不擅长，比如要监控线程池的状态、某个内部接口的执行时间等，这种通常都要做侵入式埋点。相反，新一代的监控系统 Open-Falcon 和 Prometheus 在这一点做得很好。

从整体表现上来看，新一代监控系统也有明显的优势，比如：灵活的数据模型、更成熟的时序数据库、强大的告警功能，如果之前对 Zabbix 这种传统监控没有技术积累，建议使用 Open-Falcon 或者 Prometheus。

Open-Falcon 的核心优势在于数据分片功能，能支撑更多的机器和监控项；Prometheus 则是容器监控方面的标配，有 Google 和 K8s 加持。

Zabbix、Open-Falcon 和 Prometheus 都支持和 Grafana 做快速集成，想要美观且强大的可视化体验，可以和 Grafana 进行组合。

用合适的监控系统解决相应的问题即可，可以多套监控同时使用，这种在企业初期很常见。

到中后期，随着机器数据增加和个性化需求增多（比如希望统一监控平台、打通公司的 CMDB 和组织架构关系），往往需要二次开发或者通过监控系统提供的 API 做集成，从这点来看，Open-Falcon 或者 Prometheus 更合适。

如果非要自研，可以多研究下主流监控系统的架构方案，借鉴它们的优势。先明确清楚你的监控需求：要监控的对象有哪些？机器数量和监控指标有多少？需要具备什么样的告警功能？

监控是一项长期建设的事情，一开始就想做一个 All In One 的监控解决方案，我觉得没有必要。从成本角度考虑，在初期直接使用开源的监控方案即可，先解决有无问题。

从系统成熟度上看，Zabbix 属于老牌的监控系统，资料多，功能全面且稳定，如果机器数量在几百台以内，不用太担心性能问题，另外，采用数据库分区、SSD 硬盘、Proxy 架构、Push 采集模式都可以提高监控性能。

Zabbix 在服务器监控方面占绝对优势，可以满足 90% 以上的监控场景，但是应用层的监控似乎并不擅长，比如要监控线程池的状态、某个内部接口的执行时间等，这种通常都要做侵入式埋点。相反，新一代的监控系统 Open-Falcon 和 Prometheus 在这一点做得很好。

从整体表现上来看，新一代监控系统也有明显的优势，比如：灵活的数据模型、更成熟的时序数据库、强大的告警功能，如果之前对 Zabbix 这种传统监控没有技术积累，建议使用 Open-Falcon 或者 Prometheus。

Open-Falcon 的核心优势在于数据分片功能，能支撑更多的机器和监控项；Prometheus 则是容器监控方面的标配，有 Google 和 K8s 加持。

Zabbix、Open-Falcon 和 Prometheus 都支持和 Grafana 做快速集成，想要美观且强大的可视化体验，可以和 Grafana 进行组合。

用合适的监控系统解决相应的问题即可，可以多套监控同时使用，这种在企业初期很常见。

到中后期，随着机器数据增加和个性化需求增多（比如希望统一监控平台、打通公司的 CMDB 和组织架构关系），往往需要二次开发或者通过监控系统提供的 API 做集成，从这点来看，Open-Falcon 或者 Prometheus 更合适。

如果非要自研，可以多研究下主流监控系统的架构方案，借鉴它们的优势。

-----------------------------------

©著作权归作者所有：来自51CTO博客作者一盏烛光的原创作品，请联系作者获取转载授权，否则将追究法律责任

灵雀云k8s面试题剖析

<https://blog.51cto.com/yw666/2550027>

# 90个常见的Kubernete 题，值得收藏学习

<https://zhuanlan.zhihu.com/p/428243855>

<https://www.cnblogs.com/zrxuexi/p/15320933.html>

<https://xuxinkun.github.io/2018/07/23/k8s-interview/>

这几个月参与了几场面试，设计了多道面试题，觉得可以综合考察应聘人对kubernetes的掌握情况。在这里分享下，供应聘人自查以及其他面试官参考。

这些面试题的设计初衷并不是考察kubernetes的使用。这种笔者认为较为流于表面，因为这些使用大多可以通过查看文档获得。笔者更多考察的是对于kubernetes的理解，包括对其架构、设计及一些相应原理的认识，以及对一些实践经验和技术视野的考察。后续有想到更好的题目，直接在此篇中持续更新。

## **基础篇**

基础篇主要面向的初级、中级开发工程师职位，主要考察对k8s本身的理解。

* kubernetes包含几个组件。各个组件的功能是什么。组件之间是如何交互的。
* k8s的pause容器有什么用。是否可以去掉。
* k8s中的pod内几个容器之间的关系是什么。
* 一个经典pod的完整生命周期。
* k8s的service和ep是如何关联和相互影响的。
* 详述kube-proxy原理，一个请求是如何经过层层转发落到某个pod上的整个过程。请求可能来自pod也可能来自外部。
* rc/rs功能是怎么实现的。详述从API接收到一个创建rc/rs的请求，到最终在节点上创建pod的全过程，尽可能详细。另外，当一个pod失效时，kubernetes是如何发现并重启另一个pod的？
* deployment/rs有什么区别。其使用方式、使用条件和原理是什么。
* cgroup中的cpu有哪几种限制方式。k8s是如何使用实现request和limit的。

## **拓展实践篇**

拓展实践篇主要面向的高级开发工程师、架构师职位，主要考察实践经验和技术视野。

* 设想一个一千台物理机，上万规模的容器的kubernetes集群，请详述使用kubernetes时需要注意哪些问题？应该怎样解决？（提示可以从高可用，高性能等方向，覆盖到从镜像中心到kubernetes各个组件等）
* 设想kubernetes集群管理从一千台节点到五千台节点，可能会遇到什么样的瓶颈。应该如何解决。
* kubernetes的运营中有哪些注意的要点。
* 集群发生雪崩的条件，以及预防手段。
* 设计一种可以替代kube-proxy的实现
* sidecar的设计模式如何在k8s中进行应用。有什么意义。
* 灰度发布是什么。如何使用k8s现有的资源实现灰度发布。
* 介绍k8s实践中踩过的比较大的一个坑和解决方式。