云原生面试专题-02

1 云原生是什么？？

1.1 工作角度聊聊云原生

从职业发展反向：

1）从事云原生基础架构开发： 必须熟悉 kubernetes, docker 技术体系；深刻了解其运作原理；

2）从事业务开发：必须了解 kubernetes 架构体系所构建的 devops 一套运作体系；kubernetes 成为业务开发的基础技能；

1.2 什么是云原生？

云原生（Cloud Native）技术：

1）、以 kubernetes 为核心的架构体系，就把这种架构的服务体系叫做云原生技术；

2）、软件架构思想，遵循 kubernetes 云原生架构设计实现，这种架构就叫做云原生架构；

通俗易懂的方式解释：

1）、架构维度： 软件架构设计思想

2）、应用角度： 让应用程序（应用服务：消息系统，营销系统，订单系统....，应用软件: mysql,es,rocketmq.....）都必须运行在容器中（docker 容器），然后使用 kubernetes 容器编排技术进行容器管理，这样的架构就叫做云原生架构；

云原生特点（表现形式）：

1）、容器化：所有的服务都必须运行在容器中；（轻量级—基于操作系统虚拟化技术）

2）、微服务：把服务拆分的尽肯能小，小而专注，功能内聚，微服务更适合在云原生环境中部署，以便于实现云原生架构；

3）、DevOps：企业实现自动化，数字化开发的体系，一种敏捷的开发思想，让开发以一种更加高效的组织形式实现代码的流水线的生产模式；实现产品的高效率的交付；

4）、CI/CD：可持续交付，可持续部署

云原生还解决了服务限流，降级，熔断等等这些问题？---- ServiceMesh 服务网格架构

ServiceMesh – 云原生技术

云原生技术使用在项目中后： 大大减轻运维压力，减轻程序员上线测试部署的压力，可以实现代码的高效交付；

1.3 云原生技术优势

以 kubernetes 为核心的原生技术体系正在成为企业的开发，服务上线，服务测试的标准；越来越多的企业纷纷把服务迁移到 kubernetes 为核心的云原生体系中；

无论是传统企业 IT 研发部分，还是互联网企业都在做数字化转型（提高研发效率，开发效率），云原生就是企业数字化（自动化，智能化）转型的唯一路径；实现企业研发效率的倍增；

在企业中：kubernetes 和 git 地位是一样的，是企业必备的基础技能；

技术发展趋势： kubernetes 为核心云原生技术必然会成为企业开发的主流技术，有了这一套技术后，实现在家办公， 实现开发效率提升，为企业实现降本增效；

技术发展方向：

1)、阿里达摩院发布 2021 年具有 10 大颠覆性科技： 云原生技术---颠覆的是 IT 开发行业

2)、云原生技术，实现了任何语言都可以挂平台的能力

Java 一大优势：具有挂平台的能力

云原生技术后：Java,Go,Ruby,PHP,Python……..

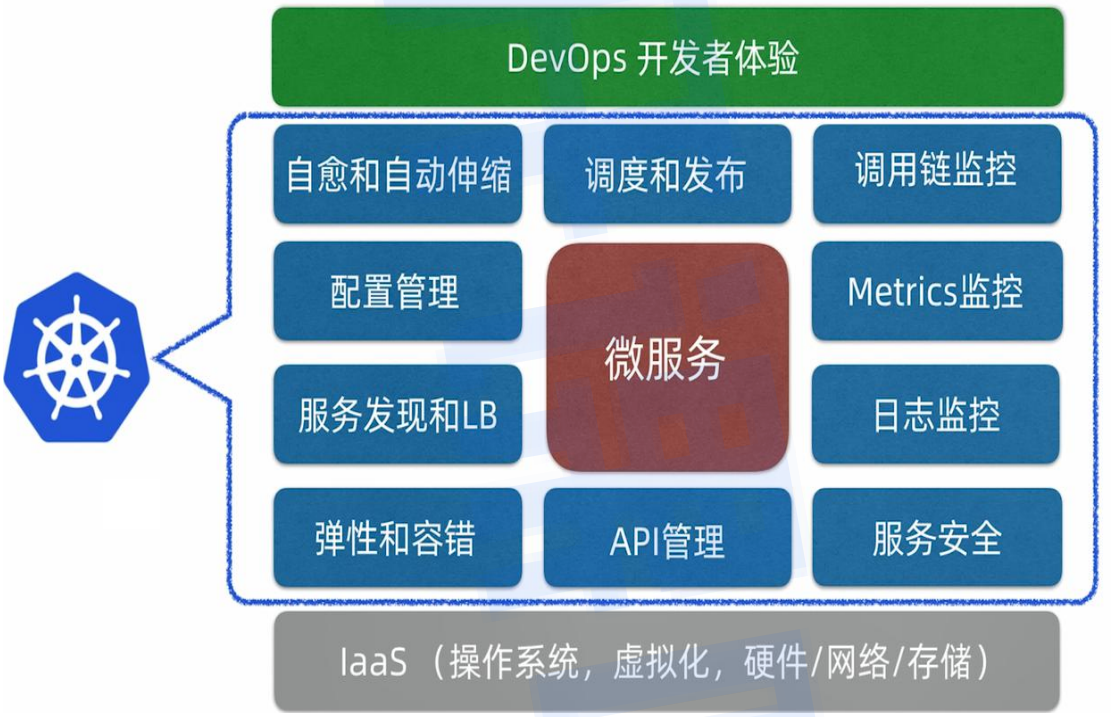
Java 唯一优势：庞大的生态体系，庞大的用户群体，缺点很明显： 占用内存大，性能低；JDK16 全面拥抱云原生，Spring 框架也在拥抱云原生技术；

2 Kubernetes 技术

2.1 轻装上阵

一旦使用了 kubernetes 技术，不需要关心那些和项目业务开发没有关系的代码实现（服务治理，服务发现，负载均衡，服务容错，服务探测.....），只需要关心项目业务开发即可；

因此有 kubernetes 云原生技术后，企业只需要一个小而精悍的团队即可实现业务开发；



2.2 全面的拥抱微服务

思考：小企业做项目开发时候，想要使用微服务架构？？ 面临的困难是什么？？

1）服务监控（日志监控，接口监控）

2）故障追踪（调用链路变长，增加寻找 bug 困难）

3）服务治理（降级，限流，熔断）

4）配置管理，服务安全，负载均衡，服务发现

现在：有了 kubernetes 把开发微服务的困难全部解决了，所有的企业都可以全面使用微服务架构开发项目；

一旦使用 kubernetes，就需要关心以上问题，只需要关心业务开发即可；

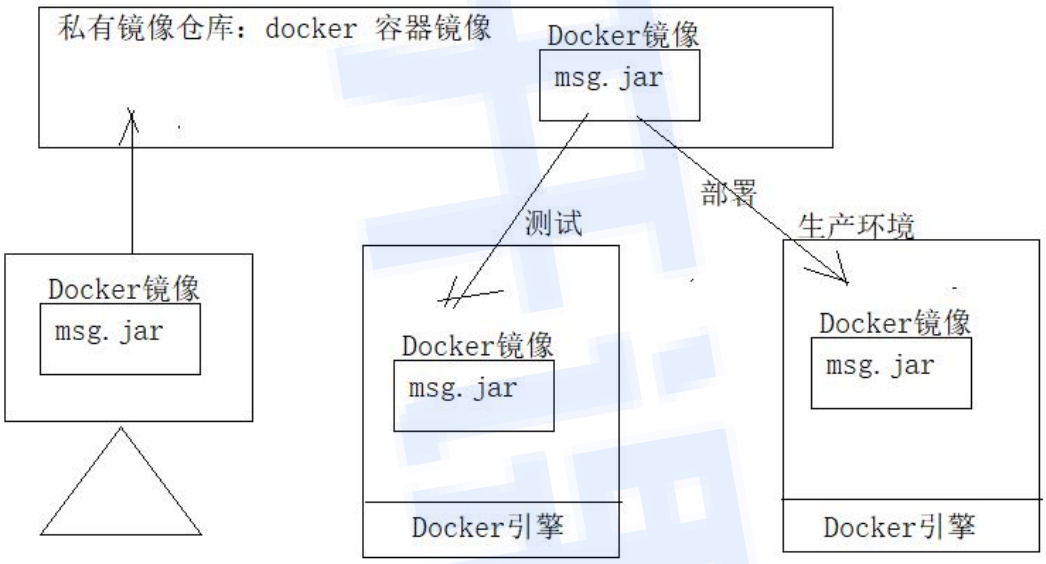
2.3 无缝迁移

项目开发： 开发环境 -> 发布版本 -> 测试 -> 发布版本 QA -> stable -> 上线

存在问题：

测试环境，生产环境不一致，造成项目上线出现问题

云原生（底层依赖容器化技术）：



Docker 容器运行在操作系统内部的独立的沙箱环境，不会受到宿主机操作系统的影响，因此这个容器可以实现无缝迁移；

2.4 弹性扩容

大规模服务上线后，可以实现服务弹性扩容缩容（自动化实现）：

促销活动：流量持续增加（事先预知： 活动预演，压测预案）

突发事件：西安一码通-突发流量-事先并不知道流量增加，根据突发流量动态扩容，缩容；

Kubernetes 可以实现动态的扩容缩容：

1）、根据 cpu 使用率进行动态扩容缩容

2）、根据内存使用率进行动态扩容缩容

3）、根据自定义的指标进行扩容缩容（QPS, TPS—阈值）

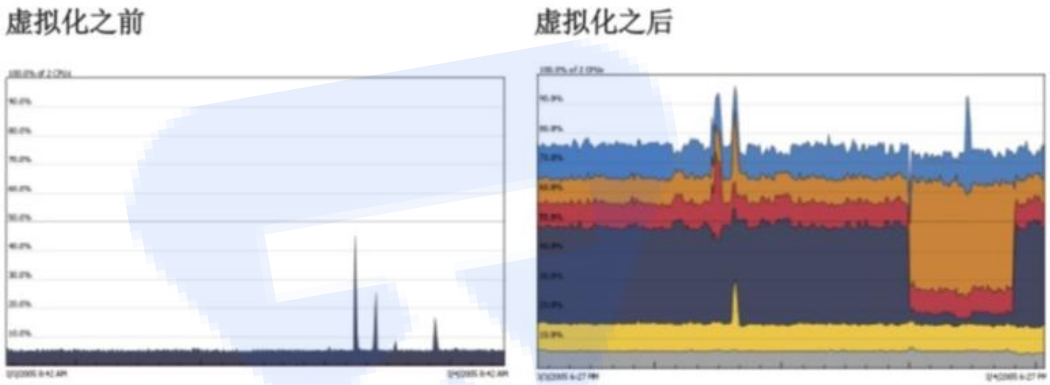
Kubernetes 实现自动化扩容，缩容非常简单，只需要配置 HPA 控制器即可：

```
Metrics:
- Resource:
    Name: cpu
    Target:
      averageUtilization: 70   # 按照 cpu 利用率 70%进行扩容
```

2.5 充分利用服务器资源

思考：物理机，虚拟机部署应用程序的时候，这些服务器 cpu，内存资源是否得到了充分的利用呢？？

以上图示，表示物理机节点部署的程序，在大部分时间，内存存在大量的闲置的资源；



在机器节点中部署一个应用，大部分资源都处理闲置状态，一旦使用虚拟化技术(docker 容器化技术)，可以在一个机器节点中部署多个应用的技术，可以看到机器节点资源被充分利用；

根据以上的资源利用情况，是否有好的解决办法更好的实现资源的充分利用呢？？

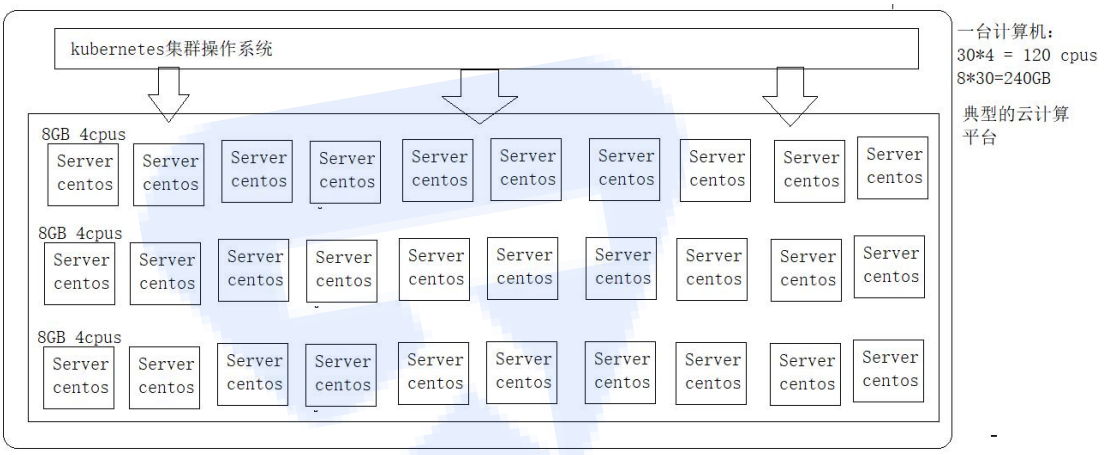
思考：Windows，mac 如何实现资源的充分利用的？？（在一台电脑把 cpu，内存资源分配给多个进程使用）？

答案：操作系统—调度系统（单机环境）：充分的调度底层的 cpu,内存资源，实现资源的最大化合理分配利用；

云原生环境（云计算环境-> 非常多的机器节点）--- 因此要想实现在云环境的资源的充分利用，那就必须开发一个跨节点的调度系统；

因此也把 kubernetes 叫做云原生的操作系统；其实就是跨节点的调度系统； kubernetes 可以跨节点管理一个规模庞大集群网络（windows,mac，linux 都是单机）,这个集群的网络也可以把他叫做云计算网络；

