入门Redis & 了解云原生

第一部分：Redis快速入门

初步认识

Redis 是一个开源（BSD 许可）的，**内存中的键值对（K-V）存储系统**，它可以用作 数据库、缓存和消息中间件。 它支持多种类型的数据结构，如 字符串（strings），散列（hashes），列表（lists），集合（sets），有序集合（sorted sets）与范围查询 bitmaps， hyperloglogs 和 地理空间（geospatial） 索引半径查询。 Redis 内置了复制（replication），LUA脚本（Lua scripting），LRU驱动事件（LRU eviction），事务（transactions）和不同级别的磁盘持久化（persistence），并通过Redis哨兵（Sentinel）和自动分区（Cluster）提供高可用性（high availability）

你得知道：

- Redis是Nosql型的非关系型数据库（Nosql：Not Only sql）
- Redis是基于内存存储的，速度快，当然也支持数据的持久化
- Redis是在处理客户端请求的时候是单线程的，在处理客户端连接时候使用IO多路复用技术
- Redis其实是支持事务的，但是一次性、顺序性、排他性的执行队列中的命令，是不支持回滚的，缺乏传统ACID事务的隔离性和持久性

安装

Windows： 官方推荐使用WSL2官网指南，当然也可以在Github上下载Zip,解压缩到本地，然后在安装目录执行`redis-server.exe redis.windows.conf`即可临时启动 **Linux & Mac：** 使用包管理器安装即可

> 但待会我们会讲容器技术，这会让Redis的安装使用更加方便😀

常用命令

- 使用select切换数据库：`select [number]`（redis共16个数据库）
- 查看数据库大小：`DBSIZE`
- 查看库内容：`keys *`
- 判断某个值存在：`exists key`
- 存储数据：`set key value`云
- 数据限时：`expire key number\setex key number value`限时多少秒后数据失效，`ttl key`查询还有多久截止
- 查看key类型：`type key`
- 删除数据库：`flushall or flushdb DbName`
- 批量插入、获取：`mset k1 v1 k2 v2...\mget k1 k2 k3...`

五大基本数据类型

不用记，用的时候查就行🐶

- String
  - 拼接key的value和value，并返回新value的长度：`append keyName value`
  - 获取字符串长度：`strlen key`

    - 对数字内容String自增、自减：`incr key , decr key`
    - 任意步长自增：`incrby key number`
    - 截取子串：`getrange key start end`（截取后包括start end所在的值，end为-1则获取所有项，java则不会包含end）
    - 替换子串：`setrange key offset value`（从offset开始将value替换后面的内容）
    - 如果存在则：`setnx key value`
    - 插入一个json字符串：`set user:1 {name:JayChou,id:1}`
- List：所有的List命令都是l开头
    - 插入：`lpush listName value(头插法)\rpush listName value(尾插法)}`
    - 获取：`lrange listName start end , lindex listName index`
    - 移除：`lpop\rpop`：头删\尾删，`lrem listName number vlue`：移除特定集合的特定值的number个
    - 长度：`llen listName`
    - 集合子集：`ltrim listName start end`
    - 移动元素到新集合：`rpoplpush oldlist newlist`:尾删头插
    - 替换：`lset listName index newValue`，将list中index位置的值替换为newValue
    - 插入：`linsert list after|before value1 value2`在value1前|后插入value2
- Set：无序集合
    - 添加：`sadd key value`
    - 查看所有：`smembers setName`
    - 访问大小：`scard setName`云ield value`
    - 获取：`hget\hmget\hgetall\hkeys\hvals`
    - 删除：`hdel set field`
    - 计算长度：`hlen set`
    - 自增任意数：`hincrby set field number`
- Zset：有序集合，按照score排序
    - 添加：`zadd key scores values`
    - 查看：`zrange key start end , zrangebyscore set min max [withscores] [limit offset count`:offset表示元素下标， - count表示展示多少]，zrevrange key start end和zrange是反序的
    - 移除：`zrem zset value`
    - 访问大小：`zcard zset , zcount zset min max`：指定范围内元素个数

