链滴

运维高手第 15 课：服务向 Kubernete 容器平台迁移必须了解的事情

作者：chenteng

原文链接：https://ld246.com/article/1636555461066

来源网站：链滴

服务向 Kubernete 容器平台迁移必须了解的事情

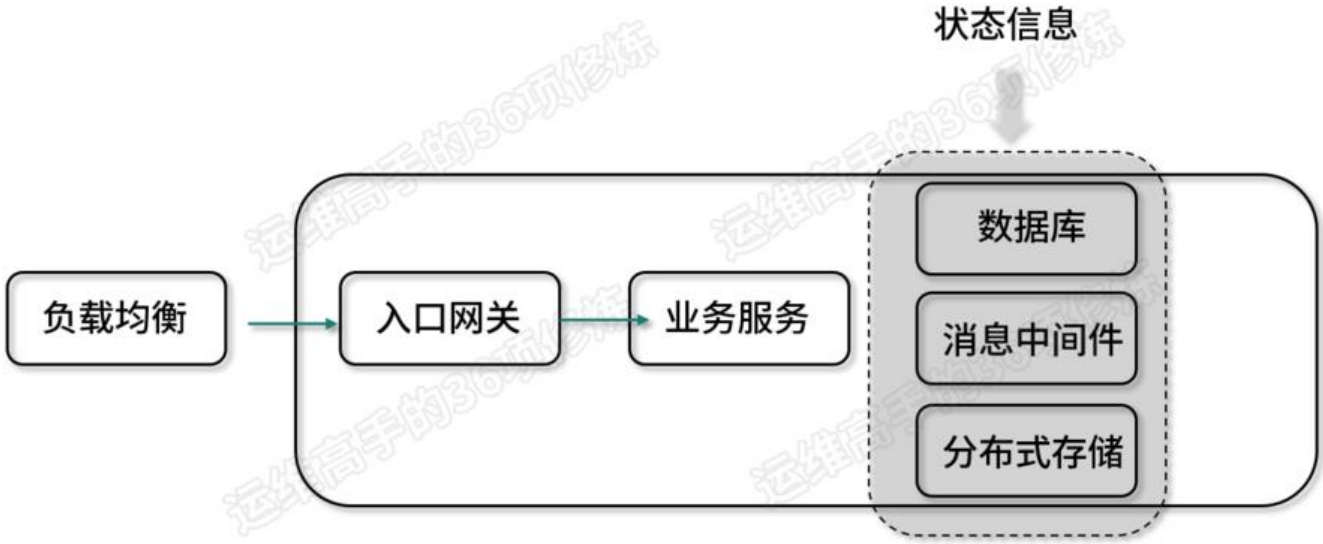
一、业务迁移到 K8S 需要关注的问题

我们先来学习第 1 个模块：业务迁移到 K8S 所需要关注的问题，这里我罗列了几个相关的问题：

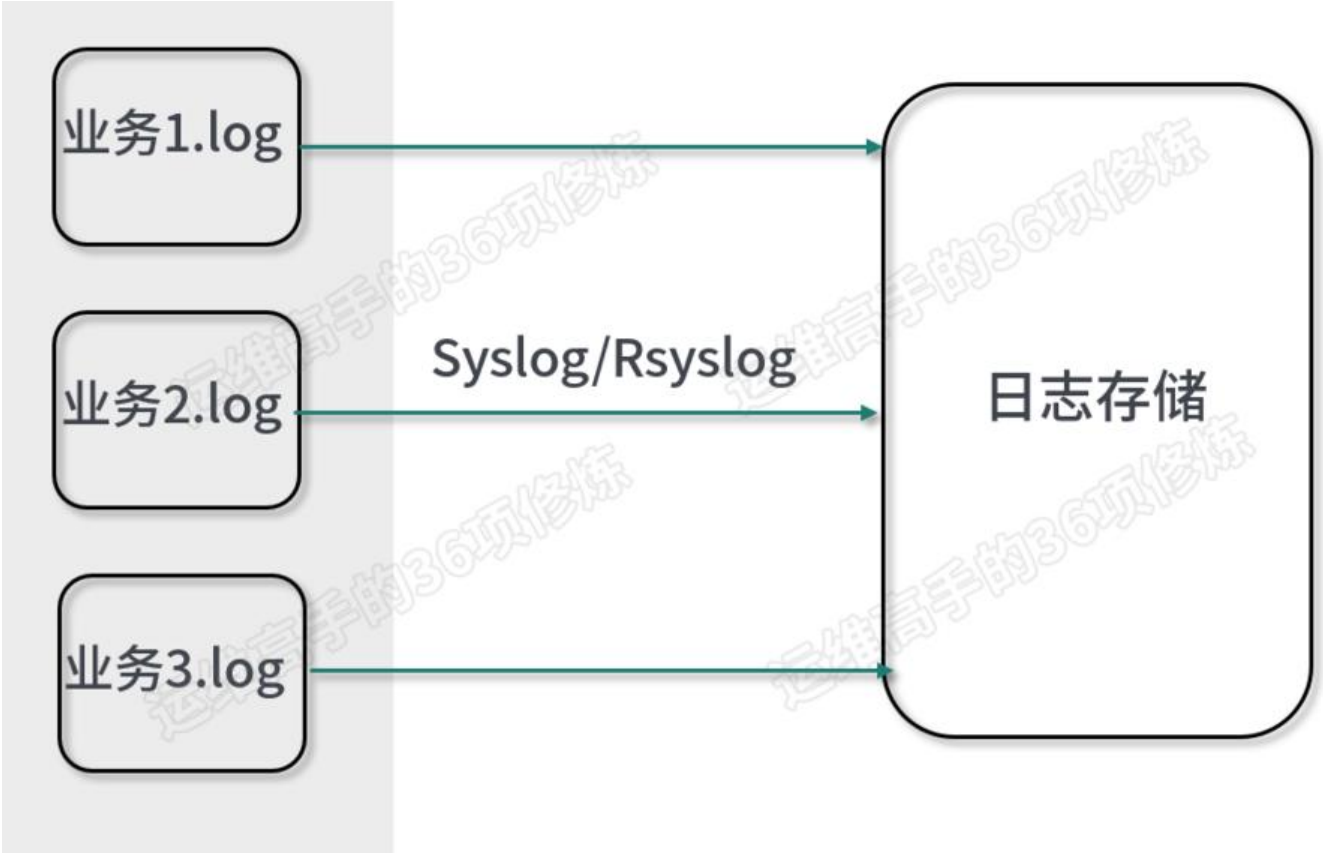
程序中有状态的内容

第一点，我们需要关注程序中已有状态的数据存放。因为传统方式开发程序通常会保存一些本地化的据，我们称它为一个本地有状态的服务，比如业务节点内存中保存用户登录所需的 Session 数据，这数据经常被保存在本地节点所开辟的一块内存空间里。

为什么需要关注程序中有状态的这些内容呢？这是因为程序迁移到 K8S 的架构下，K8S 调度需要对 Pd 节点的弹性扩缩或资源调度。如果服务固定依赖本地存储空间数据，那么就可能出现因为 K8S 的调导致这部分已经存储在本地的数据丢失，因为启用新的业务 pod，无法读取原有的 pod 上的数据。时我们需要将数据在本地做无状态化。



以上就是在迁移到 K8S 之前所需要重点考虑的第一个问题，如何解决呢？就需要将业务服务做到本无状态化，这里我画了一张图。



当请求到达业务服务后，业务服务会把这部分有状态的信息数据存储到一个公共的服务组件里去，而是放在业务服务本地资源的空间里，比如我们可以把数据存储在数据库、消息队列、分布式存储里。