Kubernetes 管理一个庞大云计算网络，实现多个节点共同的计算；



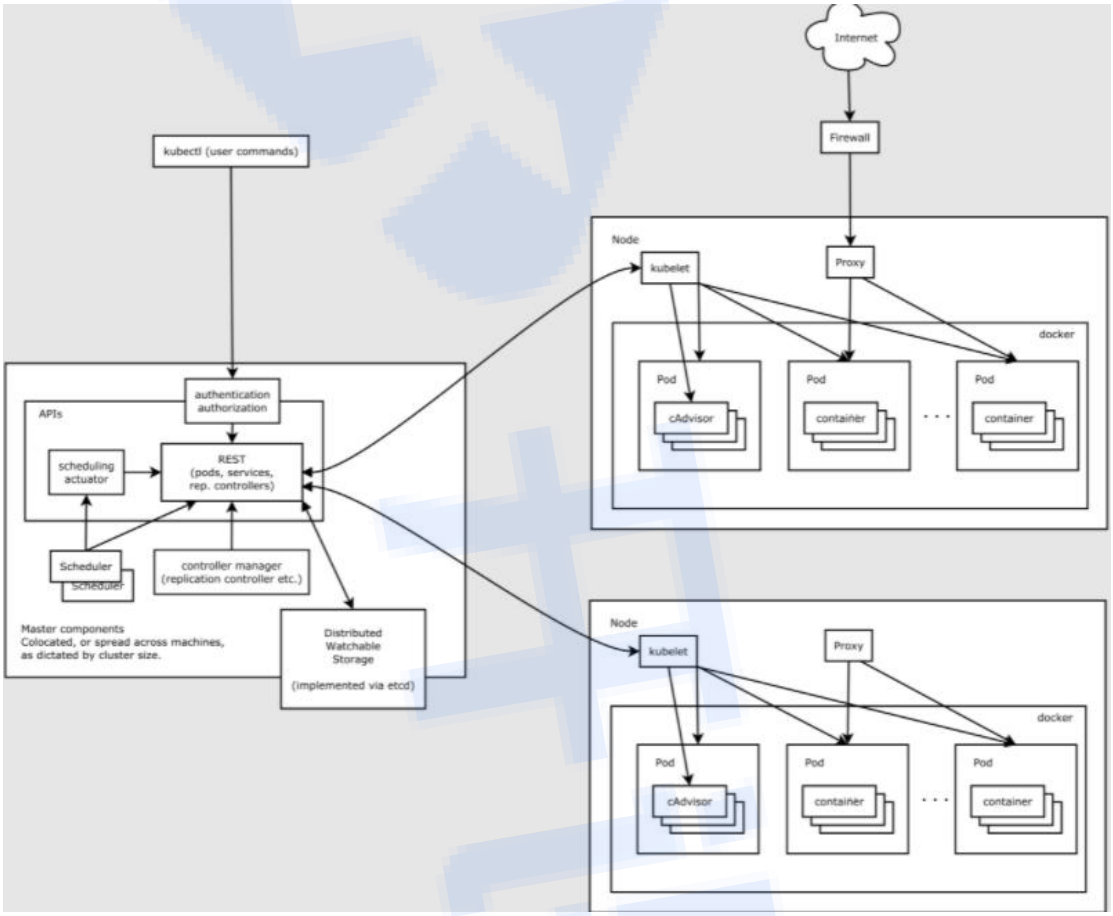
以 kubernetes 所构建的集群网络平台，叫做 pass 平台（容器云），可以实现服务调度，充分利用服务器资源，实现服务合理部署，使得服务以更好的组态实现运行；

2.6 自动化运维

以 kubernetes 为核心建立一套 DevOps 流水线的生产模式，实现服务的生产，服务自动化发布，服务自动化测试；大大减轻程序员的工作量，增加我们的生产力；节省开发时间，节省运维开销；

3 Kubernetes 架构

3.1 分布式架构



Kubernetes 是一个分布式架构的调度平台（大规模的集群网络），组成的结构：一个或多个 master,对应一群 node 节点；

Master 节点实现高可用部署，可以部署多个节点，实现节点之间的 keepalive;

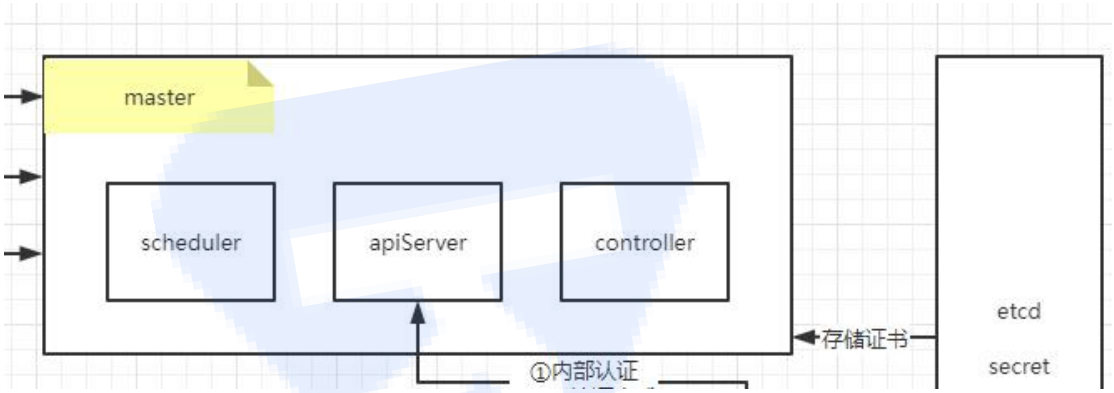
Node 节点：服务最终被部署在 node 节点；大规模的 node 集群组成了云计算节点，或云原生的节点；

Master 节点：负责调度；master 是整个集群的操作系统；

3.2 Master 节点

Master 节点包括 ApiServer, Scheduler , Controller , etcd , ApiServer 是整个系统的对外的接口，供客户端和其他组件使用；

Scheduler 负责对集群内部的资源进行调度，controller 负责管理控制器；



Master 核心组件：

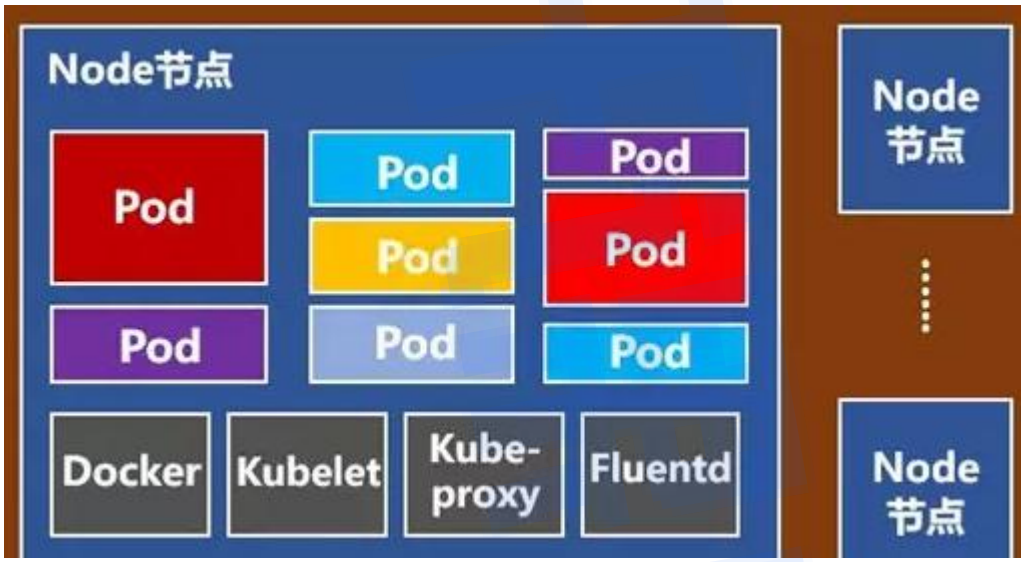
1）、ApiServer：服务网关，所有的服务请求都必须经过 apiserver 进行统一授权认证

2）、Scheduler：调度器，负责把服务调用到一个合适的 node 节点进行部署

3）、Controller：控制器，负责管理 kubernetes 资源对象；

4）、etcd: nosql 数据库，用来存储集群元数据（集群状态，属性信息,节点信息，pod 信息），同时 etcd 存储资源对象；

3.3 Node 节点



Docker: 基础引擎，容器化实现跨平台，每一个机器节点底层必须有 docker 容器引擎；

Kubelet:

1）服务就是和 master 建立连接的 node 节点服务进程，和 master 节点保持心跳，向 master 节点传输本地 node,pod 节点信息；

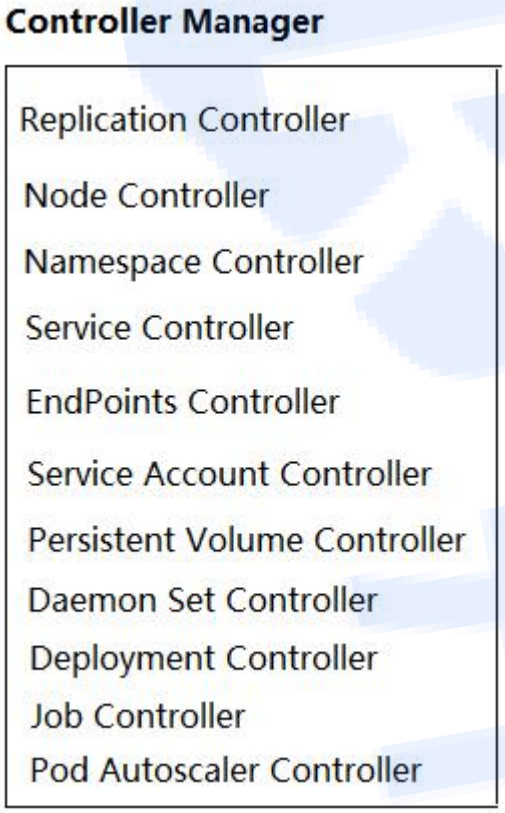
2）Scheduler 负责调度，把调度结果存储在 etcd, kubelet 实现 pod 在本地节点的部署

Kube-proxy: 负载均衡，服务发现

POD: kubernetes 管理的最小单元，pod 内部封装的是容器，容器内部封装是应用程序；

3.4 K8s 的控制器

Controller 控制器： 控制服务资源对象，管理 kubernetes 资源对象，实现资源对象 CRUD

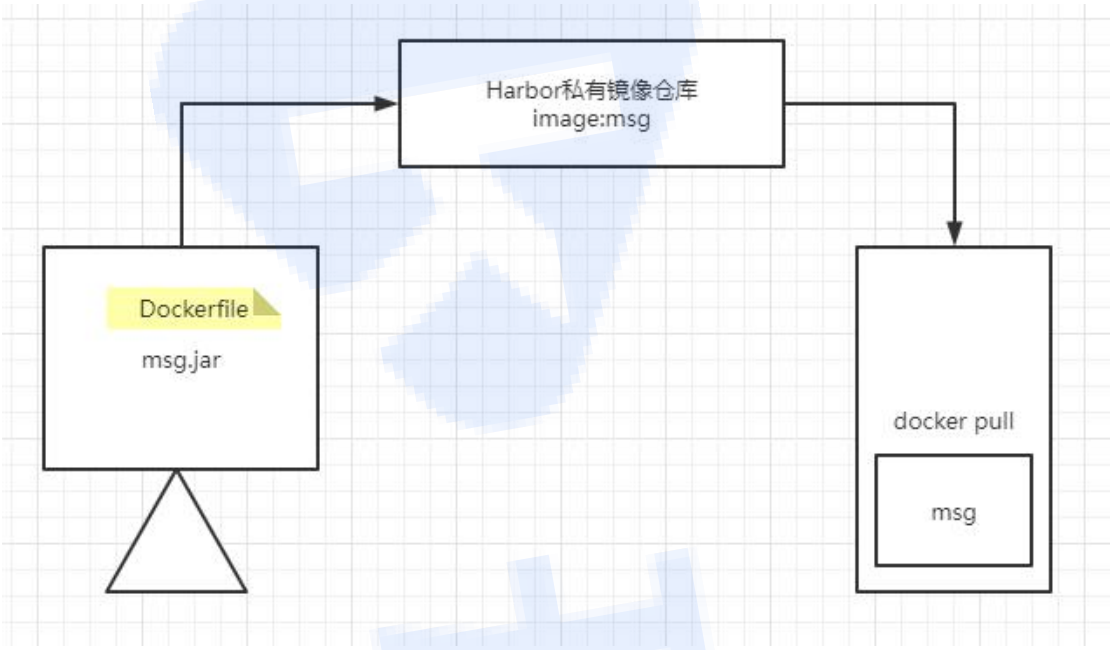


Replication Controller： 副本控制器，控制副本（集群节点）数量，当副本数量设置为 3，副本控制器就会保证副本数量始终为 3，如果如果服务宕机了，也会立马对服务进行重建；
Node Controller：节点控制器，管理 node 节点
Namespace Controller：命名空间控制，管理命名空间
Service Controller：虚拟服务控制器，主要实现服务发现，服务负载均衡，控制器管理 service
Endpoints Controller：维护 pod,service 之间关系的
Service Account Controller：本地服务访问安全控制
Persistent volume controller：持久化数据卷控制器
Deamon set Controller：保证每一个服务节点都部署一个相同的服务
Deployment Controller：管理无状态服务的部署
Job Controller：批处理任务控制器
Pod AutoScaler Controller：自动扩容控制器

