第13章 Kubernetes

Author： 中山大学 17数据科学与计算机学院 YSY

https://github.com/ysyisyourbrother

Kubernetes是对docker的管理

容器 Container

内核系统调用的虚拟化

隔离机制 isolation

chroots：切换根目录。根目录可以变化。有多个容器在跑的时候，每个容器都有自己的根目录，看不到别人的。

namespaces：命名空间。每个containers中的进程pid都是从1开始的，说明只能看到自己空间里的process。

cgroups：限制、记录、隔离进程组所使用的物理资源（包括：CPU、memory、IO等），为容器实现虚拟化提供了基本保证，是构建Docker等一系列虚拟化管理工具的基石。

打包 packaging

- 不需要外部依赖
- 不会出现dll链式依赖
- 可以在很多环境运行

很不安全，因为公用同一个内核。在公有云环境下会部署在虚拟机环境下使用。

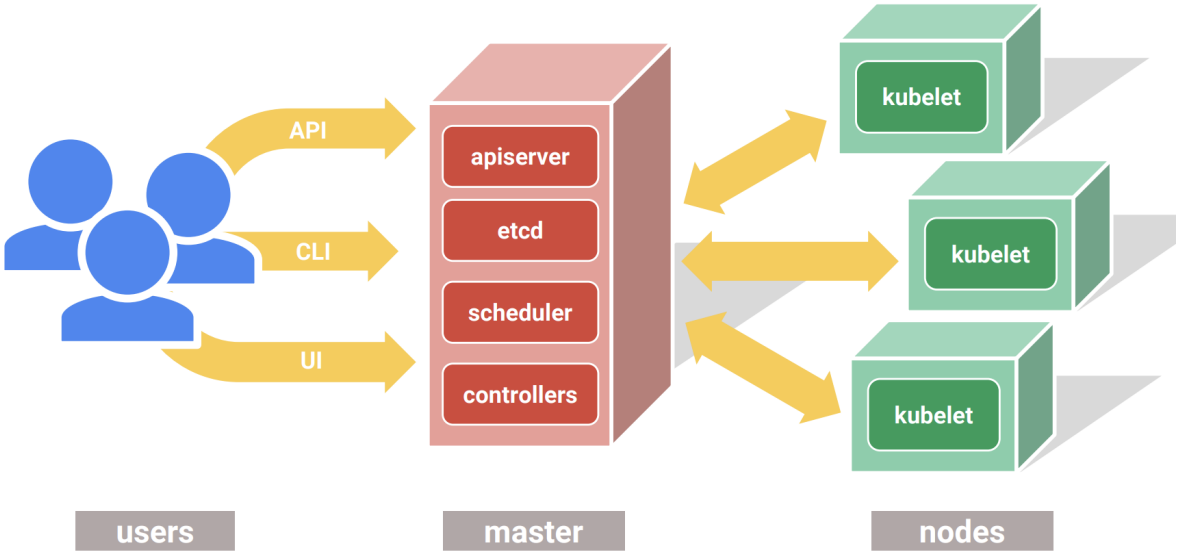
创建镜像

```
% cat Dockerfile
 FROM node:4.4  # 基镜像 镜像名+版本号
EXPOSE 8080  # 服务对外提供的端口号
COPY server.js . # 往基镜像 copy本地的代码
CMD node server.js # 跑程序
% docker build -t gcr.io/mohr-dev/hello-node:v1 .
[log spam]
% docker run -d -p 8080:8080 --name hello_tutorial gcr.io/mohr-dev/hello-node:v1
% curl http://localhost:8080/
Hello World!
```

-d：跑的程序是后台程序

-p：端口映射，服务端口映射主机端口

Kubernetes



apiserver：接口，可以和集群交互

etcd：容器的信息，集群元信息

scheduler：分配调度容器

controllers：控制容器，比如update kill

kubelet：基本控制容器，如开关容器，生命周期等

kuproxy：代理

API：通过接口和集群交互

CLI：命令行工具，和集群交互

UI：用户界面，直接拖拽

部署集群：

1. Setting up the cluster

- Choose a cloud: GCE, AWS, Azure, Rackspace, on-premises, ...
- Choose a node OS: CoreOS, Atomic, RHEL, Debian, CentOS, Ubuntu, ...
- Provision machines: Boot VMs, install and run kube components, ...
- Configure networking: IP ranges for Pods, Services, SDN, ...
- Start cluster services: DNS, logging, monitoring, ...
- Manage nodes: kernel upgrades, OS updates, hardware failures...