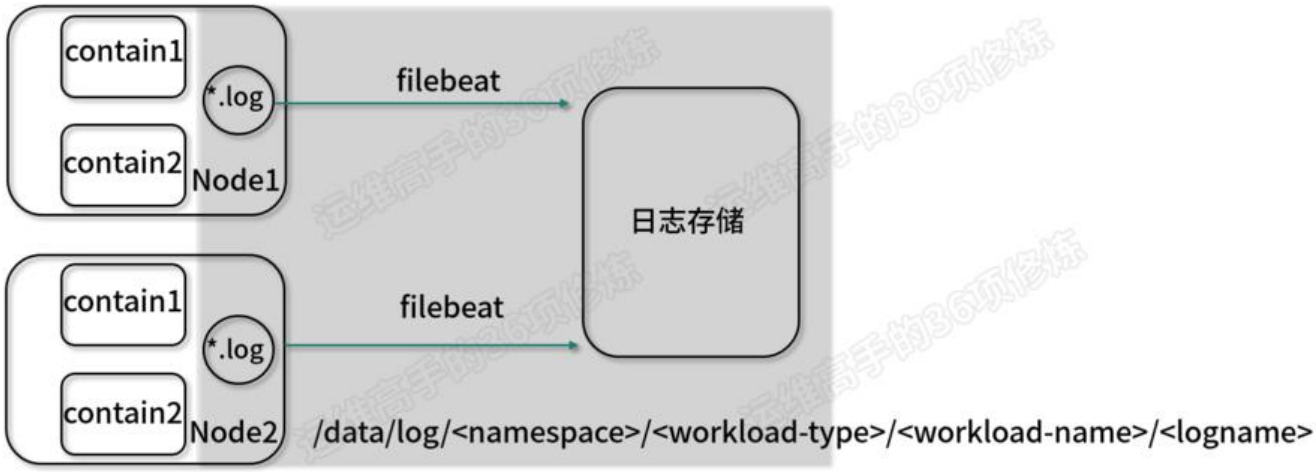
业务日志收集排查

第二个问题，当程序从非 K8S 架构迁移到 K8S 架构后，对于日志的收集也是我们需要关注的问题，在 Linux 的开源系统环境下，传统的收集方式是首先通过 Syslog 协议，然后通过 Linux 系统上的 Syslo -d 或者 Rsyslog 日志服务对业务日志进行收集。这里我画了一张图，这张图显示的是在传统收集方和非 K8S 架构下，对日志常用的收集方式。

我们看到在这样一种架构下，对日志的收集有几个特点，第 1 个特点就是收集的业务节点是固定的某节点。

但是当程序迁移到 K8S 架构下后，日志的收集会遇到一个很大的问题，因为容器上的业务日志可能随时飘移，这就导致 K8S 所有 node 节点上的日志、目录路径和信息会动态地增加和减少。这个时我们会采用一种新的日志收集工具来通配 K8S 架构下的模式，这时常用到工具有ELK 、EFK。

EFK 是个什么样的架构呢？E 表示 Elasticsearch，它是一个搜索存储日志引擎，既可以进行日志的存，也可以进行日志的检索，是一个非常强大的工具。K 表示 Kibana，它是 Elasticsearch 的界面化展。有了这两个东西，我们相当于有了一套后端日志的检索和展示平台。相比之前方式重要的区别在于日志收集方式，Rsyslog 是非 K8S 架构下常用到的日志收集器，那么在 K8S 架构里，工具常用 filebet 或者 fluent。相比 Rsyslog，这两个搜集组件支持对日志路径、名字的通配，我们不用关心某个具的日志，它们可以通配某一级目录或者是某一个目录的所有通配日志名称。



针对这套 EFK 架构，我同样画了一张图，这里介绍最常用到的架构方案。在 K8S 的 node 节点里可通过 PVC 让容器将日志目录挂载到 node 节点的本地目录下，然后通过 filebeat 去收集日志信息，到中心日志存储里。

filebeat 可以通配 *.log 这样的样例文件，这样就能直接把整体的日志信息作同步。在图示中我画了个路径，看到固定的日志路径 /datal/log 下，我可以通配它的 <namespace> 目录，或是 <workloa-type> 、 <workload-name> 和 <logname> 这些日志文件，它们都可以通过 filebeat 来通配。对于 Rsyslog 而言，filebeat 对于日志收集器对 K8S 的弹性和资源调度模式更加友好。

Java 内存溢出问题排查

刚刚讲到的就是对日志的收集，对于程序员或者是运维工程师而言，K8S 架构及非 K8S 架构在排查样一个问题时，思路可能会有一些小的差异。这里以对 Java 内存溢出问题的分析为例。这个差异在我们上 K8S + 容器常有一个痛点（K8S会自动销毁出现故障或问题的 pod），即我们很难通过原有方法分析出业务 Java 内存溢出的原因，这是因为内存一旦溢出，就会导致 K8S 里面的 Pod 节点不用，所以我们无法通过登录容器进行分析、排查，这会对很多开发人员造成一些困扰，因为按照之前有部署模式，一旦业务服务出现问题，至少本地的日志、场景仍然能够保存，通过登录系统是可以查的。但是由于 K8S 对于 Pod 节点进行了管理调度，原 Pod 节点不可用，这个时候该如何进行分析呢?

其实方法并不困难，我们知道开发人员可以使用 jmap，通过 -dump 选项，把 Java 堆中的对象 dum 到本地文件，然后使用 eclipse 的 MAT 进行分析。容器化的方式也是同样的道理，但需要了解的是还是要把容器里面的日志导入 node 节点，或者导出到中心化的存储上去。所以我们可以用这种方式做：在 docker 里面挂接一个 node 节点存储空间（PVC），接着配置一个参数,在 JVM 上通过配置umpPath（路径在你共享的硬盘上），这样就可以实现在内存溢出后，把这些信息都 dump 到你的享硬盘的路径下。那么再出现内存溢出问题时，即使你的 Pod 节点不可用了，但是日志还是能够非清晰的在 node 本地并留存。

配置文件管理

刚刚我们讲到了三个方面的问题：存储的日志收集，状态的服务，以及 Java 内存溢出排查。还有一分重要的内容就是程序的相关配置如何在 K8S 中管理。对这些配置的内容，我要如何去进行管理?