3.5 部署流程

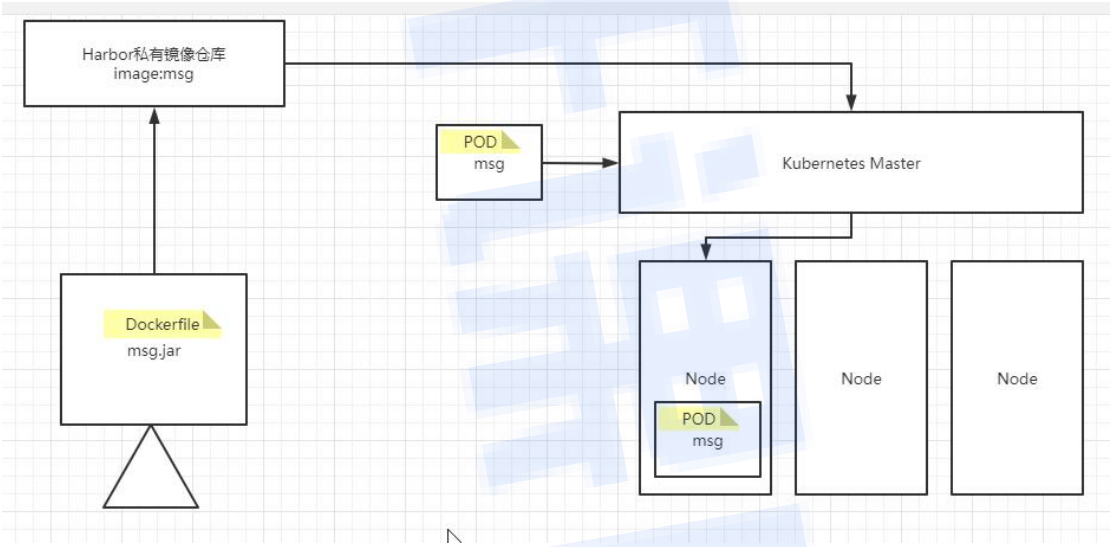
Docker 容器化部署：
直接把服务部署在 Docker 容器中即可；

1）、根据 dockerfile 构建镜像
2）、把镜像发布镜像仓库(harbor)
3)、拉取镜像实现部署

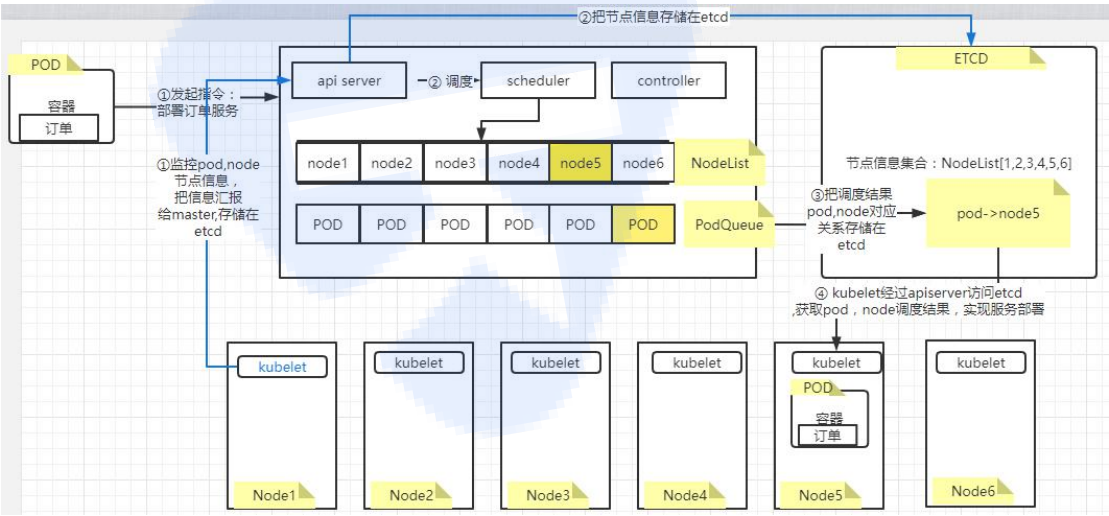


Kubernetes 服务部署：

不会直接部署容器，而是部署 POD 服务，POD 就是 k8s 服务部署的最小单位，POD 内部封装是容器；



3.6 调度流程



思考：scheduler 如何知道该把 pod 调度到那个节点进行部署呢？？

前提：Scheduler 在 master 节点中，因此 master 节点必须知道所有的 node 节点状态，然后才能实现调度；

1）每一个 node 节点都有一个 kubelet 进程服务，这个服务将会向 master 节点汇报每一个 node 节点状态信息，然后 master 把这个信息存储在 etcd 服务；

2）master 接受一个指令：部署一个订单服务指令（kubernetes 部署的最基本的单元是 pod,也就是说它部署的是一个 pod,实现 pod 的调度）

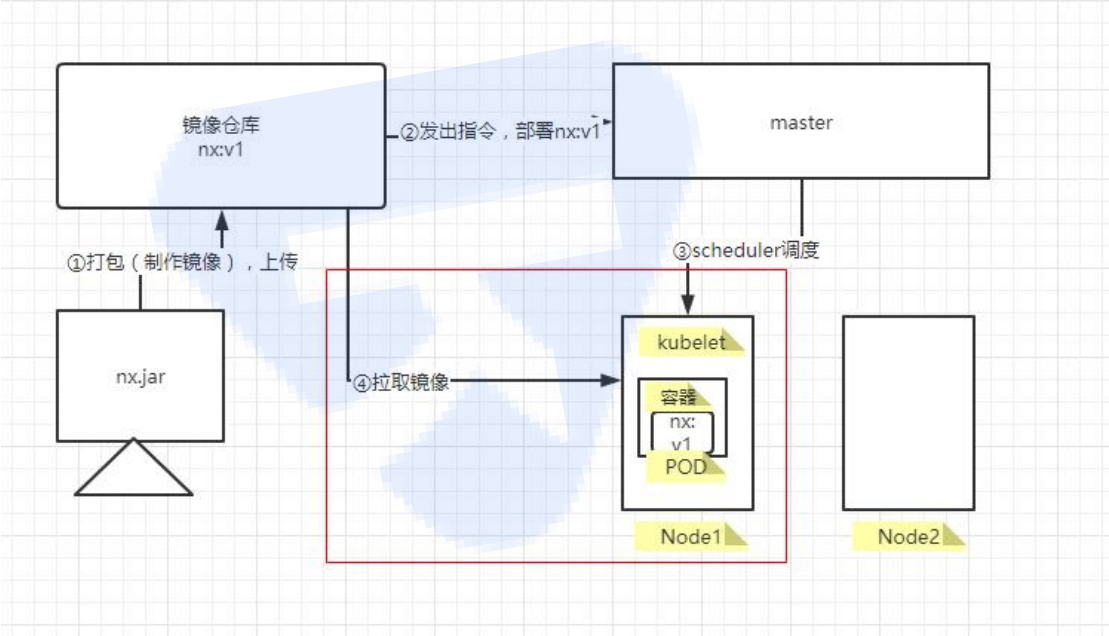
3）master 从 etcd 中获取 NodeList 集合（集合中存储的就是 node 节点信息对象），然后和 PodQueue(等待被调度的 pod 队列)进行匹配；匹配成功后把匹配的关系存储在 etcd，这个关系存储完成后，scheduler 使命完成了；

4）每一个 node 节点都有一个 kubelet 服务进程，这个服务将会去访问 etcd，获取调度的结果，判定是否属于本节点的调度结果；如果是就直接部署；

4 服务部署入门

4.1 部署流程

使用 kubernetes 部署一个应用基本流程：



Kubernetes 在服务服务的时候，拉取镜像的操作是在 node 节点完成的，当把 pod 调度到 node 节点时候，由 node 节点在部署应用的时候完成镜像拉取，实现服务部署；

4.2 应用部署

使用 kubernetes 完成服务部署：必须有一个 kubernetes 的环境；

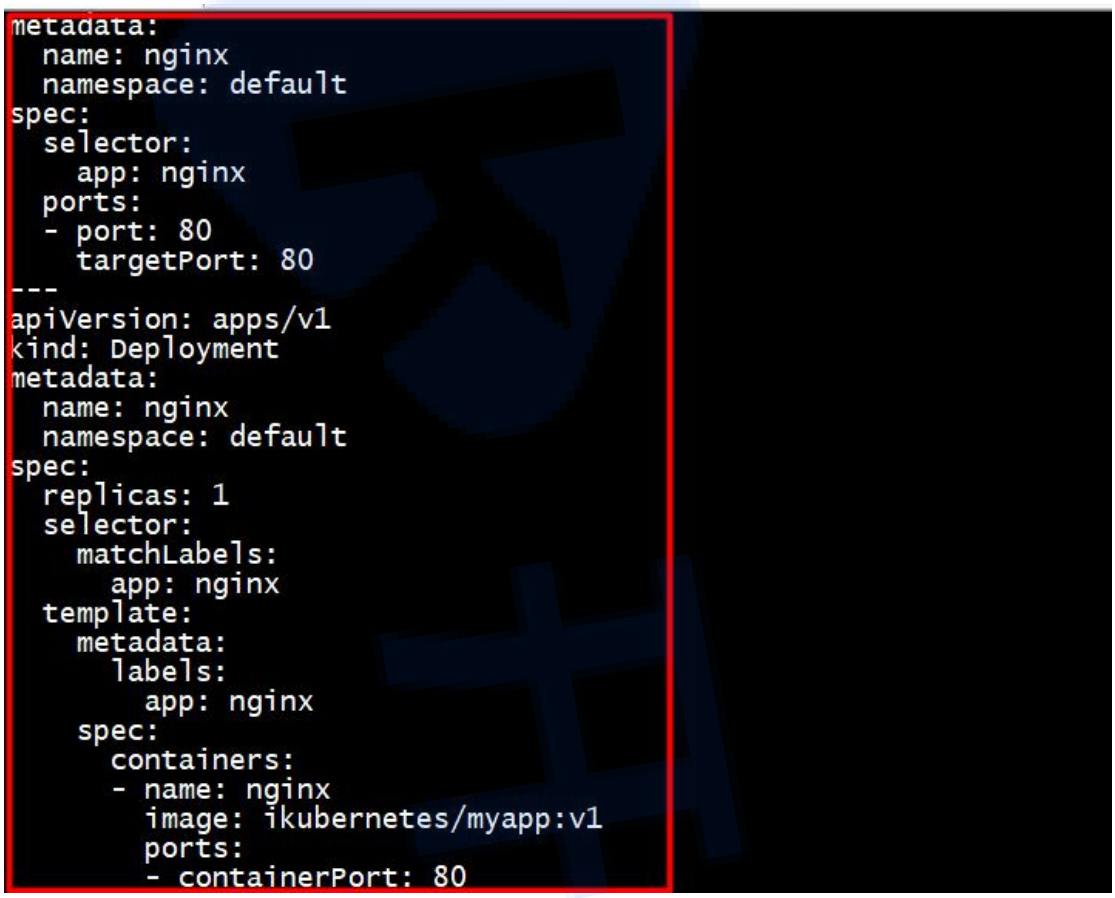
```
[root@k8s-master ~]# kubectl get node
NAME         STATUS   ROLES    AGE    VERSION
k8s-master   Ready    master   234d   v1.18.2
k8s-node1    Ready    <none>   234d   v1.18.2
k8s-node2    Ready    <none>   234d   v1.18.2
[root@k8s-master ~]#
```

部署一个应用：myapp 应用（内部就是 nginx）

部署指令： kubectl run 应用名称 –image=镜像地址 –port=容器端口

```
[root@k8s-master ~]# kubectl run my-app --image=ikubernetes/myapp:v1 --port=80
pod/my-app created
[root@k8s-master ~]# kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
deploy01-67d5c9f579-jnkmj     1/1     Running   1          72d
my-app                        1/1     Running   0          3s
web-0                         1/1     Running   1          72d
web-1                         1/1     Running   1          72d
web-2                         1/1     Running   1          72d
web-3                         1/1     Running   1          72d
[root@k8s-master ~]# kubectl get pod -o wide
NAME                          READY   STATUS    RESTARTS   AGE   IP              NODE
S
deploy01-67d5c9f579-jnkmj     1/1     Running   1          72d   10.244.2.233    k8s-node2
my-app                        1/1     Running   0          24s   10.244.1.136    k8s-node1
web-0                         1/1     Running   1          72d   10.244.2.234    k8s-node2
web-1                         1/1     Running   1          72d   10.244.1.133    k8s-node1
web-2                         1/1     Running   1          72d   10.244.2.237    k8s-node2
web-3                         1/1     Running   1          72d   10.244.1.131    k8s-node1
[root@k8s-master ~]# curl 10.244.1.136
Hello MyApp | Version: v1 | <a href="hostname.html">Pod Name</a>
[root@k8s-master ~]#
```

4.3 Yaml 部署



```
metadata:
  name: nginx
  namespace: default
spec:
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: ikubernetes/myapp:v1
        ports:
        - containerPort: 80
```

5 K8s 核心资源对象

5.1 副本控制器

资源对象名称：

Replication Controller（废弃），ReplicaSet(正在使用的副本控制器)

