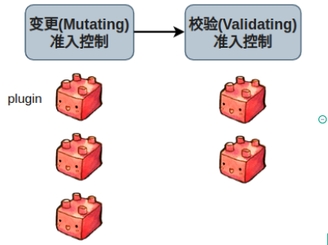
声明式API

命令式与声明式API

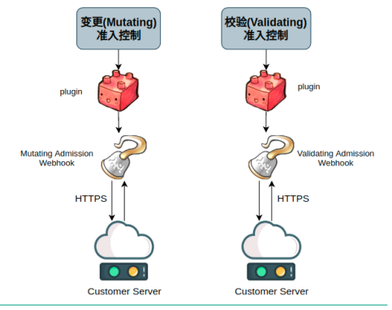
- 命令式与声明式编程
  - 对于命令式编程而言，我们会告诉执行体要具体怎么做，例如一条汇编指令: movl %eax, %ebx，将eax寄存器中的值移动到ebx寄存器中
  - 而对于声明式编程而言，我们通常不会关心执行体到底是怎么做的，只关心做什么，例如一条SQL语句: select * from hugo，从hugo表中取出所有数据，无需关心这是从内存取出来的，还是从硬盘中取出来的
  - 从上面的小例子中可以看出，声明式编程要比命令式编程更加高级，同时实现也更加复杂
- 命令式与声明式API
  - 换到API这个领域，命令式API会告诉执行体具体的行为，例如systemctl restart/start/stop mysql，需要用户去指定到底是重启还是启动还是暂停MySQL服务的执行
  - 声明式API则会告知执行体我们需要的最终状态是什么
- Kubernetes声明式API
  - 在创建或者更新资源时，我们可能会使用 kubectl create/replace xxx命令，使用该命令的前提就是我们已经得知了资源的当前状态
  - 例如kubectl replace xxx，既然使用的是replace，那么当前资源一定已经存在，我们想要做的是更新资源，很明显这是一种命令式API
  - 在kubernetes中，kubectl apply 表达的正是声明式API的思想: 我不关心k8s是创建还是更新资源，反正资源最终需要达到的状态我已经通过YAML文件给你了，你给我把相关资源整到那个状态就行
- k replace 与 k reply
  - replace可以理解是PUT方法，而apply可以理解成PATCH方法
  - **在kubernetes中，这两者最重要的区别在于k replace每次只能响应一次写请求，否则就会产生冲突的可能。而k apply则能够一次处理多个写操作，自身具有Merge的功能** 一个哈希表，使用Lock锁住整个数组(replace)，和使用RWLock(apply)锁住数组中的一个元素

Admission Controller

- Admission Controller，即准入控制器，是 API Server用于拦截请求的一种手段，可以对资源对象进行校验和修改，非常类似于Web框架中的hooks，例如Django和Go Web框架的Middleware，Flask框架中的before_request
- Admission Controller大体可以分为两部分， mutating admission plugins 以及 validating admission plugins，前者对资源对象进行修改，后者则对资源对象进行校验



- 准入控制器由一系列的插件所构成，开发者可以添加或者删除某些插件
  - kube-apiserver -h | grep enable-admission-plugins
  - 可通过上述命令查看哪些插件是默认启用的
  - 例如Mutating Controller中的 DefaultTolerationSeconds，将Pod的 tolerationSeconds 修改为默认的5分钟时间
- 动态准入控制
  - k8s为了使得准入控制更加的灵活，内置了一组称之为admission webhook的准入控制器，用户可自行编写一个Web Server，结合admission webhook来实现自己的变更或者是校验逻辑
  - admission webhook的出现使得用户不必将自己编写的插件放入至API Server并且重新编译，属于一种热拔插的机制



Admission Controller与声明式API

- 在前面的 Admission Controller提到过，我们可以通过部署 Mutating Admission Webhook 来动态地修改用户提交至API Server的资源对象，例如Pod，Deployment，我们甚至可以向所有的Pod对象中均添加一个sidecar容器，来做一些其它的事情
- 那么，如何将sidecar容器的相关配置注入到原有的Pod中呢?
  - Customer Server硬解析用户提交的Pod声明，将新增的内容手动地添加至资源对象中 这个方式是不是和命令式编程很像?
  - 使用Kubernetes提供的PATCH方法注入，将新增的内容自动地添加至资源对象中 这个方式是不是和声明式编程很像?
- 所以，Kubernetes所提供的声明式API进一步地减轻了开发者开发的负担，已经有了内置实现，为什么还需要额外造轮子呢?
- **声明式API最大的优点就在于我们无需关心原来的配置文件内容，Merge的过程完全交由Kubernetes处理**

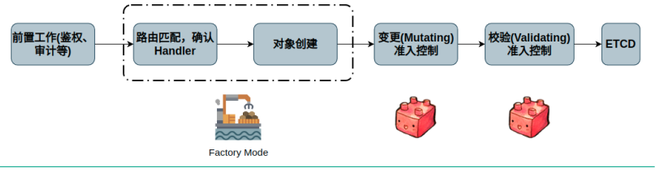
API对象

- 当我们使用 k apply -f 创建或者是更新一个资源时，kubectl将发送一个符合RESTful API风格的HTTP请求至API Server，API Server将创建相关的资源对象，并将其序列化保存在ETCD中
- API资源
  - 一个API资源最基本的信息包括资源的名称、组以及版本信息

```
apiVersion: apps/v1
kind: Deployment
```

  - 如上所示，其中"apps"表示组(Group)，"v1"为版本(Version)，"Deployment"为资源名称
  - 获取当前k8s集群所支持的资源 k api-resources -o wide
  - APIGroup(组)
    - 组的概念其实就是分类，k8s将不同的资源对象分配至不同的组中 从上图中可以看出，核心资源，例如Pod，Node，Service等核心资源对象是没有APIGroup的
    - 当我们在声明一个资源，例如StatefulSet，不知道该资源属于哪个APIGroup时，grep一下就好 k api-resources -o wide | grep StatefulSet，结果表明StatefulSet属于apps组
  - APIVersion
    - k8s能够向后兼容，为了实现这一点，相关的资源都会添加版本号，这和Web服务向后兼容没有什么区别
    - 获取当前k8s集群所支持的版本 k api-versions
    - 如果想要知道某个资源到底有哪些可用版本的话，需要执行两条命令 为什么不能提供"获取某资源所有的可用版本"这样的API呢?

总之，这是个比较恶心的地方



- API对象的创建流程