这里有两种模式，在 K8S 架构里，第 1 种模式是把相关的普通文件配置信息，交给 K8S 的 Configmap 进行处理。第 2 种模式则是把加密类型数据信息，交给 K8S 的 Secret。

Configmap 和 Secret 使用的其实是同一套存储机制，只不过 Secret 相较于 Configmap 而言，通了一些安全性的 Base64 数据加密，所以它的安全性会更高一些。

```
apiVersion: v1
 kind: ConfigMap
metadata:
   name: Mysql-config # ConfigMap的名字, 在引用数据时需要
   labels:
       app: k8smysq;
data:
    MYSQL_HOST: J-database-service # 数据库主机
    MYSQL_PORT: "3306" # 数据库端口
    MYSQL_DATABASE: jesonc_com# 数据库名
```

举个例子，在 K8S 里通过 yaml 语法创建对象的时候，如果我创建一个 Configmap，它的作用是创一台数据库相关配置文件的相关信息，我们知道数据库的配置文件信息包含主机名称、主机端口、以数据库名称，这些都是数据库在进行初始化安装的时候，我们通常所需要配置的一些相关信息。这些息我们可以写到 Configmap 里，然后通过 Configmap 传递到你的容器里，去做初始化配置的生成。

```
apiVersion: v1
kind: Secret
metadata:
    name: Mysql-secret
    labels:
        app: k8smysql
data:
    MYSQL_ROOT_PASSWORD: jesonc.com  //root口令
    MYSQL_USER_NAME: jesonc 普通用户名
    MYSQL_USER_PASSWORD: # 普通用户口令（"dbuser"）
```

第 2 种方法则涉及到安全性较高的一些内容，我们可以将其写到一个单独的 Secret 对象里去。比如我们同样要去创建一台 MySQL 数据库时，那么我们可以把数据库的口令、密码、数据库用户名称和据库用户名这些信息，放到 Secret 里，这样它的安全性就更高。

开发库配置自适应

除了刚讲到的应用程序中一般的配置文件和需要加密的安全文件，第 3 种模式是开发库的配置自适应我用一个例子来为你讲解。比如我们经常在 Docker 里面跑 Java 服务，常常要设置 JVM 参数，如果们使用的是 OpenJDK 9 +的版本，JVM 就可以自己去检测运行在容器中的相关资源，并且动态地去行限制。那么我们就可以直接在设置 JVM 参数的时候，用 UseCGroupMemoryLimitForHeap 这个数进行设置，JVM 会根据容器的资源限制，自适应地设置容器的JVM里堆和栈的大小。这样我们就以不用传统的模式具体作固定绑定。