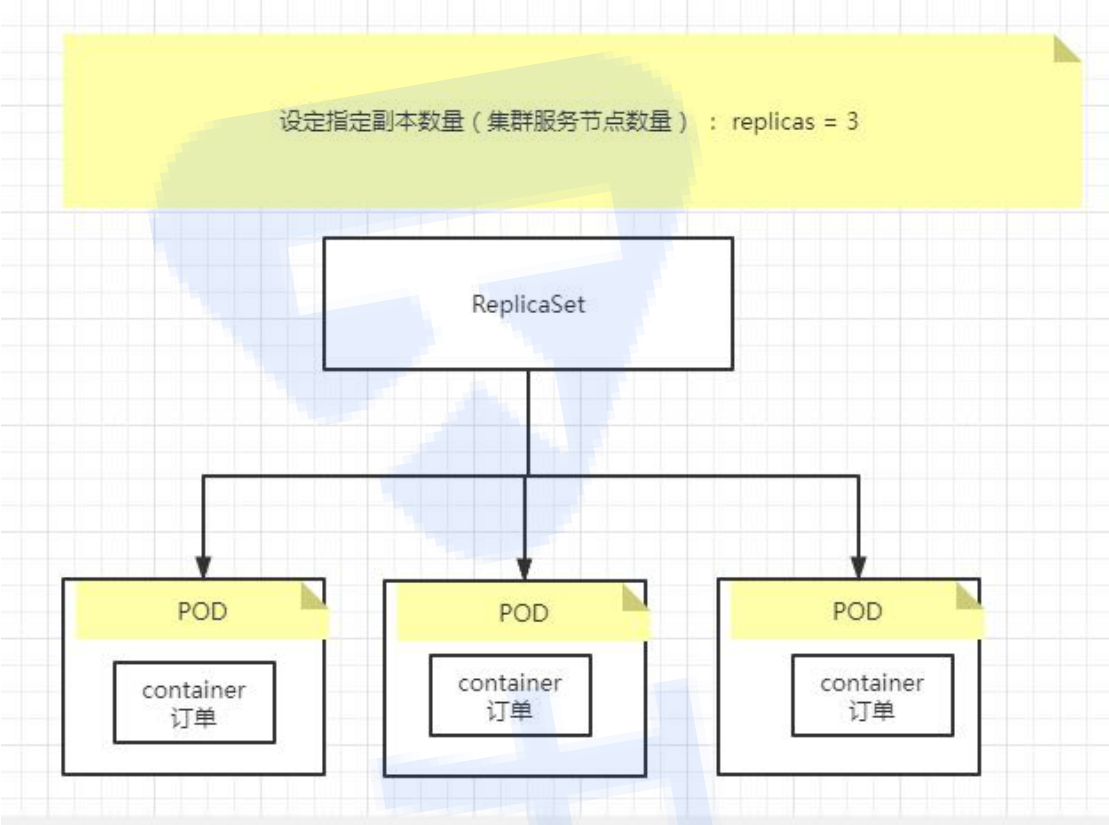
副本控制作用：

用来保证服务副本数量（集群节点数量）与预期所设定的数量永远保持一致性；也就是说部署集群的时候设定了集群数量为 3，副本控制将会永远保证副本集群节点为 3；

应用场景：

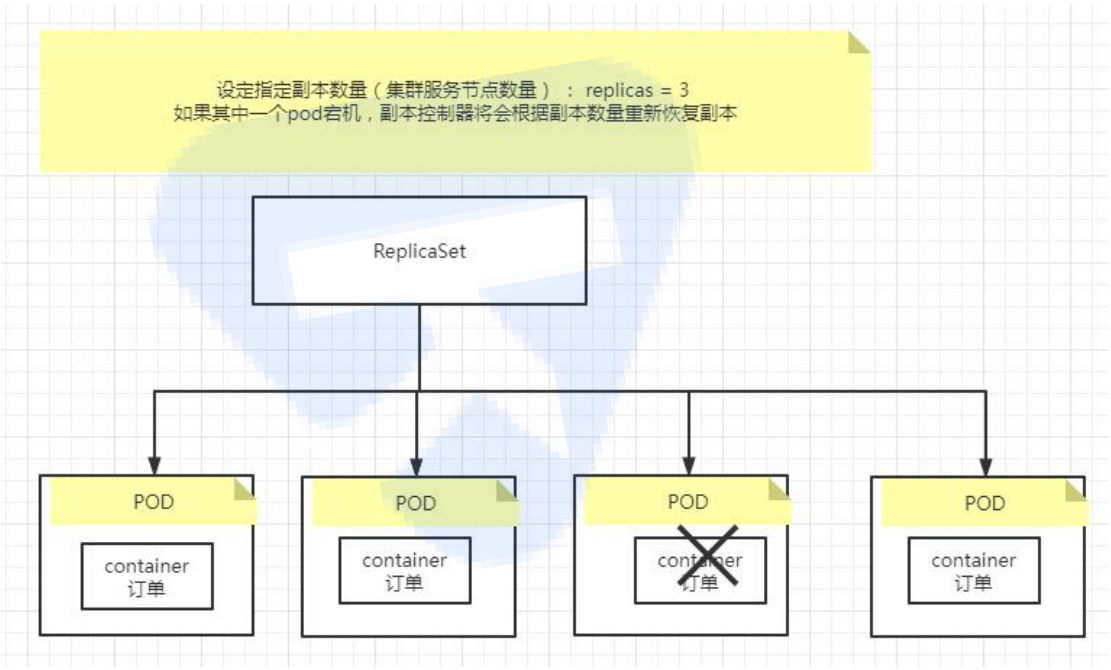
服务上线后，pod 宕机了，副本控制器就会对服务进行重建；保证服务高可用性；

1）副本控制器结构



2）pod 服务宕机（服务一旦宕机，副本控制器立即对服务进行重建）

```
[root@k8s-master deploy-server]# kubectl scale deployment nginx --replicas=3
deployment.apps/nginx scaled
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-56d786cd98-4ft7f   1/1     Running   0          3s
nginx-56d786cd98-d6cvz   1/1     Running   0          84s
nginx-56d786cd98-nzzwx   1/1     Running   0          3s
[root@k8s-master deploy-server]# kubectl delete pod nginx-56d786cd98-nzzwx
pod "nginx-56d786cd98-nzzwx" deleted
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-56d786cd98-4ft7f   1/1     Running   0          42s
nginx-56d786cd98-d6cvz   1/1     Running   0          2m3s
nginx-56d786cd98-fjlwf   1/1     Running   0          6s
[root@k8s-master deploy-server]#
```



思考题： 在企业中，每天部署的服务成千上万个业务服务（pod），那么你的副本控制器是如何知道你控制的是那几个 pod 呢？？
答案：通过 label 标签进行控制

5.2 Label 标签

标签用于区分对象（比如 Pod、Service），键/值对存在；每个对象可以有多个标签，通过标签关联对象。

Kubernetes 中任意 API 对象都是通过 Label 进行标识，Label 的实质是一系列的 Key/Value 键值对，其中 key 于 value 由用户自己指定。

Label 可以附加在各种资源对象上，如 Node、Pod、Service、RC 等，一个资源对象可以定义任意数量的 Label，同一个 Label 也可以被添加到任意数量的资源对象上去。

Label 是 Replication Controller 和 Service 运行的基础，二者通过 Label 来进行关联 Node 上运行的 Pod。

我们可以通过给指定的资源对象捆绑一个或者多个不同的 Label 来实现多维度的资源分组管理功能，以便于灵活、方便的进行资源分配、调度、配置等管理工作。
一些常用的 Label 如下：

版本标签："release":"stable","release":"canary"......

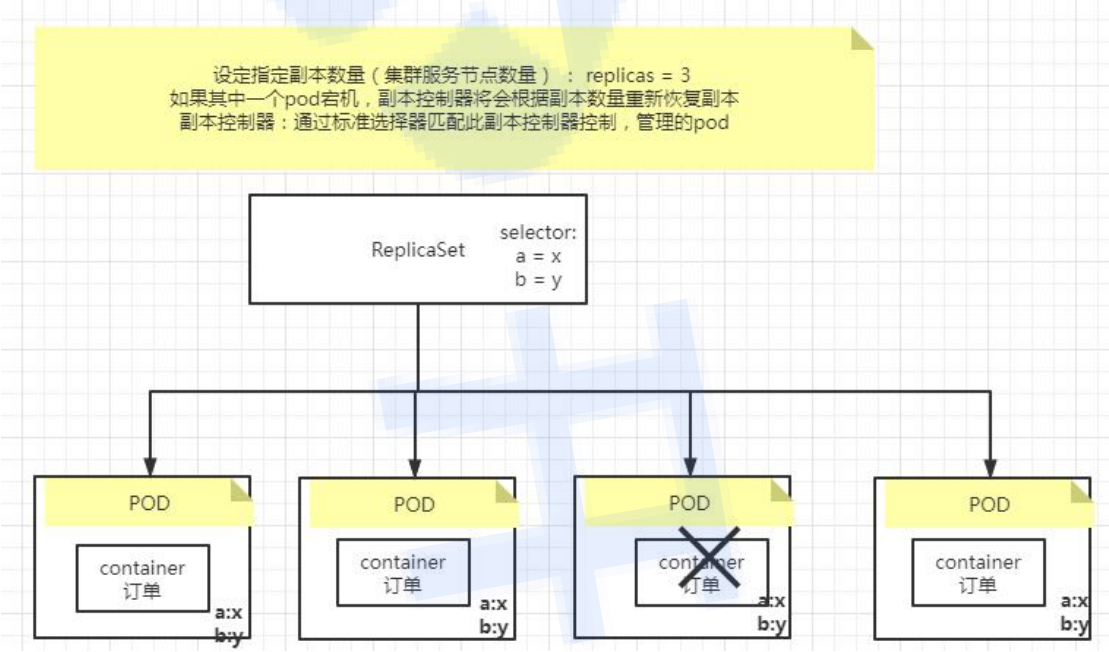
环境标签："environment":"dev","environment":"qa","environment":"production"

架构标签："tier":"frontend","tier":"backend","tier":"middleware"

分区标签："partition":"customerA","partition":"customerB"

质量管控标签："track":"daily","track":"weekly"

Label 相当于我们熟悉的标签，给某个资源对象定义一个 Label 就相当于给它大了一个标签，随后可以通过 Label Selector（标签选择器）查询和筛选拥有某些 Label 的资源对象，Kubernetes 通过这种方式实现了类似 SQL 的简单又通用的对象查询机制。



一般是一台机器(node)会有多个 pod,一个 pod 对应多个 docker,一个 docker 一般对应一个应用，是这样对应吗？

答：一个机器有多少个 pod 是有这个机器 cpu，内存计算能力所决定的； docker 是一个基础引擎（相当于 JVM）,运行 docker 容器，必须有 docker 引擎；pod 是 k8s 管理，但是 pod 内部封装的是容器；

5.3 Deployment

副本控制器管理 pod，已经实现了 pod 的管理控制；但是副本控制器不支持服务滚动更新，扩容缩容等等，如果需要实现以上的功能，就必须要应用 Deployment 资源对象；

Deployment 为 Pod 和 ReplicaSet 提供了一个 声明式定义方法，用来替代以前的 ReplicationController 来方便的管理应用。

典型的应用场景：

（1）、定义 Deployment 来创建 Pod 和 ReplicaSet

（2）、滚动升级和回滚应用

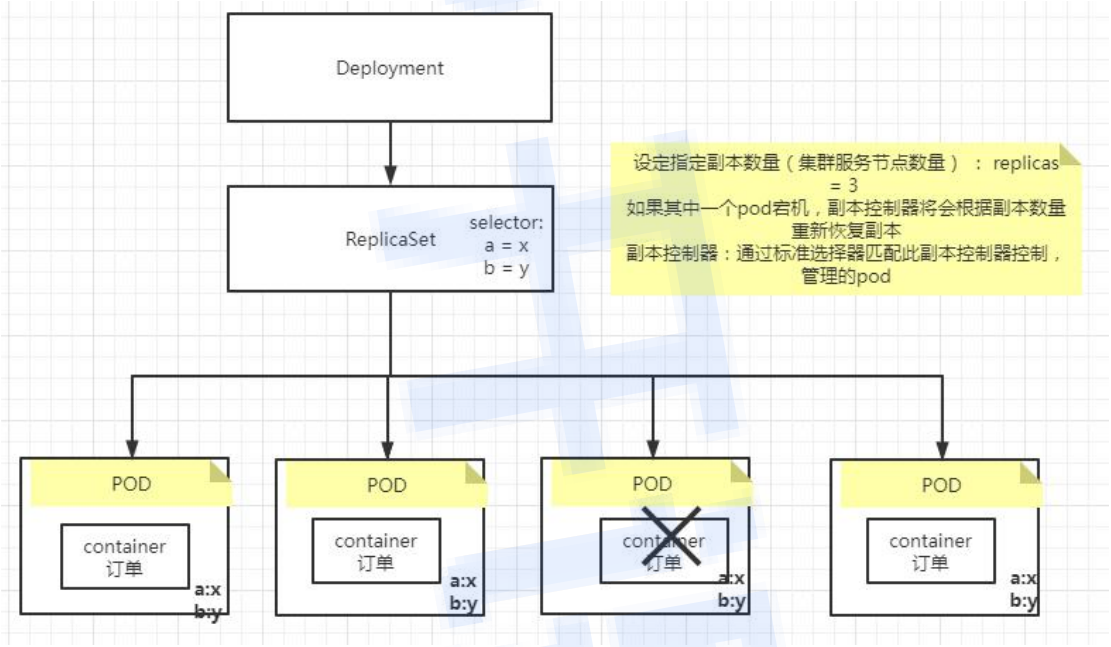
（3）、扩容和索容

（4）、暂停和继续 Deployment

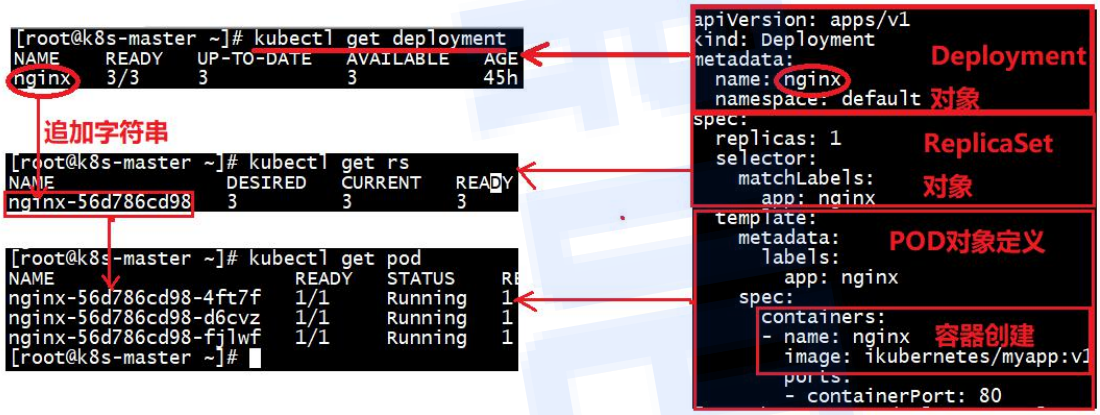
Deployment 不仅仅可以滚动更新，而且可以进行回滚，如果发现升级到 V2 版本后，发现服务不可用，可以回滚到 V1 版本。

注意事项：Deployment 对象仅仅是**无状态服务**的部署对象；**有状态服务**部署对象是 StatefulSet;