1. 选择部署的云平台
2. 选择节点的操作系统
3. 软件安装
4. 部署网络：给pods services SDN 分配IP地址
5. 给集群提供一些服务：DNS分配服务名，日志，监控运行的指标（服务响应时间
6. 管理集群：升级操作系统、内核等

使用集群：

2. Using the cluster

- Run Pods & Containers
- ReplicaSets & Deployments & DaemonSets & StatefulSets
- Services & Volumes & Secrets & Autoscalers

1. 运行的容器 pods是多个容器的集合
2. 部署的方式（脚本）。比如 DaemonSets：杀不掉的进程 StatefulSets：有状态的运行
3. 描述的文件：services：外界访问的服务 Volumes：存储系统 Secrets：访问的密码 Autoscalers：弹性伸缩服务数量

Kubernetes：a Cloud OS

Perhaps grandiose, but attempts at "Cloud OS" primitives:

- **Scheduling**: Decide where my containers should run
- **Lifecycle and health**: Keep my containers running despite failures
- **Scaling**: Make sets of containers bigger or smaller
- **Naming and discovery**: Find where my containers are now
- **Load balancing:** Distribute traffic across a set of containers
- **Storage volumes**: Provide data to containers
- **Logging and monitoring**: Track what's happening with my containers
- **Debugging and introspection**: Enter or attach to containers
- **Identity and authorization**: Control who can do things to my containers

Debugging：在线调试debug

Workload Portability

平台可移植性 写一次，在哪里都可以跑

Goal: Write once, run anywhere*

Don't force apps to know about concepts that are cloud-provider-specific

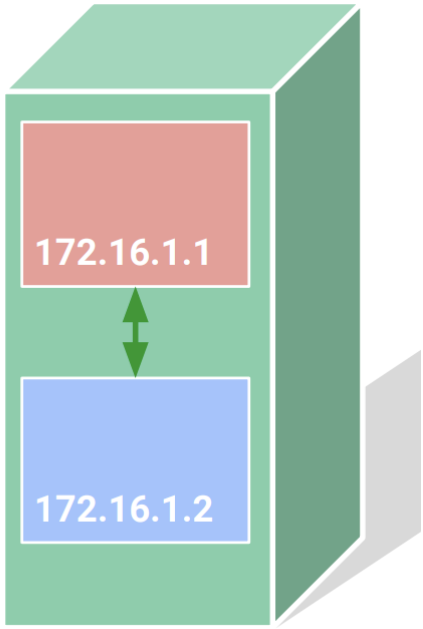
- Network model
- Ingress
- Service load-balancers
- PersistentVolumes