刚刚讲到的这些问题，是从非 K8S 架构迁移到 K8S 架构里，业务上需要先关注的一些问题。

二、迁移通用步骤

接下来我要为你介绍，从非 K8S 架构迁移到 K8S 架构里通用的迁移步骤。

1. 步骤一：考虑如何将原有应用封装容器。

2. 步骤二：考虑在容器里面，如何对需要的数据进行持久化。如果存储到本地的存储块或中心存储，个时候就需要通过创建 PV 或者 PVC 来做持久化的数据存储。

3. 步骤三：程序中的配置信息管理，我们可以把这些配置信息通过 K8S 的 ConfigMap 或 Secret 进配置和管理。

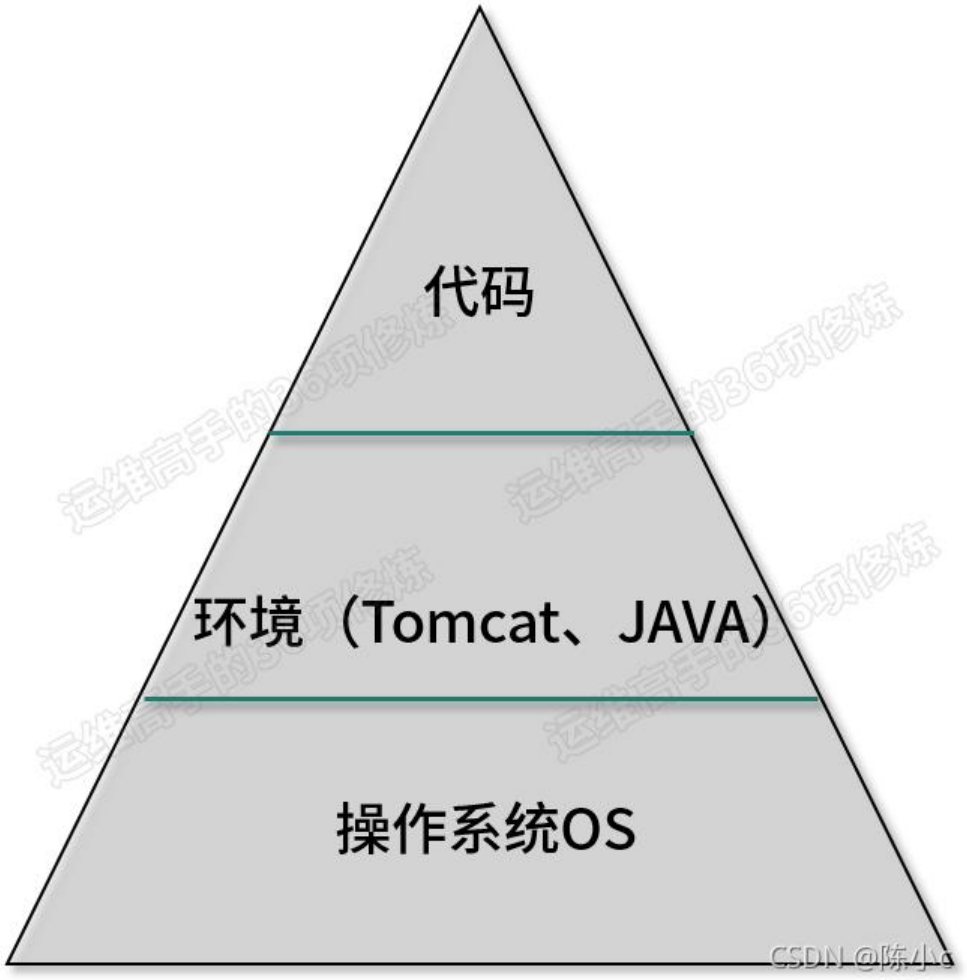
4. 步骤四：封装 Pod，容器在 K8S 里并不是一个最小化的个体，其最小化的个体是 Pod 节点。Pod 能由多个容器（container）组成，这个时候我们就要考虑如何将容器封装到 Pod 里。

5. 步骤五：配置 Service。让 Pod 能以 service 的方式对 K8S 内部集群提供访问。

6. 步骤六：配置 ingress。如果你需要对外部服务提供访问，比如一个网站服务需要对用户提供访问这个时候我们就需要考虑入口网关的配置，在 K8S 里配置一个 ingress 对象，通过它实现外部服务问。

将非容器的应用迁移到 K8S 容器化的结构下，通常的步骤就是这样，接下来我们对每个步骤进行具讲解。

步骤一：封装应用容器



第 1 步需要将应用封装到容器里，这里我画了一张图，它把业务、环境和系统整体打包到一个容器镜

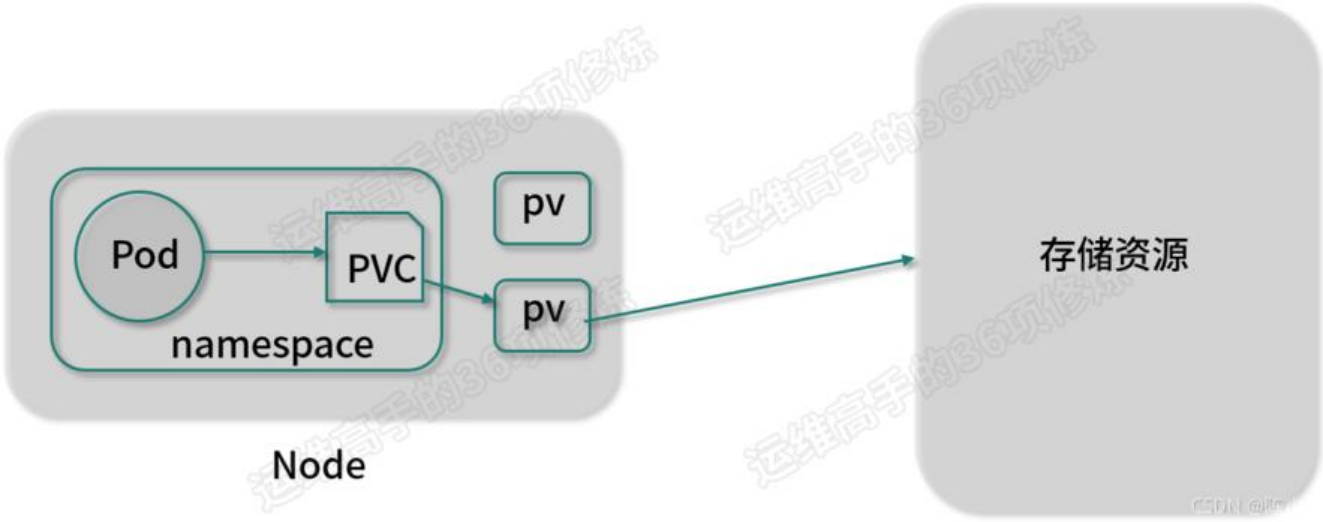
里面去。最底层还是一个 OS 操作系统层（可以选择 CentOS、Ubuntu、Alpine），Alpine 是官方小化的 Linux 镜像。