1）、Deployment 服务部署结构 – 无状态服务部署的结构

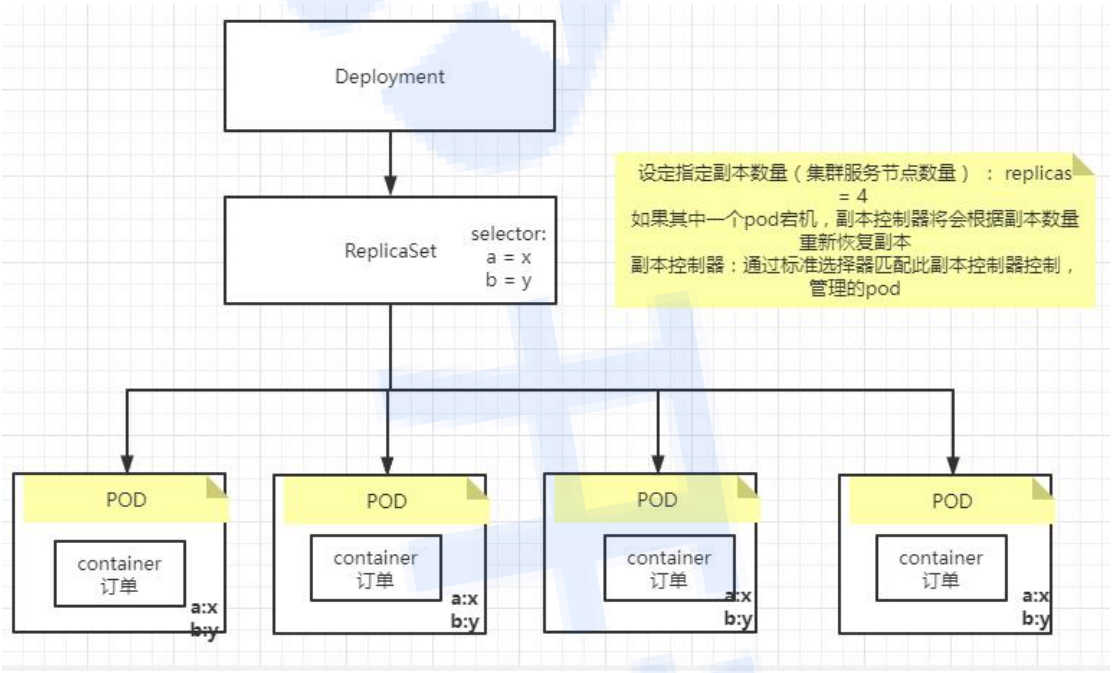


根据服务部署：查看了部署的 3 个资源对象：Deployment , RS，POD

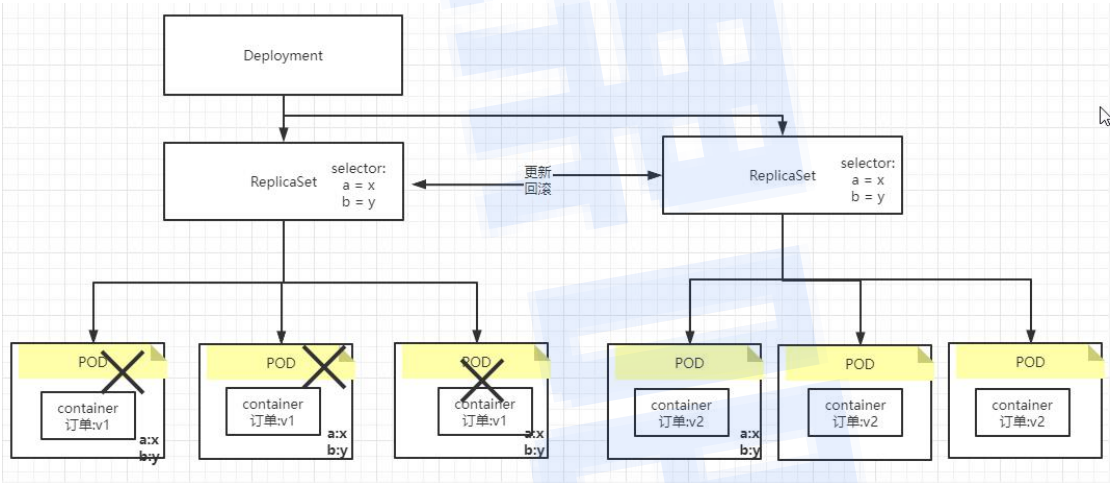


2）、Deployment 扩容缩容

```
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-56d786cd98-4ft7f   1/1     Running   1          45h
nginx-56d786cd98-d6cvz   1/1     Running   1          45h
nginx-56d786cd98-fjlwf   1/1     Running   1          45h
[root@k8s-master deploy-server]# kubectl scale deployment nginx --replicas = 2
Error: invalid argument "=" for "--replicas" flag: strconv.ParseInt: parsing "=": invalid syntax
See 'kubectl scale --help' for usage.
[root@k8s-master deploy-server]# kubectl scale deployment nginx --replicas=2
deployment.apps/nginx scaled
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-56d786cd98-4ft7f   1/1     Running   1          45h
nginx-56d786cd98-d6cvz   1/1     Running   1          45h
[root@k8s-master deploy-server]#
```



3）Deployment 支持滚动更新操作



软件版本迭代，从 v1 版本到 v2 版本上线部署，做版本的切换部署；Deployment 可以提供支持；

为了服务稳定性，做版本更新时候，逐渐把流量切换到新版本中，先删除一个老的版本 pod,然后创建一个新的版本的 pod；然后逐渐的切换流量；逐渐的更新；

```
[root@k8s-master deploy-server]# kubectl set image deployment/nginx nginx=ikubernetes/myapp:v3
deployment.apps/nginx image updated
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS        RESTARTS   AGE
nginx-5f59564844-bnjr6   1/1     Running       0          6s
nginx-5f59564844-khq8l   1/1     Running       0          4s
nginx-868855d887-zxbjj   0/1     Terminating   0          33s
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS        RESTARTS   AGE
nginx-5f59564844-bnjr6   1/1     Running       0          10s
nginx-5f59564844-khq8l   1/1     Running       0          8s
nginx-868855d887-zxbjj   0/1     Terminating   0          37s
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-5f59564844-bnjr6   1/1     Running   0          11s
nginx-5f59564844-khq8l   1/1     Running   0          9s
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-5f59564844-bnjr6   1/1     Running   0          12s
nginx-5f59564844-khq8l   1/1     Running   0          10s
[root@k8s-master deploy-server]# kubectl get pod -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP              NODE        NOMINATE
nginx-5f59564844-bnjr6   1/1     Running   0          20s   10.244.2.246    k8s-node2   <none>
nginx-5f59564844-khq8l   1/1     Running   0          18s   10.244.1.144    k8s-node1   <none>
```

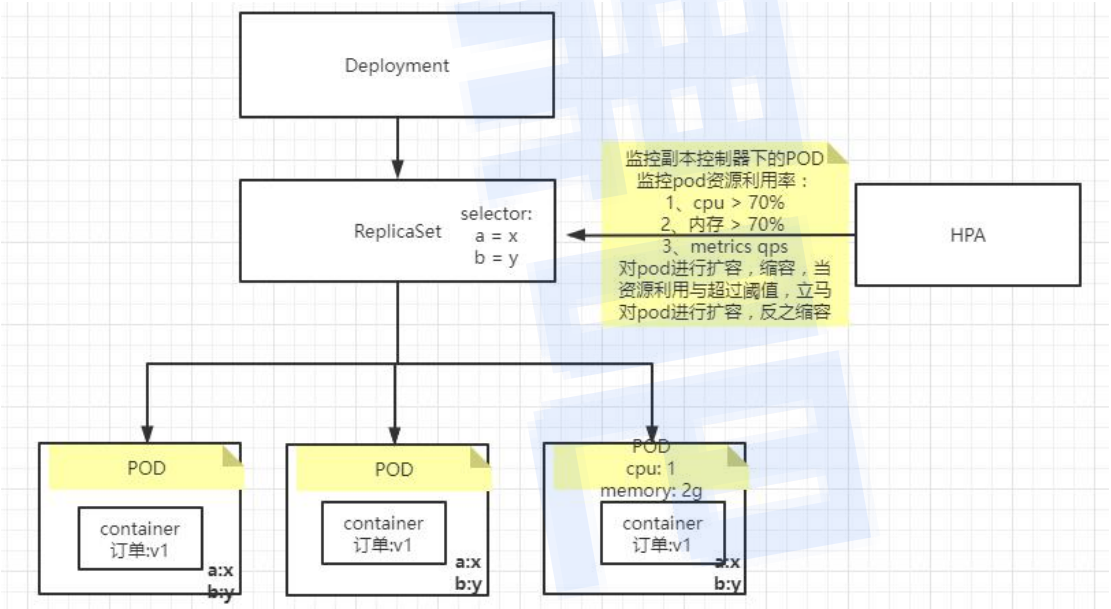
服务上线后进行版本回滚：

```
Use "kubectl options" for a list of global command-line options (applies to all commands).
[root@k8s-master deploy-server]# kubectl rollout undo deployment nginx --to-revision=4
deployment.apps/nginx skipped rollback (current template already matches revision 4)
[root@k8s-master deploy-server]# kubectl rollout undo deployment nginx --to-revision=3
deployment.apps/nginx rolled back
[root@k8s-master deploy-server]# kubectl get pod -o wide
NAME                     READY   STATUS        RESTARTS   AGE    IP              NODE        NOMINA
TES
nginx-5f59564844-9dknx   1/1     Running       0          9s     10.244.2.248    k8s-node2   <none>
nginx-5f59564844-zkf57   1/1     Running       0          7s     10.244.1.146    k8s-node1   <none>
nginx-868855d887-ldlgw   0/1     Terminating   0          5m9s   10.244.2.247    k8s-node2   <none>
[root@k8s-master deploy-server]# kubectl get pod -o wide
```

5.4 HPA

Horizontal Pod Autoscaling 仅适用于 Deployment 和 ReplicaSet,在 V1 版本中仅支持根据 Pod 的 CPU 利用率扩容，在 vlalpha 版本中，支持根据内存和用户自定义的 metric 扩缩容

HPA 控制器资源对象实现自动化扩容，缩容（因对突发流量）：



HPA 对 pod 服务进行扩容和缩容都是自动化，无需运维和开发进行接入；

```
[root@k8s-master deploy-server]# kubectl autoscale deployment nginx --min=2 --max=10
horizontalpodautoscaler.autoscaling/nginx autoscaled
[root@k8s-master deploy-server]# kubectl get hpa
NAME    REFERENCE          TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
nginx   Deployment/nginx   <unknown>/80%   2         10        0          6s
[root@k8s-master deploy-server]#
```

5.5 DaemonSet

DaemonSet 确保全部（或者一些 [node 打上污点（可以想象成一个标签）,pod 如果不定义容忍这个污点，那么 pod 就不会被调度器分配到这个 node]）Node 上运行一个 Pod 的副本。当有 Node 加入集群时，也会为他们新增一个 Pod。当有 Node 从集群移除时，这些 Pod 也会被回收。删除 DaemonSet 将会删除他创建的所有 Pod,使用 DaemonSet 的一些典型用法：

（1） 运行集群存储 daemon,例如在每个 Node 上运行 glustered,ceph

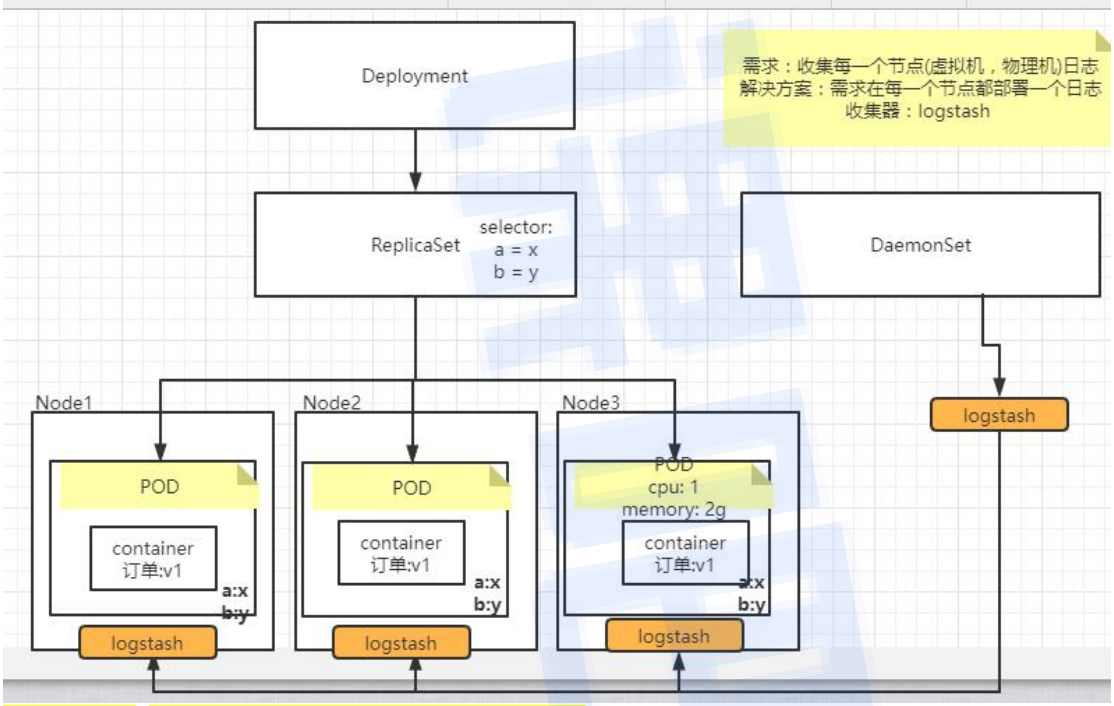
（2）在每个 Node 上运行日志收集 Daemon,例如：fluentd、logstash.