自己设计了各种的系统，不需要依赖任何平台：

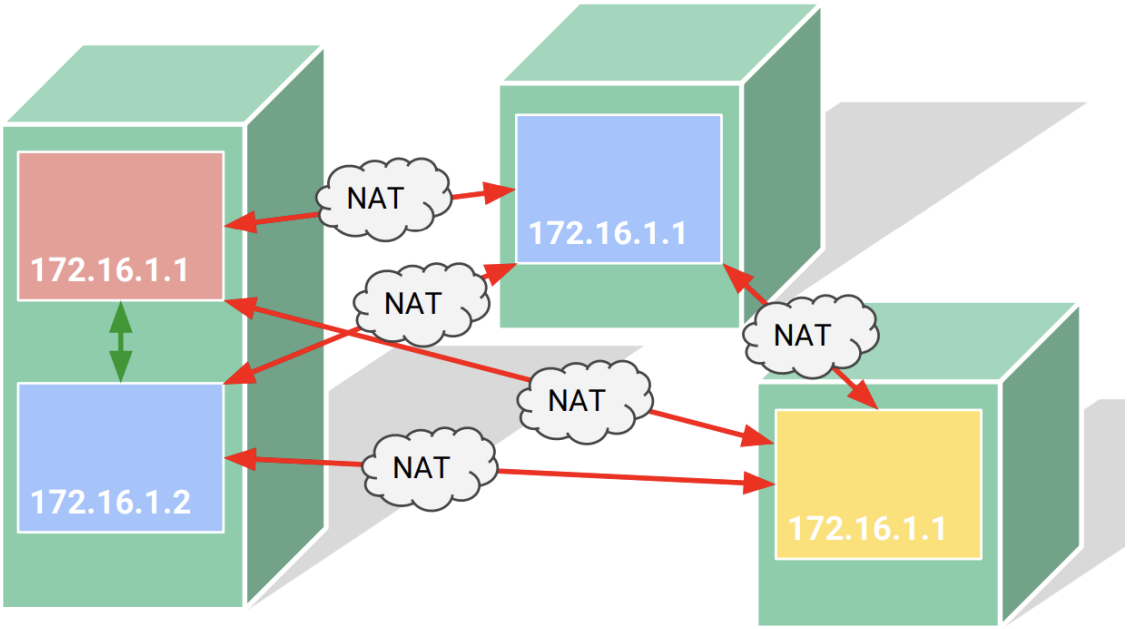
1. 网络模型
2. 网关组件
3. 服务均衡组件
4. 持久挂载卷 文件系统

Networking 网络部署

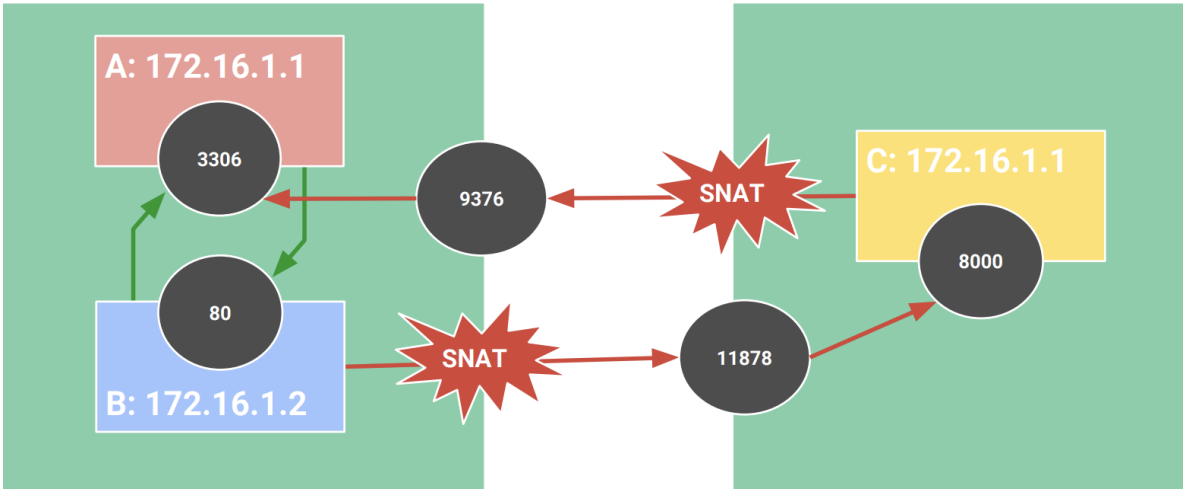