第 2 层是程序代码所需要依赖的一些环境。如果是 Java 服务的话，这里就需要依赖 Java 的解析器比如 OpenJDK 和 OrackeJDK，Servlet 容器是 Tomcat 还是其他的服务。

最上层就是代码。

这些所有内容整体构成需要打包成容器镜像.

步骤二：PV/PVC 持久化数据

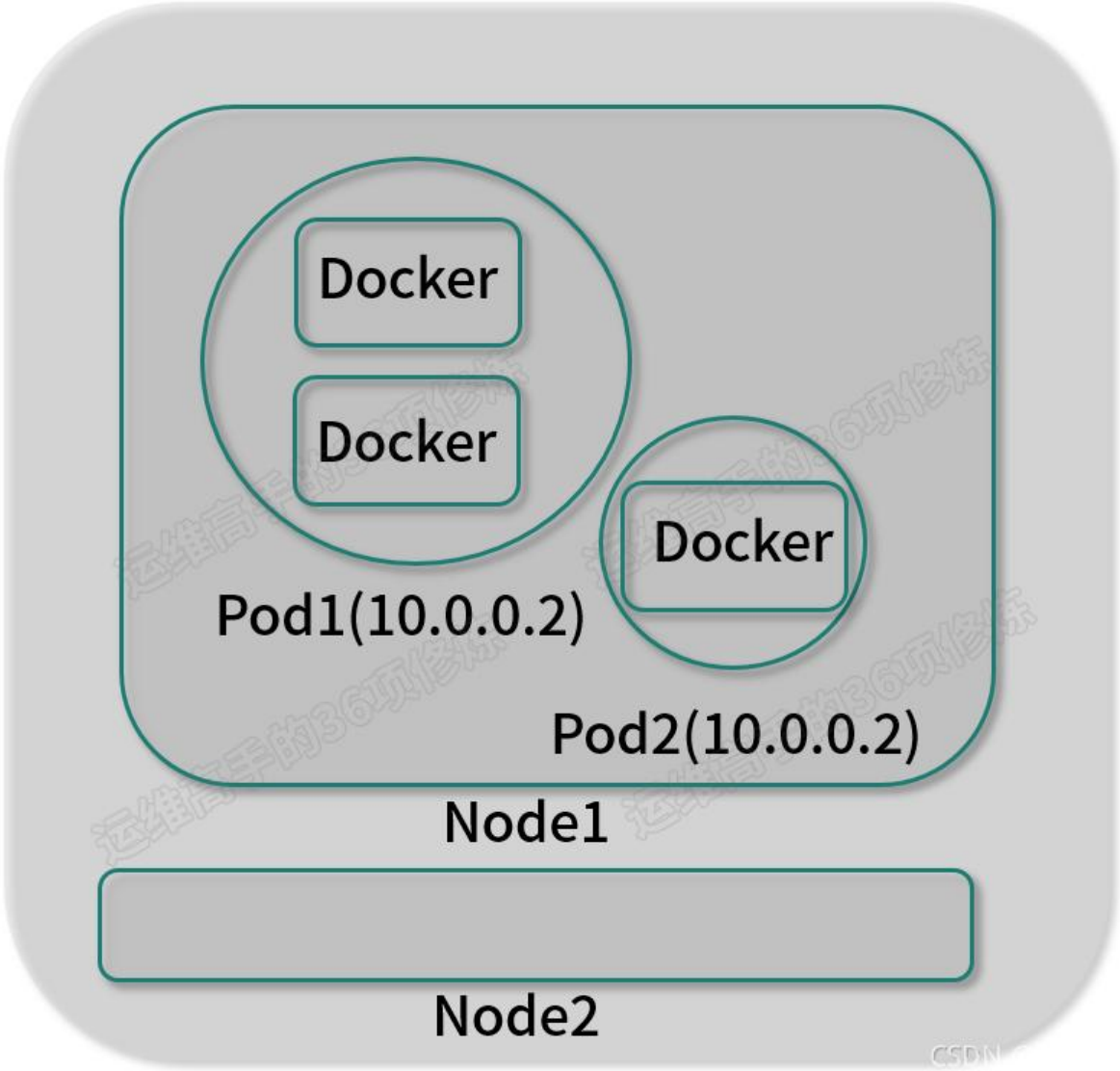


第 2 步就是我刚讲到的 PV 或者 PVC 的数据持久化。我们知道容器中存取的都是临时数据， Pod 的部数据可能会因为重启而丢失，所以我们要把这些需要共享到存储块的内容，比如说网站服务里用户上传的一些图片文件，或者是需要多个程序共享的一些数据文件和日志文件，都可以通过集中化的方存储到容器外部，这通常需要通过 PV 或者 PVC 来建立数据持久化。