（3）在每个 Node 上运行监控 Daemon,例如：Prometheus Node Exporter

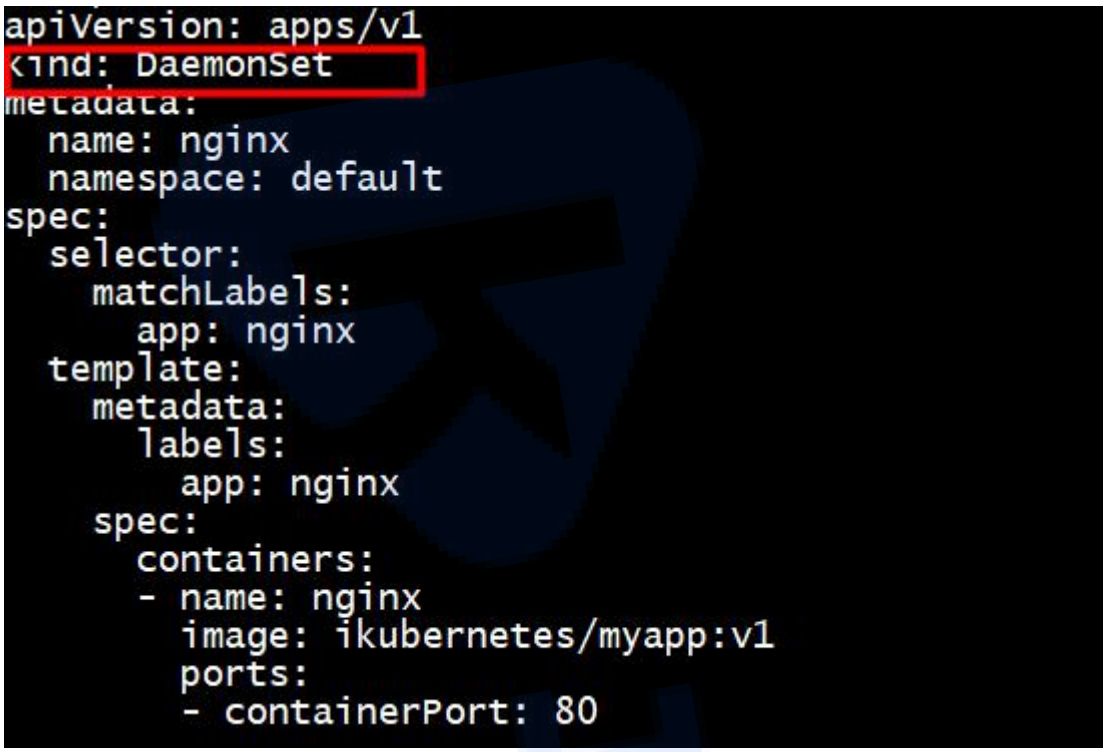
例如：项目实际生产环境中，有 100 个 node 节点，每一个节点中都部署很多的 pod 服务，此时收集这些 pod 服务的日志，就必须在每一个 node 节点去部署一个日志收集程序？

1)、传统的部署方法：挨个在每一个 node 节点部署一个日志收集程序（相当费劲）

2)、DaemonSet 就可以同时在每一个 node 节点都部署一个相同的日志收集程序；



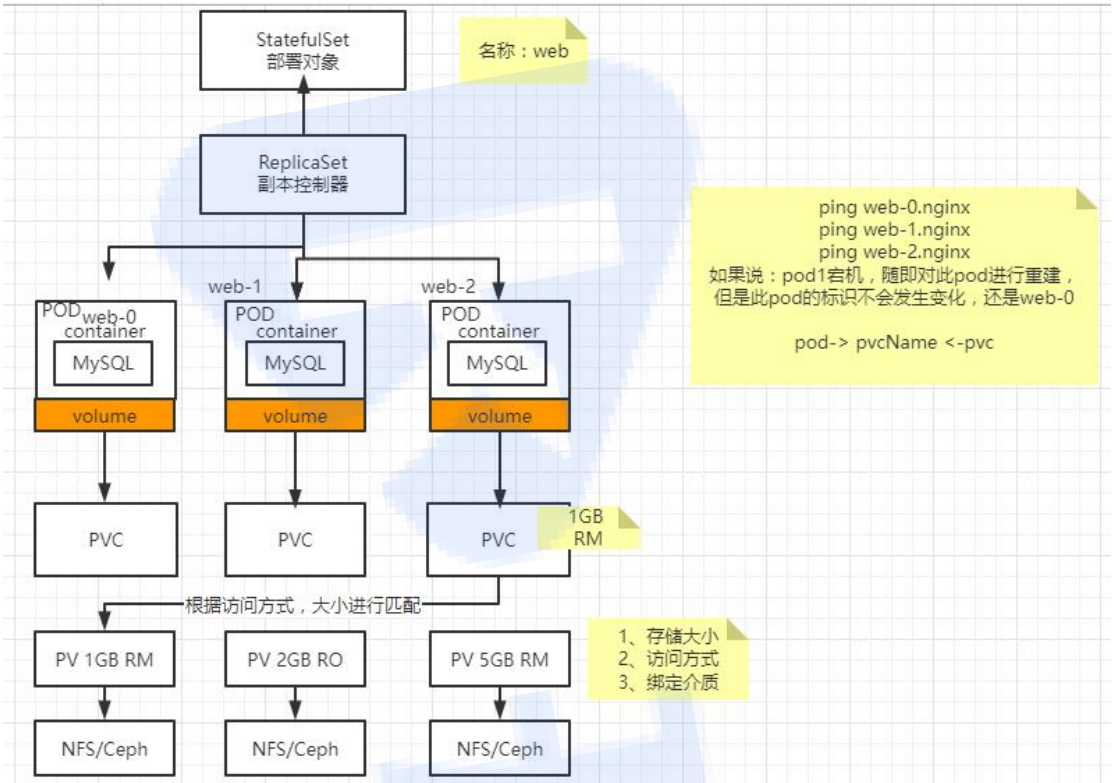
DaemonSet 是确保被部署的程序在每一个 node 节点都被部署一份（一个 pod 服务）



5.6 statefulSet

思考：是否可以使用 deployment 资源对象部署 mysql，es, rocketmq 等等这样的程序？？

答案：无法部署，因为 deployment 部署服务时候，不考虑数据存储的问题，因此在 deployment 资源对象下的 pod 服务进程宕机后，数据会丢失；pod 本身生命周期不稳定的，pod 本身仅仅是运行在操作系统中一个服务进程，随时都可能异常退出，一旦发生异常，如果 pod 存储数据就丢失了；因此在 deployment 中仅仅被用来部署无状态服务；

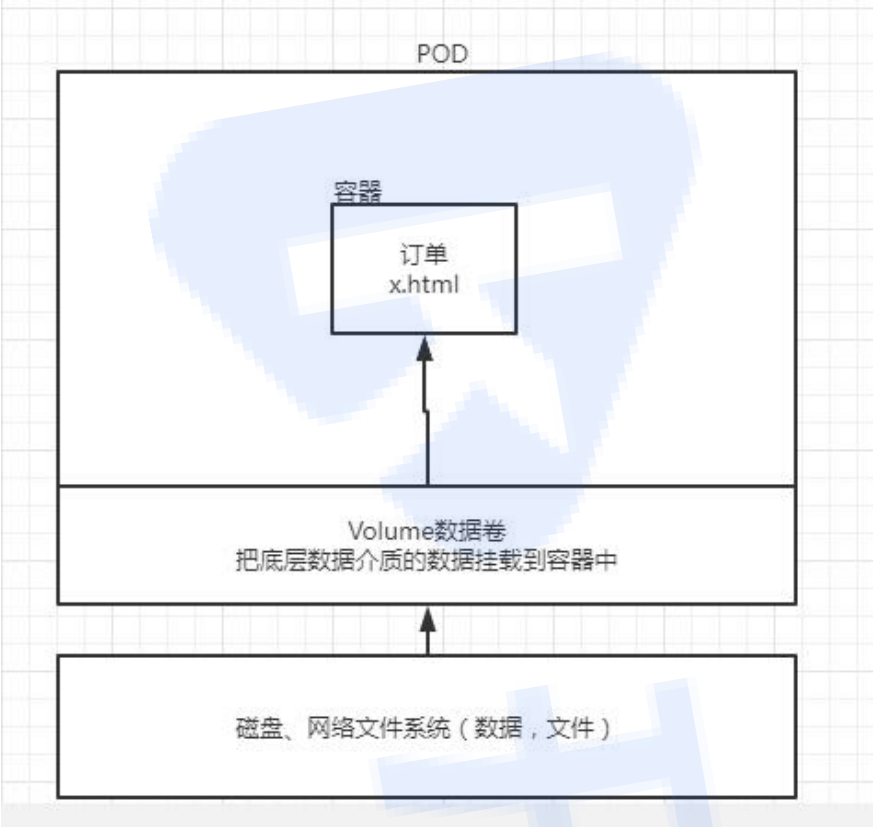


使用 statefulSet 部署有状态服务，当 Statefulset 内部的 pod 异常退出的时候，此时持久化数据卷 pv 是不会退出的，因此当 pod 被重建的时候，就会重新和 pv 进行绑定，因此此时数据不会丢失；

5.7 Volume

Volume 是 kubernetes 抽象出的数据存储的资源对象；volume 数据卷会把底层存储介质（磁盘，网络文件系统）的数据挂载到 pod 服务内部容器中；

Volume 是 kubernetes 管理的数据卷，而不是 docker 的数据卷；



Volume 数据卷特点：

1）、Volume 是 kubernetes 抽象出的资源对象，本身并不存储数据，只是一个介质，只是把数据给挂载到容器内部；

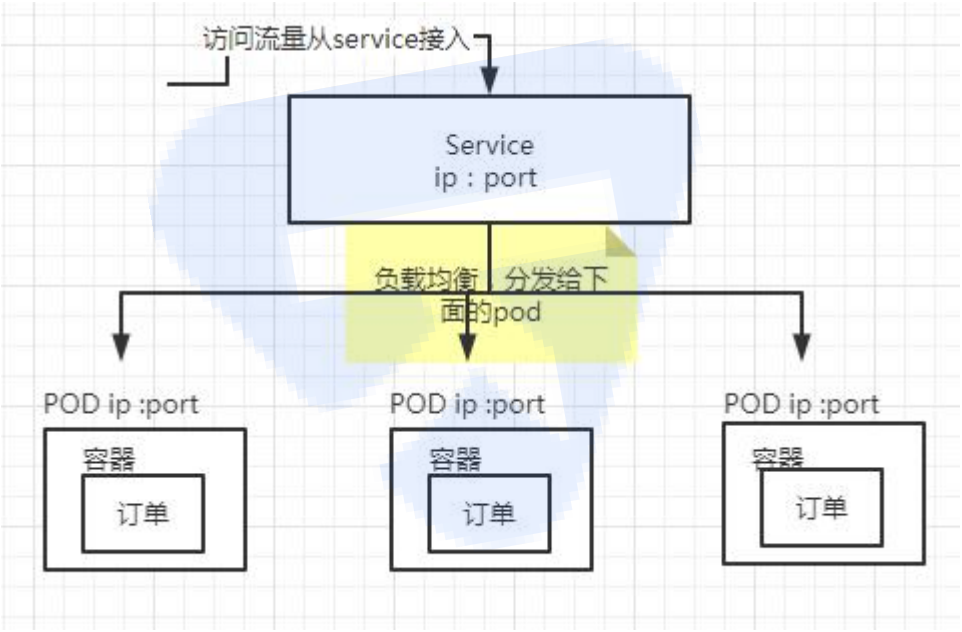
2）、容器异常退出，volume 数据卷不会消失，数据也不会丢失

3）、pod 服务异常退出了，volume 数据卷消失了，数据丢失了；

6 POD 网络原理

6.1 负载均衡

在服务的时候，部署了很多服务，一个服务有部署成了集群，一个集群有多个 pod 服务，如何在多个 pod 服务之间实现负载均衡呢？？



Kubernetes 需要在多个 pod 副本之间实现负载均衡，必须要引入一个新的资源对象：Service 资源对象，这个 Service 资源对象可以实现在多个 pod 副本之间实现负载均衡（请求分发）示例：

1）通过指令创建 service 实现负载均衡?? --- 在老版本 kubernetes 比较容易实现操作，在新版本中，不太好实现；

2）通过 yaml 文件创建 service 对象？？

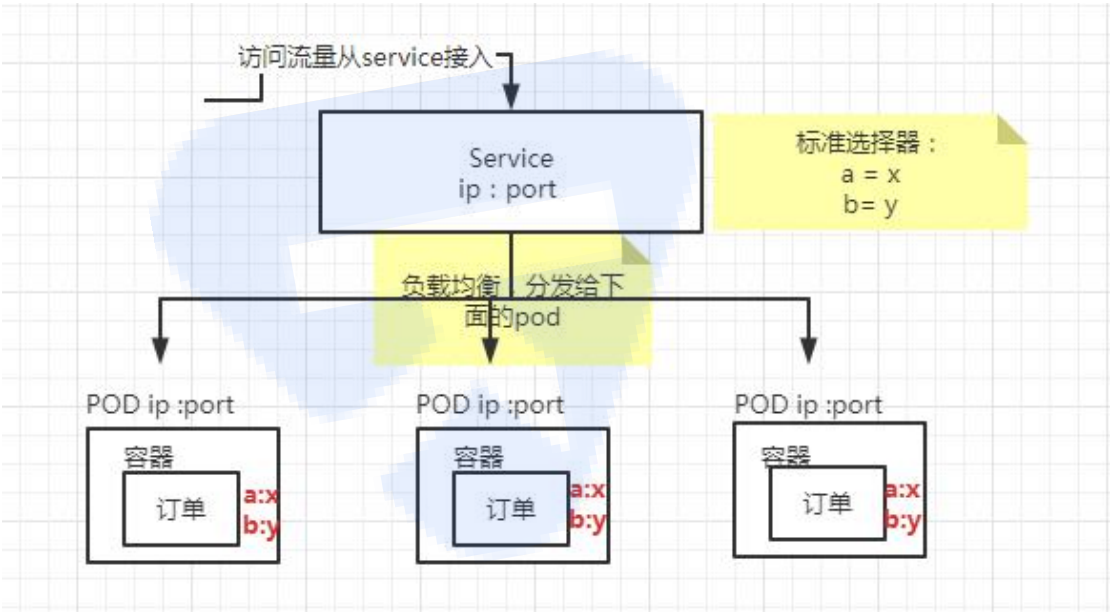
```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
spec:
  selector:
    app: nginx
  ports:
  - port: 80
    targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  namespace: default
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
```

经过测试，通过 service 来实现 pod 的方式，实现负载均衡，默认使用轮询方式；

```
[root@k8s-master deploy-server]# kubectl get pod
NAME                     READY   STATUS    RESTARTS   AGE
nginx-56d786cd98-nvlz4   1/1     Running   0          3m56s
nginx-56d786cd98-qvm42   1/1     Running   0          3m56s
nginx-56d786cd98-t6dz4   1/1     Running   0          3m56s
[root@k8s-master deploy-server]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   236d
nginx        ClusterIP   10.107.56.135   <none>        80/TCP    4m2s
[root@k8s-master deploy-server]# curl 10.107.56.135/hostname.html
nginx-56d786cd98-t6dz4
[root@k8s-master deploy-server]# curl 10.107.56.135/hostname.html
nginx-56d786cd98-nvlz4
[root@k8s-master deploy-server]# curl 10.107.56.135/hostname.html
nginx-56d786cd98-qvm42
[root@k8s-master deploy-server]# curl 10.107.56.135/hostname.html
nginx-56d786cd98-t6dz4
[root@k8s-master deploy-server]# curl 10.107.56.135/hostname.html
nginx-56d786cd98-nvlz4
[root@k8s-master deploy-server]# curl 10.107.56.135/hostname.html
nginx-56d786cd98-qvm42
[root@k8s-master deploy-server]#
经过测试： 轮询方式访问pod
```