如果docker部署在一个子网中，docker是可以相互通信的



但假如不是在一个子网中的话无法通信，要使用NAT



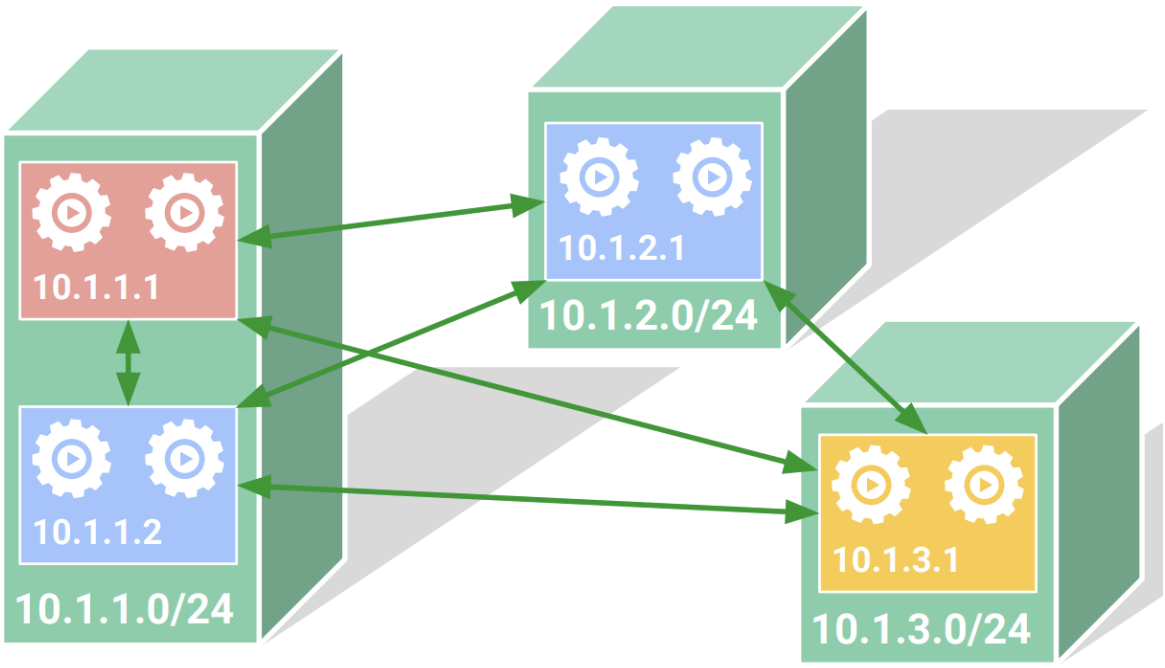
端口映射 port mapping:



内网地址和端口号转换成合法的公网地址和端口号,建立一个会话,与公网主机进行，但这种方法效率比较低。

Kubernetes networking

正确的解决方法：构建覆盖网络，覆盖所有的在不同子网的docker。每个节点都有自己的网络地址



Pods

Small group of containers & volumes

Tightly coupled

The atom of scheduling & placement

Shared namespace

- share IP address & localhost
- share IPC, etc.

Managed lifecycle

- bound to a node, restart in place
- can die, cannot be reborn with same ID

pods是一组容器的集合，容器耦合在一起，共享一个命名空间

pods是调度的基本单元

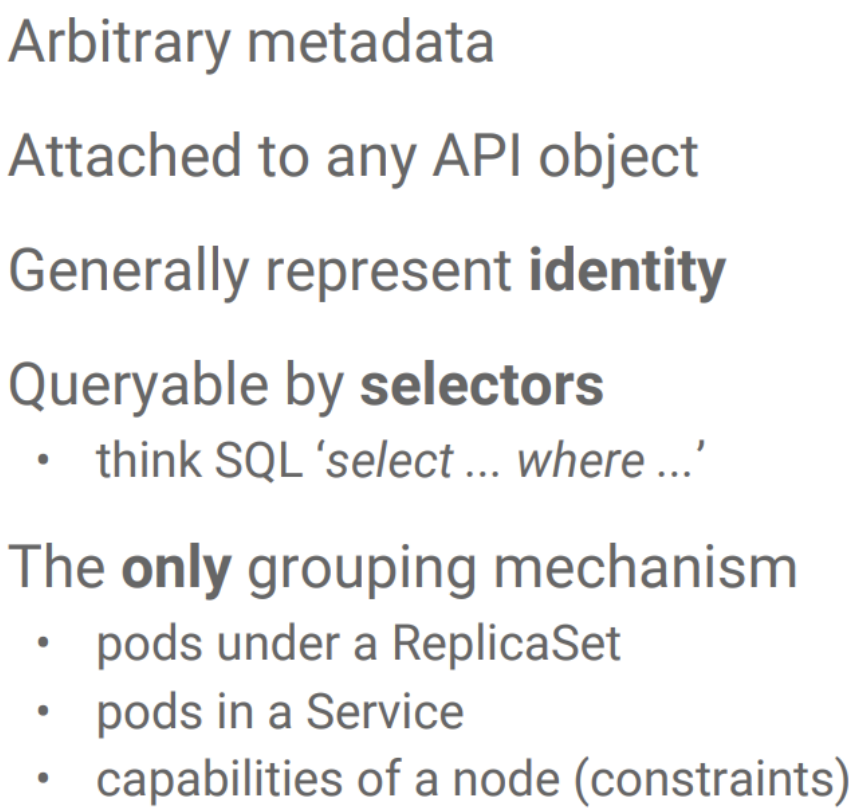
pods的生命周期由kubernetes管理

不能创建两个相同id的pods