步骤三：ConfigMap/Secret 配置

第 3 步就是应用程序所需要的相关配置，它可以通过 K8S 的 ConfigMap 或 Secret，把这些应用程所需要的一些配置，以 key-value 的形式传递到 Container（容器）里面 。

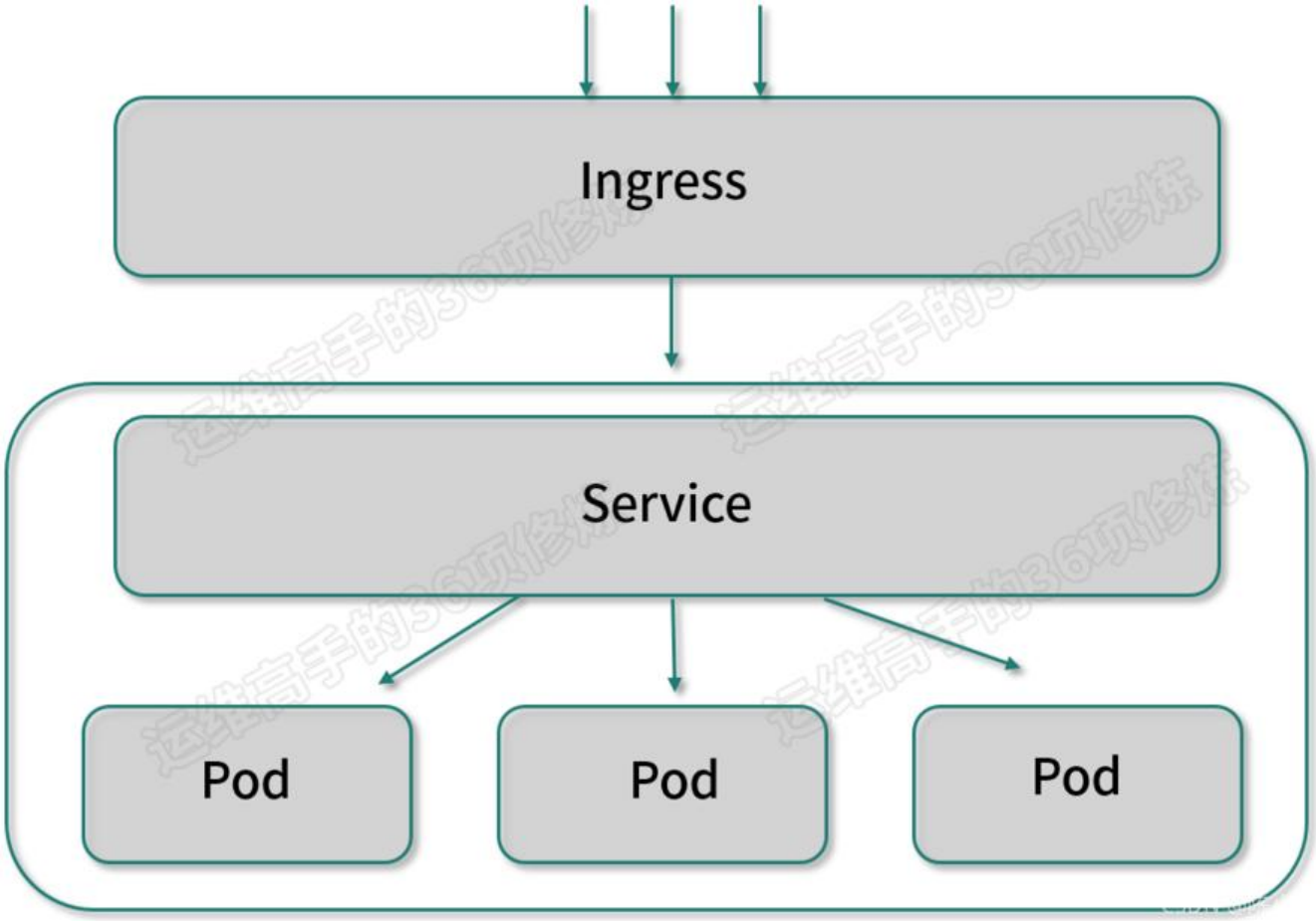
步骤四：封装 Pod



第 4 步是封装 Pod，考虑容器如何容器封装到的 Pod 里。我们需要考虑这样几个因素，首先要避免单 Pod 故障，所以在创建 Pod 时一定要考虑多个副本。其次，Pod 一般都有 Liveness 和 Readiness 健康检查配置，所以我们在 Pod 里配置对象的时候，先要提前定义好，以便 K8S 能够更精确的检测你的 Pod 是否正常。另外，我们需要了解这个业务的一些耦合性，通常一个 Pod 里面是一个容器，是也有对耦合性要求非常高的，举个例子，一些游戏类客户，有可能会在一个 Pod 里启用两个 Container，也就是两个容器，一个跑业务，另外一个跑本地缓存。这样就可以做到两个容器在一个 Pod 里共享 namespase。