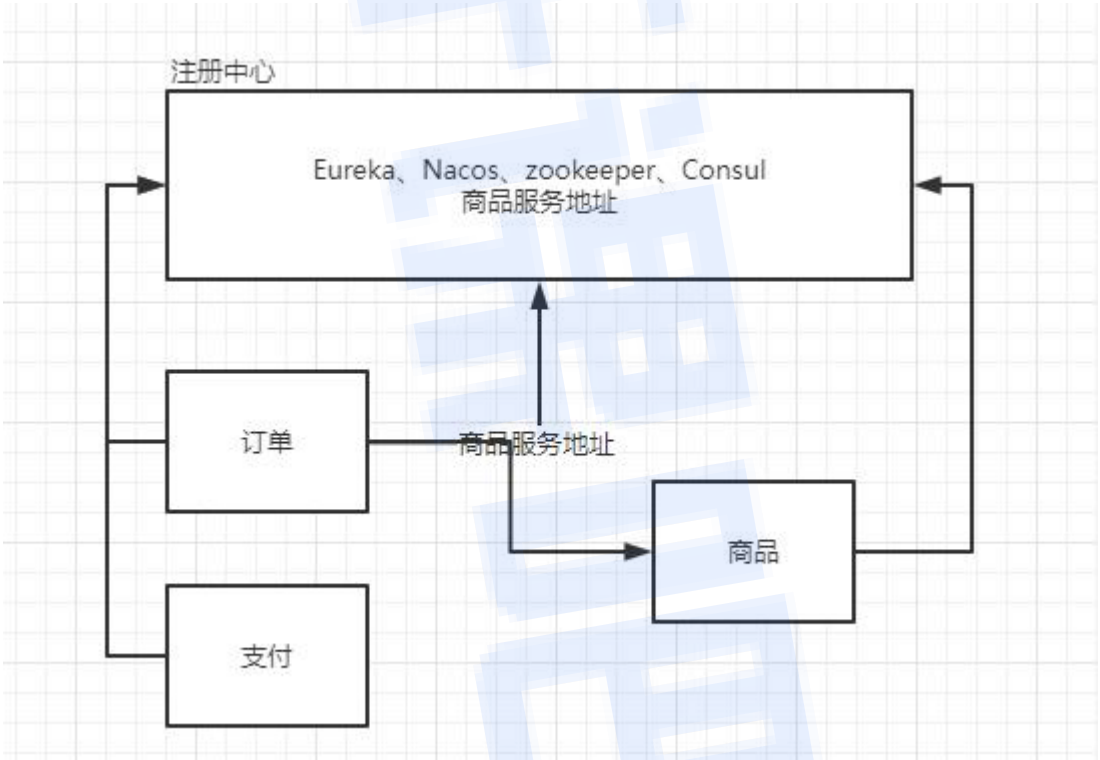
思考：Service 是如何确定为哪些 pod 服务提供服务呢？？

答案：标签—通过标签进行关联相关的 pod 服务的；



6.2 域名解析

Kubernetes 中的服务之间实现相互的访问，通过服务名称的方式识别对方的服务的地址；但是网络通过服务名称进行访问是不可行的，因此 kubernetes 提供了一个 dns 域名进解析服务来实现服务名称解析（解析的过程也叫做服务发现）；

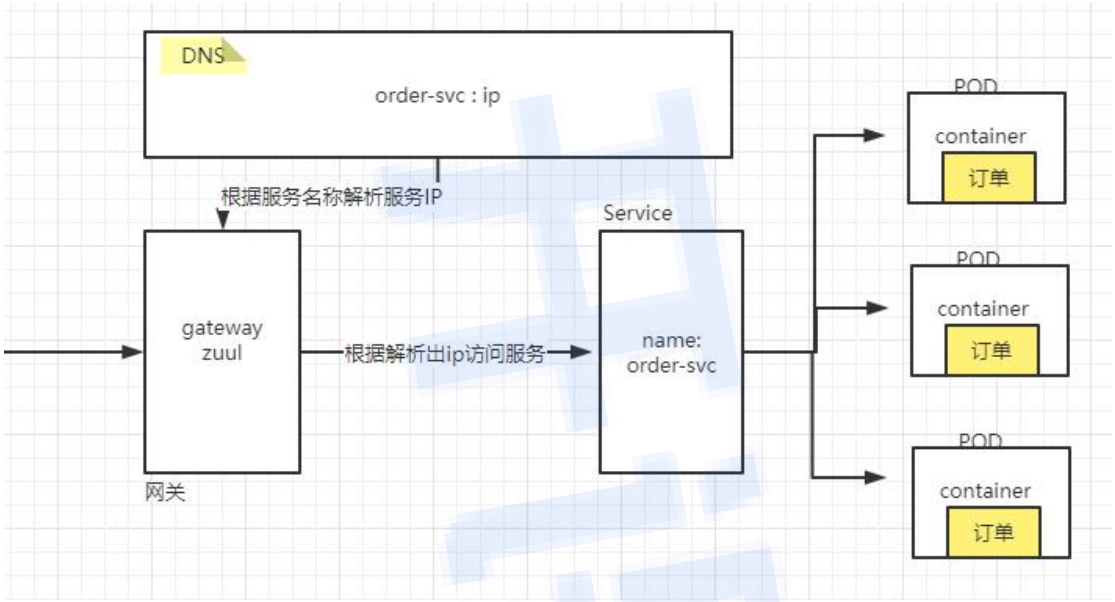


这种服务名称解析的方式类似微服务架构中注册中心（zookeeper，nacos,consul）实现服务发现；kubernetes 服务之间访问也是通过服务名称解析出服务的地址（ip 地址）最终实现服务通信；

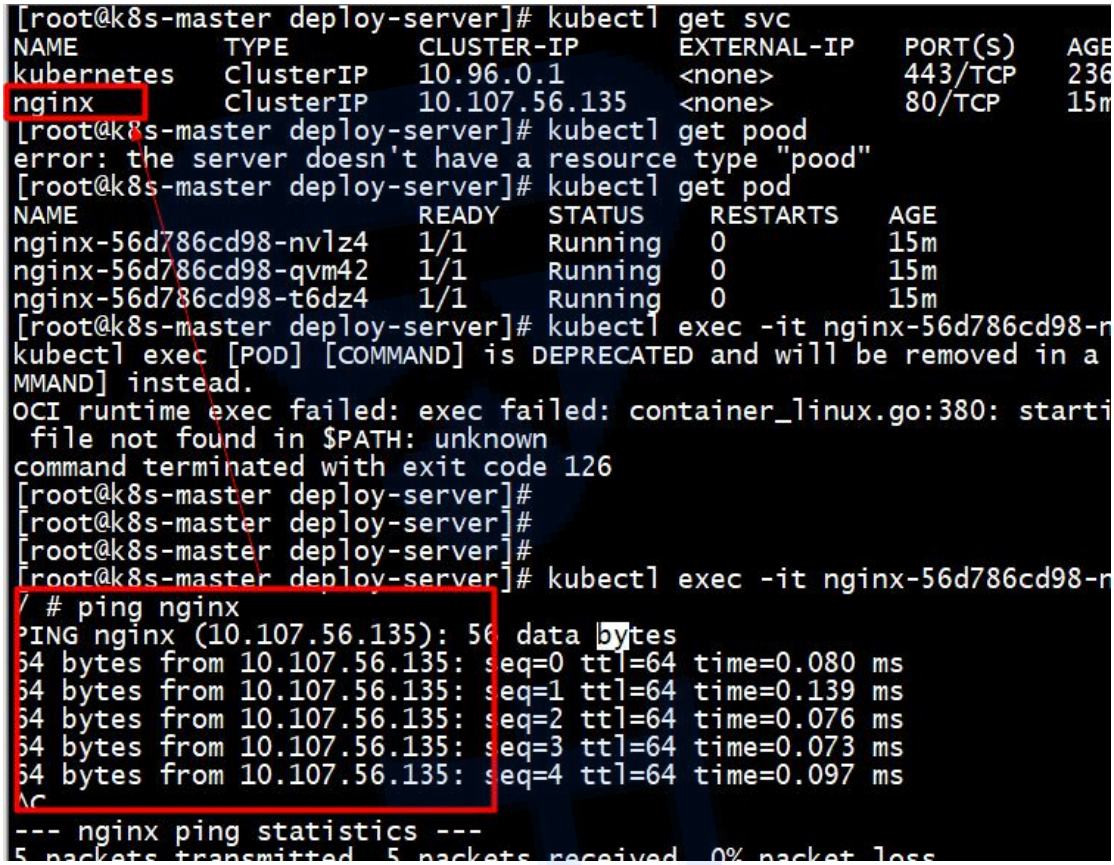
Kubernetes 内部服务直接调用是 dns 来进行地址解析，DNS 是 kubernetes 基础组件，是安装的必备的基础组件；

```
[root@k8s-master deploy-server]# kubectl get pod -n kube-system
NAME                                   READY   STATUS    RESTARTS   AGE
coredns-7ff77c879f-5splg               1/1     Running   27         236d
coredns-7ff77c879f-tm6z6               1/1     Running   27         236d
etcd-k8s-master                        1/1     Running   29         236d
kube-apiserver-k8s-master              1/1     Running   29         236d
kube-controller-manager-k8s-master     1/1     Running   29         236d
kube-flannel-ds-5zvhx                  1/1     Running   41         235d
kube-flannel-ds-dlg6n                  1/1     Running   41         236d
kube-flannel-ds-n22r9                  1/1     Running   42         235d
kube-proxy-5gv22                       1/1     Running   51         236d
kube-proxy-876hr                       1/1     Running   52         235d
kube-proxy-h4slx                       1/1     Running   50         235d
kube-scheduler-k8s-master              1/1     Running   30         236d
metrics-server-69965bbc5b-pfjvn        1/1     Running   3          81d
[root@k8s-master deploy-server]#
```

Kubernetes 服务之间的调度流程：



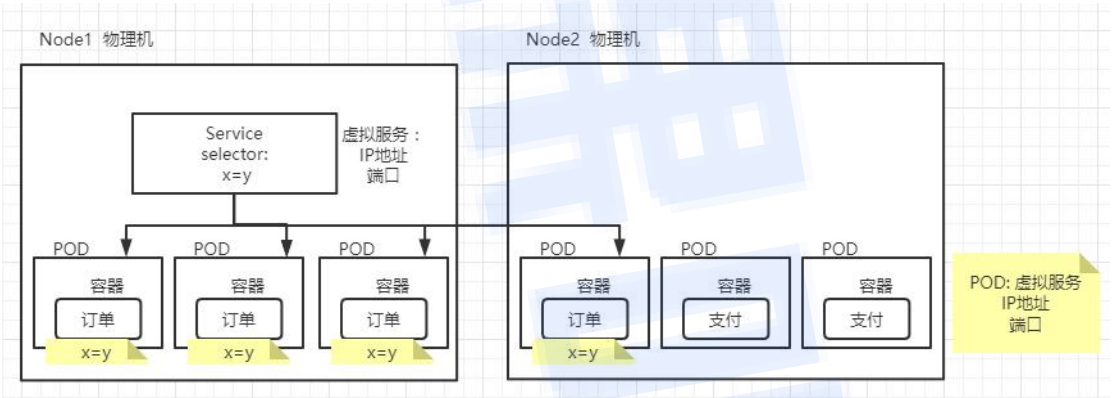
示例：在 pod 应用内部通过名称访问服务（service），是否可以通过？？



6.3 外网访问

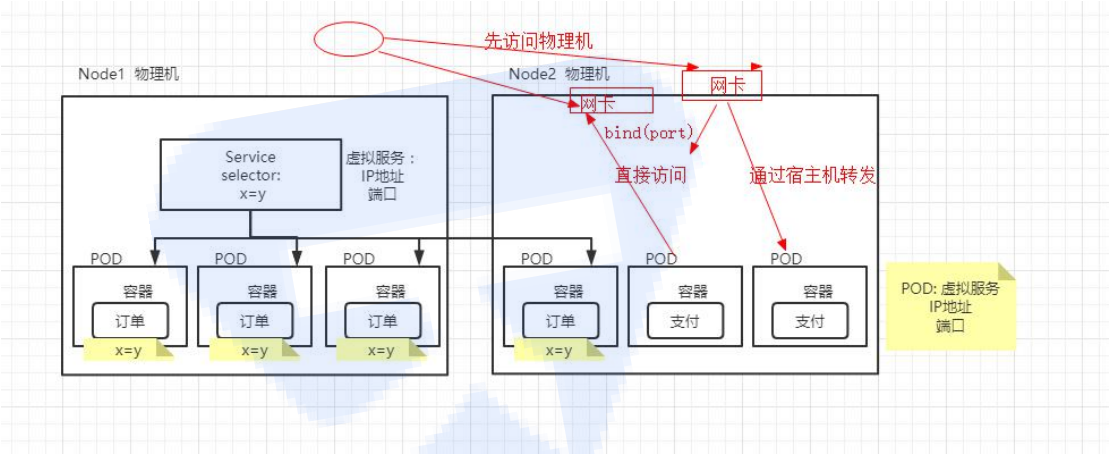
思考：是否可以直接使用 service ip 地址在外网访问呢？？

思考：为什么 Service 无法实现在外网访问呢？？

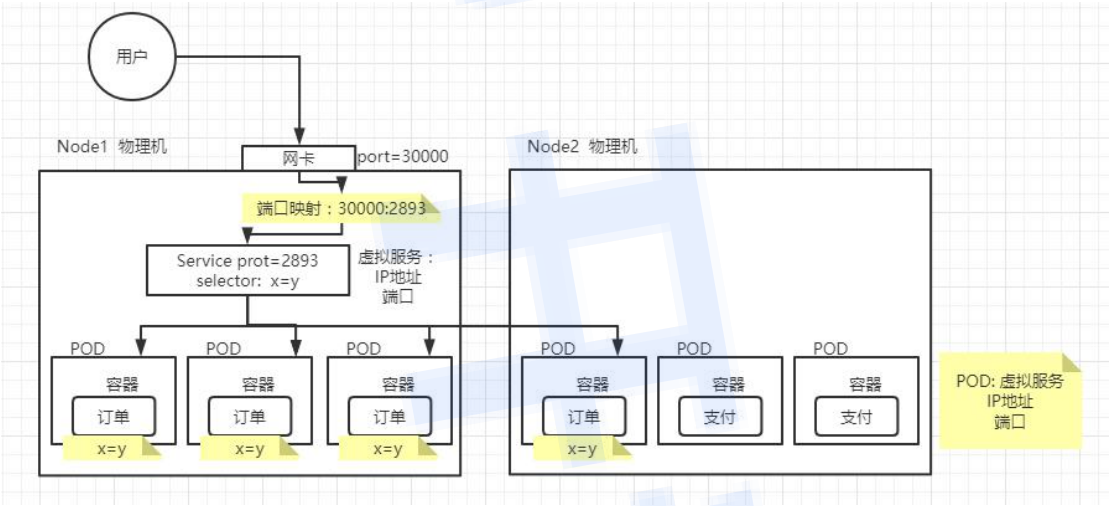


Service 是运行在操作系统内部的一个虚拟服务（有自己的 IP 地址，有自己的端口），pod 也是一个虚拟服务（有自己的 IP 地址，有自己的端口），但是他们仅仅是一个操作系统内部的进程；