Volumes 持久化存储

在pod中共享某一个文件

Labels

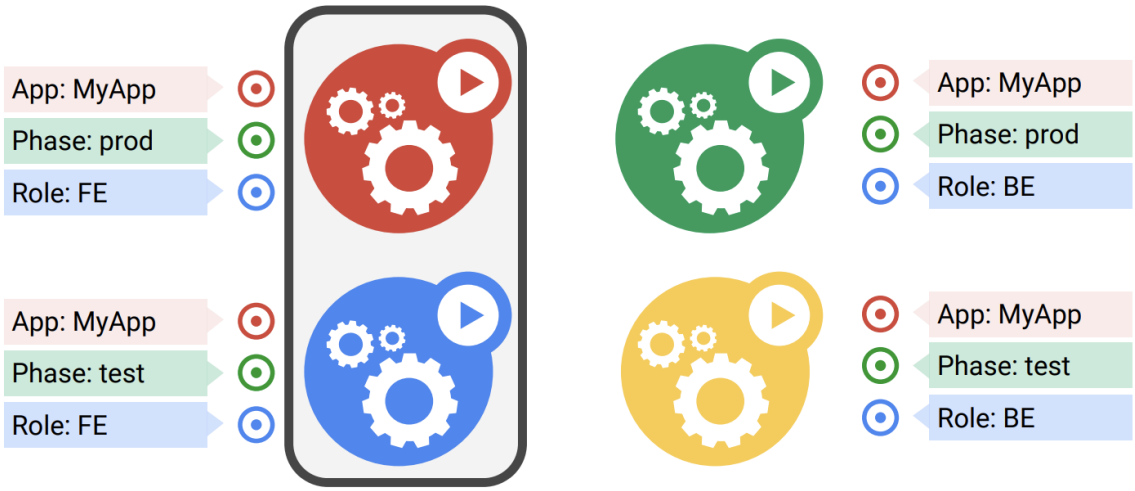


每个资源对象都会有一个标签

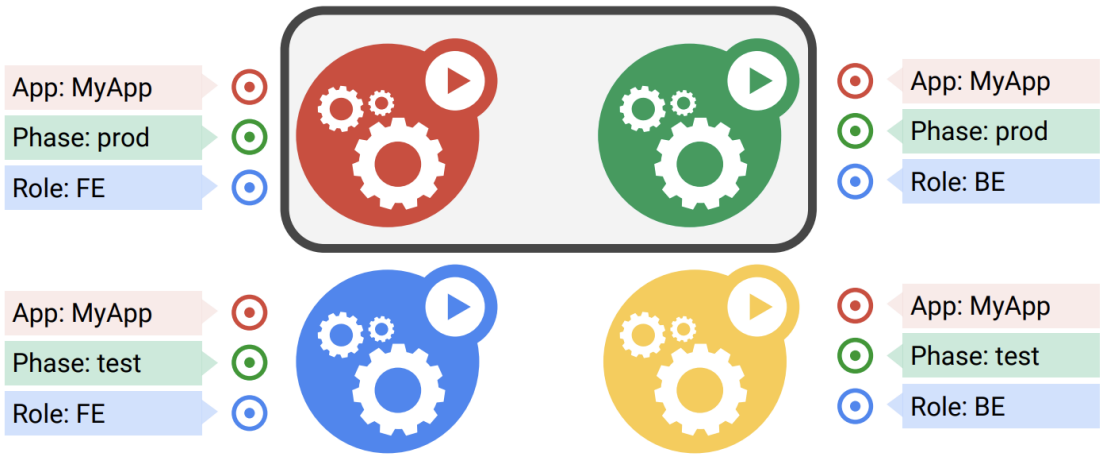
把标签告诉API来访问资源

然后通过选择器选择对应的资源， 类似数据库的select语句

Selectors



App = MyApp, Role = FE

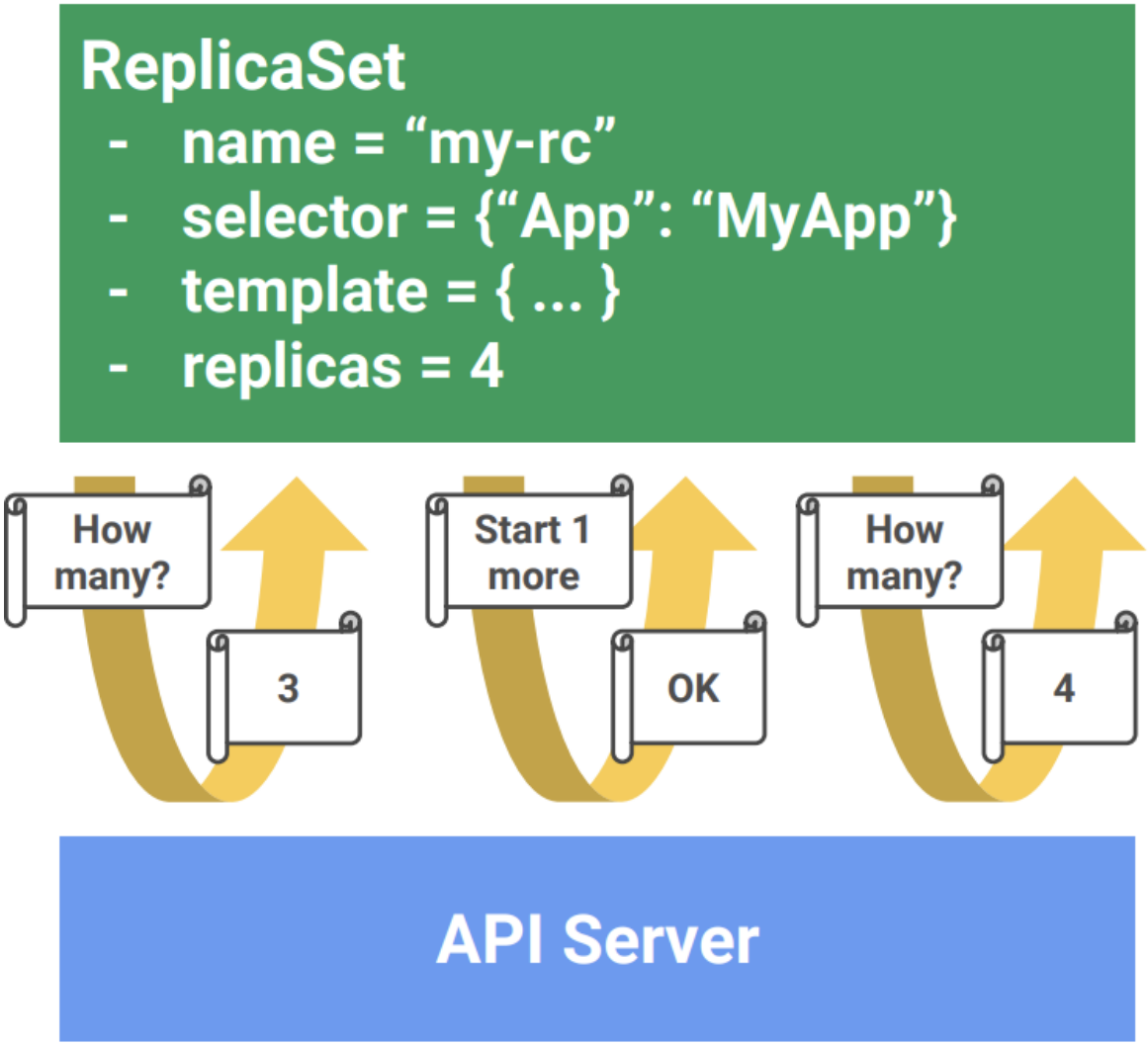


App = MyApp, Phase = prod

APP名字 Phase 阶段 Role FE前端 BE 后端

Replication

replicaset也是一种资源类型，同时复制4次


ReplicaSet
- name = “my-rc”
- selector = {“App”: “MyApp”}
- template = { ... }
- replicas = 4
How many?
3
Start 1 more
OK
How many?
4
API Server