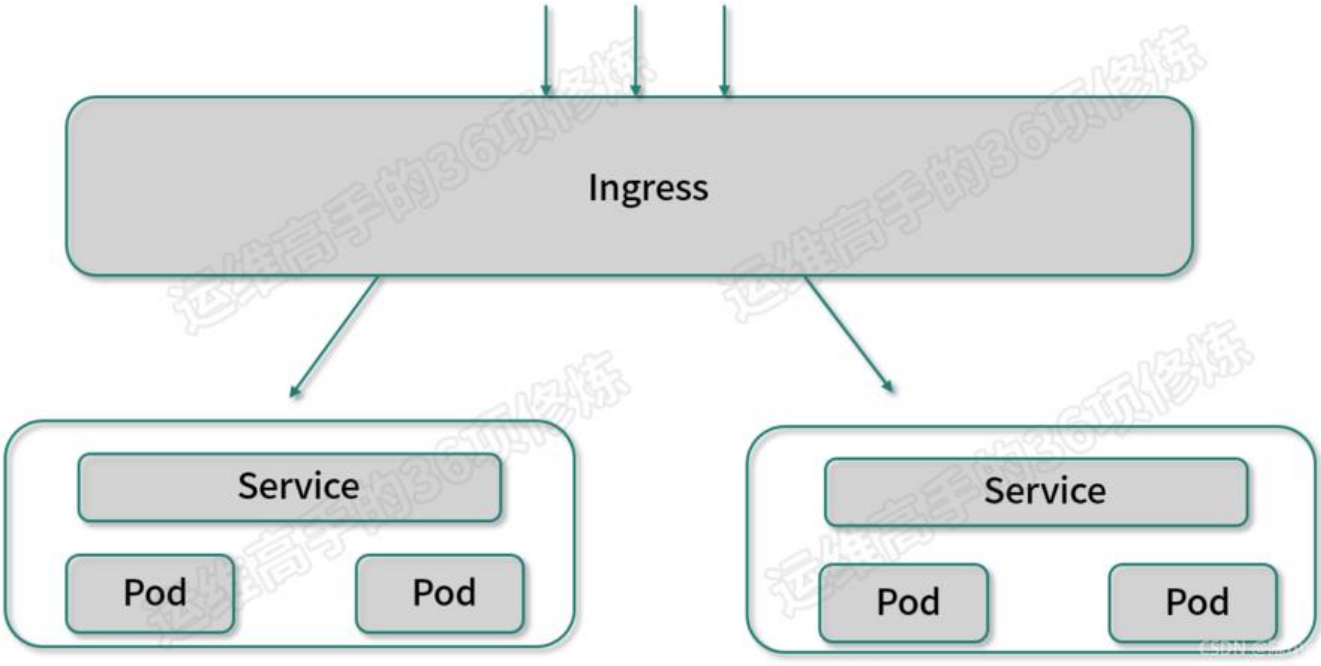
对于数据要求紧密的业务，我们可能需要这么做。所以在 Pod 、容器、业务这三个方面，需要提前好规划。另外，我们需要来根据实际情况选择生成 Pod 的 Controller，包括 Deployment、DaemoSet、StatefulSet、Job、CronJob 等相关类型，你可以去了解 K8S 相关的 Controller。

步骤五：配置 Service



第 5 步是配置 Service。Service 使得 Pod 可以提供内部访问方式，对外能够抽象成一个对象，也就是 Service 这个对象。集群内部可以直接使用 Service Name 进行通信，这就是 Service 的作用。我们一下这个图。

步骤六：配置 Ingress



第 6 步是配置 Ingress。Ingress 可以暴露内部的服务给外部，所以它常承担是 7 层反向代理和负载

衡的作用。当把内部服务暴露到外部的时候就会用到 Ingress 这个对象，

在做完 Ingress 后，把服务迁移到 K8S 集群的步骤基本上就已经完成了，这是一个通用的6个步骤。