由于他们是虚拟服务，没有在操作系统内部绑定端口，没有和物理机网卡进行绑定；因此外网是无法直接通过物理机访问的；



因此要访问内部服务程序，必须要宿主机网卡绑定一个端口，才能实现操作系统内部服务程序的访问；



因此可以在物理机上开辟一个端口，然后使得这个端口和内部虚拟服务端口实现一个映射，然后把请求从物理机网卡转发到内部的虚拟服务；

```
  name: nginx
  namespace: default
  resourceVersion: "1860347"
  selfLink: /api/v1/namespaces/default/services/nginx
  uid: 438dde76-f2e9-4171-9f92-01e72581d806
spec:
  clusterIP: 10.107.56.135
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
```

ClusterIP：局域网通信IP地址
NodePort：要在物理机开辟端口，进行映射
把clusterIP修改为NodePort

查看 service 端口映射：

```
[root@k8s-master deploy-server]# kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP        236d
nginx        NodePort    10.107.56.135   <none>        80:30086/TCP   50m
[root@k8s-master deploy-server]#
```

查看物理机端口监听状态：

```
[root@k8s-master deploy-server]# netstat -anpt | grep 30086
tcp        0      0 0.0.0.0:30086           0.0.0.0:*               LISTEN      6915/kube-proxy
[root@k8s-master deploy-server]#
```

7 Pod 网络架构

7.1 网络结构

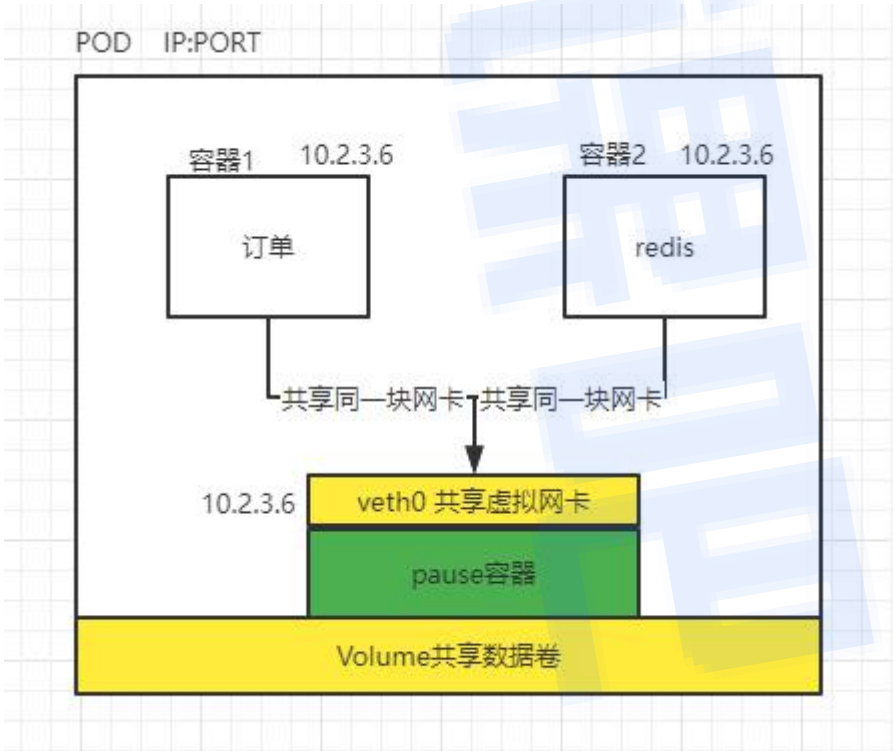
POD 基本概念：

1）、POD 是 kubernetes 操作的最小单元；kubernetes 在部署服务时候不是部署容器，而是部署的是 pod;

2）、POD 也是一个容器，具有独立的沙箱环境，有自己的 IP 地址，有自己的端口，有自己的 hostname;

3）、POD 是容器的容器，内部封装一个容器，容器内部封装的是服务程序；

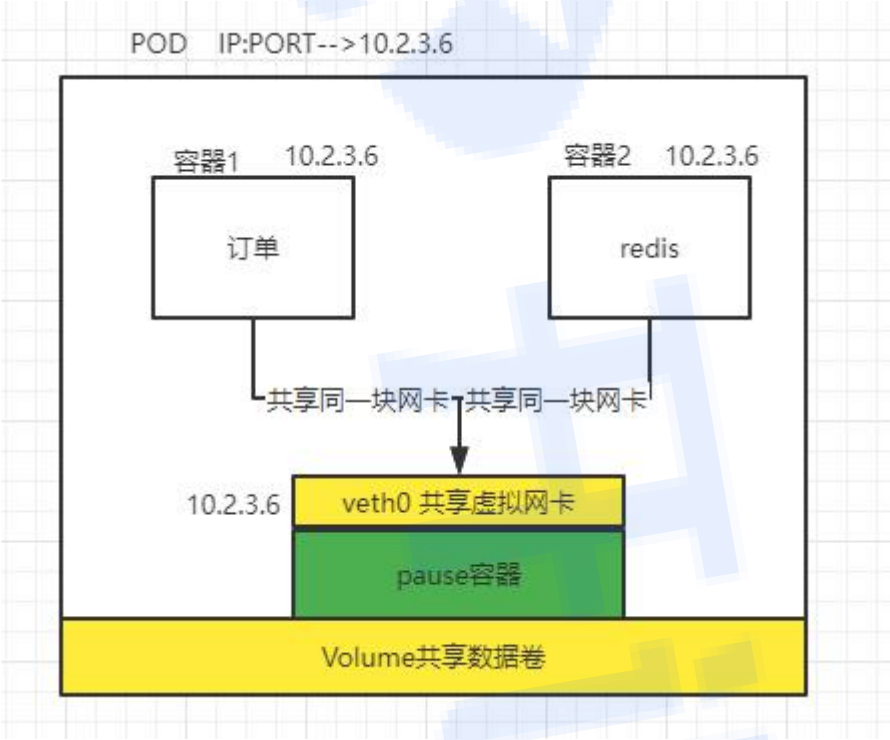
POD 本身和容器是一样的，是一个虚拟的概念（虚拟服务），本身运行在操作系统内部服务进程，相当于是一个独立服务器（虚拟化）；POD 内部封装是一个容器（pod 内部可以封装一个容器，也可以是多个容器），在部署节点上（物理机，虚拟机），POD 和 POD 之间是相互独立的沙箱环境；



思考：一个 pod 内部在实际的生产环境中，如何规划一个 pod 内部部署几个容器呢？？
答案：一个 pod 内部通过备用来部署一个容器；

7.2 IP 地址共享

POD 内部容器共享同一个块网卡，因此 ip 地址是相同的，pod 有自己的 ip 地址，pod 也是共享这块网卡，因此 podIP，容器 ip 都是同一个 ip;

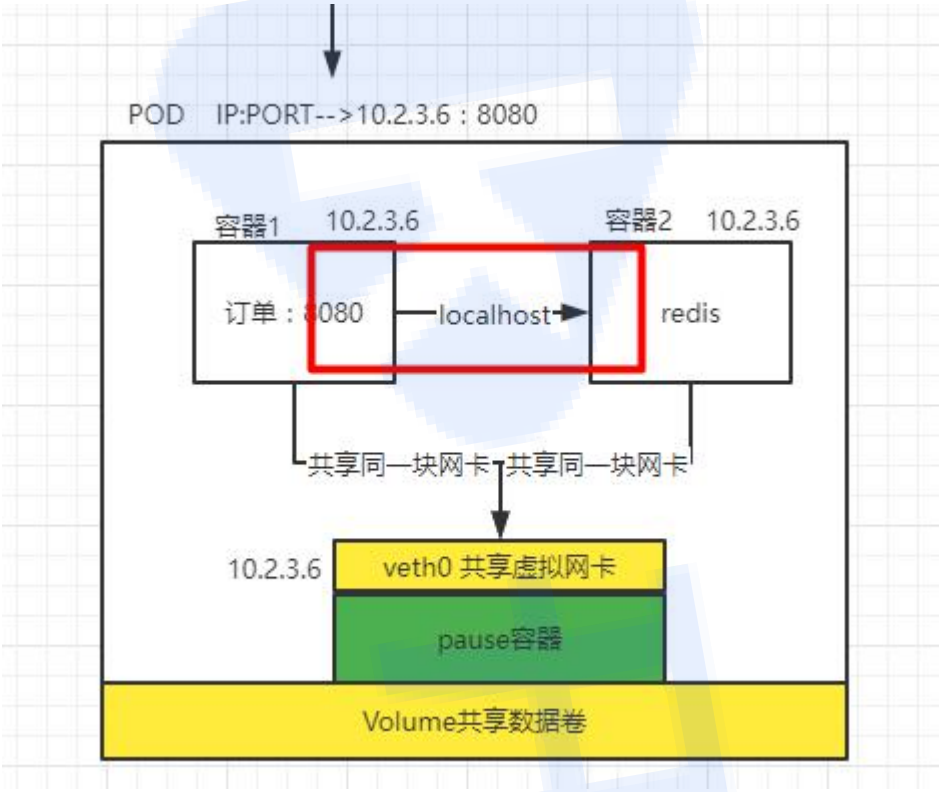


Pod 端口直接使用容器端口；因此访问 pod 内部容器服务时候，直接使用 podip:容器端口即可实现 pod 内部容器访问；

示例： 查看 pod 的 ip 地址，是否和容器的 ip 地址一致？？

```
[root@k8s-master deploy-server]# kubectl get pod -o wide
NAME                     READY   STATUS    RESTARTS   AGE   IP              NODE
nginx-56d786cd98-nvlz4   1/1     Running   0          66m   10.244.1.149    k8s-node1
nginx-56d786cd98-qvm42   1/1     Running   0          66m   10.244.2.253    k8s-node2
nginx-56d786cd98-t6dz4   1/1     Running   0          66m   10.244.2.254    k8s-node2
[root@k8s-master deploy-server]# kubectl exec -it nginx-56d786cd98-nvlz4 -- sh
/ # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
3: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue state
    link/ether de:2f:16:09:c4:0b brd ff:ff:ff:ff:ff:ff
    inet 10.244.1.149/24 brd 10.244.1.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #
```

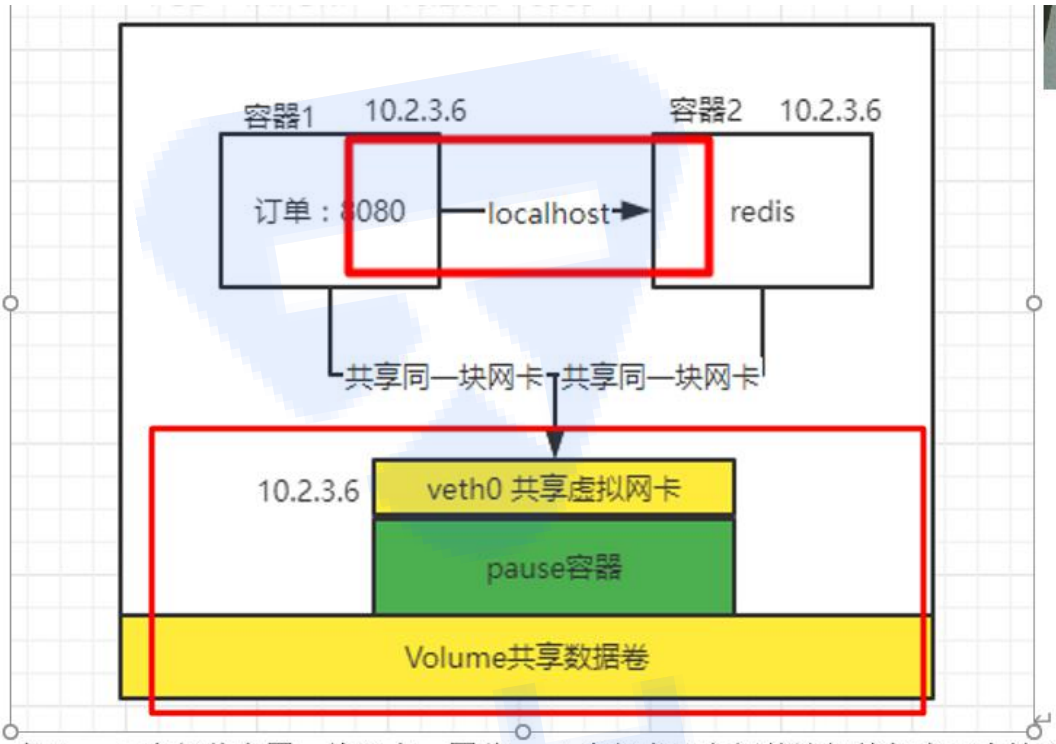
7.3 容器内部



由于 pod 内部共享同一块网卡，因此 pod 内部容器之间的访问就相当于本地服务访问，使用 localhost 访问即可；

7.4 Pause 容器

创建一个 pod 服务的时候，首先创建 pause 容器；因此 pause 容器创建一个网卡（共享网卡），同时 pause 容器会创建一个共享网卡；



由于 pod 内部共享同一块网卡，因此 pod 内部容器之间的访问就相当于本地服务