A simple control loop

Runs out-of-process wrt API server

One job: ensure N copies of a pod

- grouped by a selector
- too few? start some
- too many? kill some

Layered on top of the public Pod API

Replicated pods are **fungible**

- No implied order or identity

control loop：控制数量一直是4个

Drive **current state** -> **desired state**

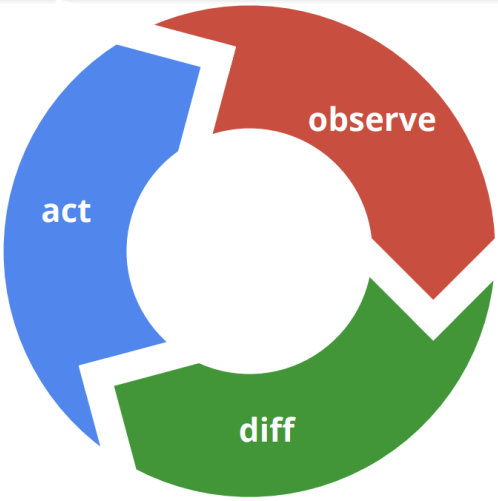
Act independently

APIs - **no shortcuts** or back doors

Observed state is truth*

Recurring pattern in the system

Example: ReplicaSet



如果当前replicate是3，但声明的时候是4，就执行操作

Services

对运行实例比如pods进行外包装，分配IP地址让外部访问

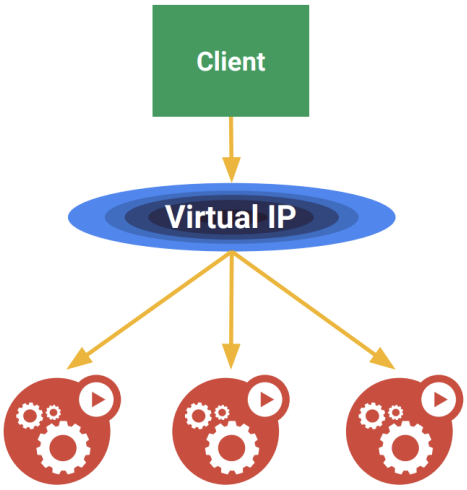
A group of **pods that work together**
- grouped by a selector

Defines access policy
- "load balanced" or "headless"

Can have a stable **virtual IP** and port
- also a DNS name

VIP is managed by *kube-proxy*
- watches all services
- updates iptables when backends change
- default implementation - can be replaced!

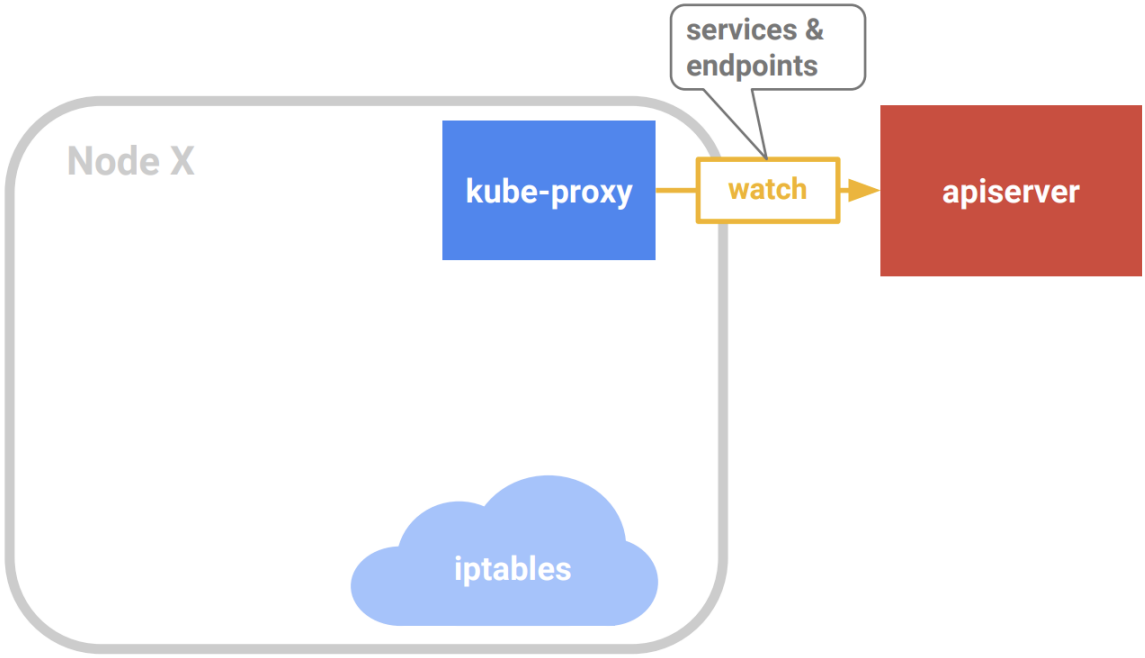
Hides complexity



client要访问资源的时候，需要先通过虚拟IP找到Services，或者用DNS换成Services名字，才能访问资源。

iptables kube-proxy 了解

用来做网络的代理



External services

Services VIPs are only available **inside** the cluster

Need to receive traffic from “the outside world”

Service “type”

- NodePort: expose on a port on every node
- LoadBalancer: provision a cloud load-balancer

DiY load-balancer solutions

- socat (for nodePort remapping)
- haproxy
- nginx

Ingress (L7 LB)