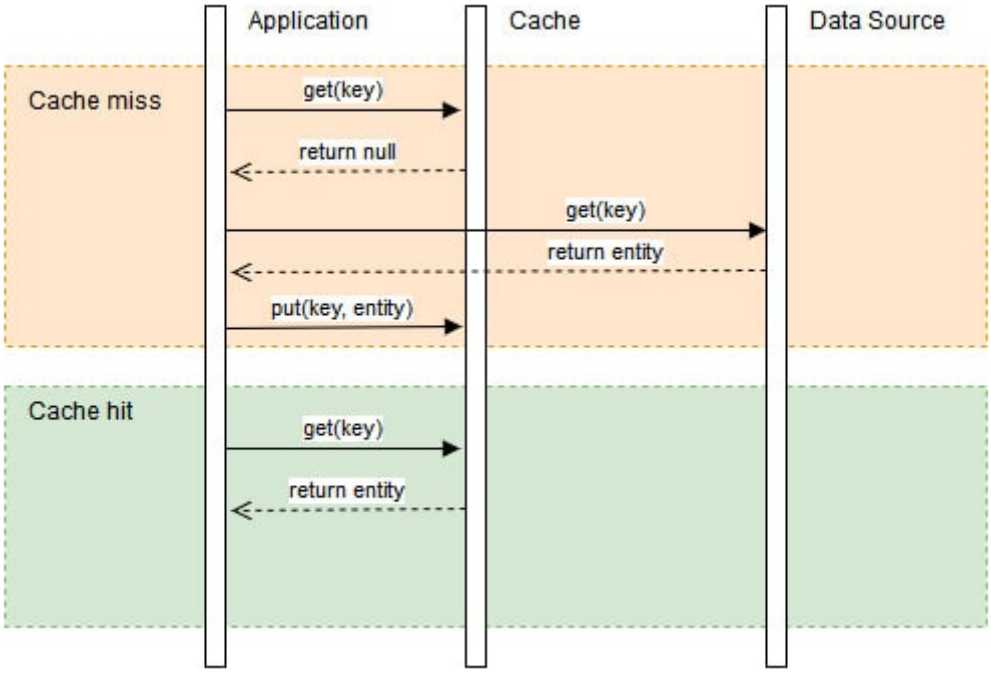
当然还存在一些特殊数据类型Geo、hyperloglog、BitMaps

> 入门的话知道上面这些就好了，但redis还有很多值得深入了解的，比如： Redis事务，持久化，乐观锁，消息订阅机制，集群部署下哨兵模式，主从复制等 这些就需要大家课下去探索啦😉

Java中使用Redis

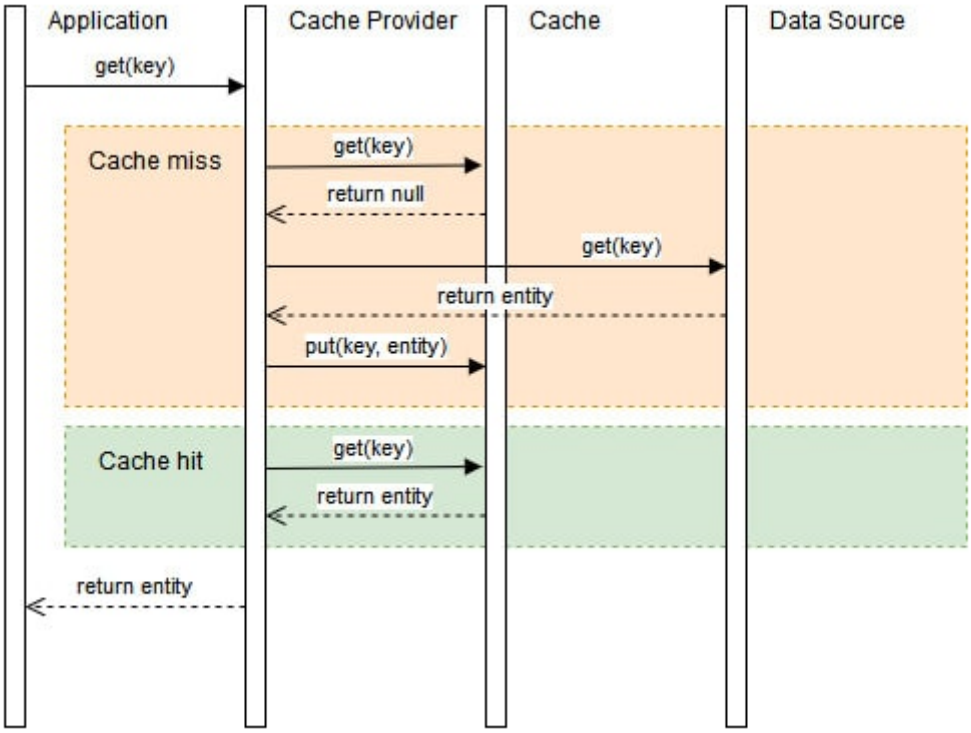
Java有两个比较常用的Redis客户端：Lettuce 和 Jedis，前者是异步的Redis客户端，线程安全（单连接共享的方式），API更加灵活；后者是同步的Redis客户端，本身线程不安全（可以使用jedispool解决），API更加直观。Spring中常使用Spring Data Redis为我们封装的RedisTemplate Getting Start API的使用相对比较直观且方便使用，项目实践中常常会根据业务需求封装合适的Util或者Service 快速掌握和实践也可以看黑马视频教程

缓存策略



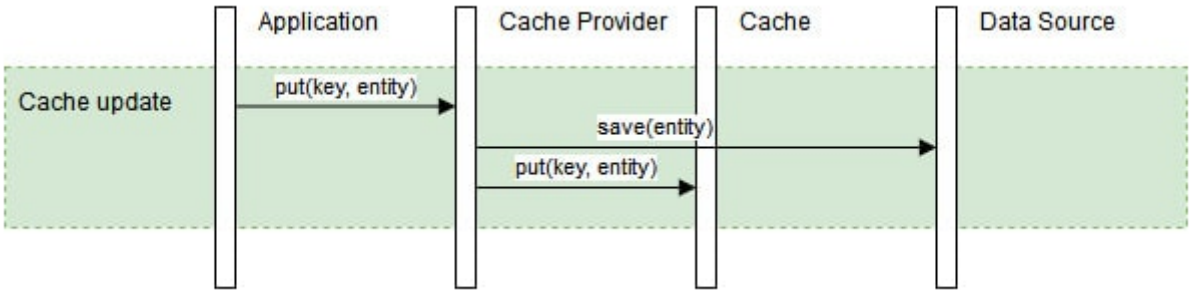
Cache-Aside 策略特别适合“读多”的应用场景，也是用的相对较多的缓存策略。使用 Cache Aside 策略的系统可以在一定程度上抵抗缓存故障。如果缓存服务发生故障，系统仍然可以通过直接访问数据库进行操作。但是第一次访问总是缓存不命中，这种情况还需要别的办法解决，比如可以通过缓存预热

- Read-Through 策略



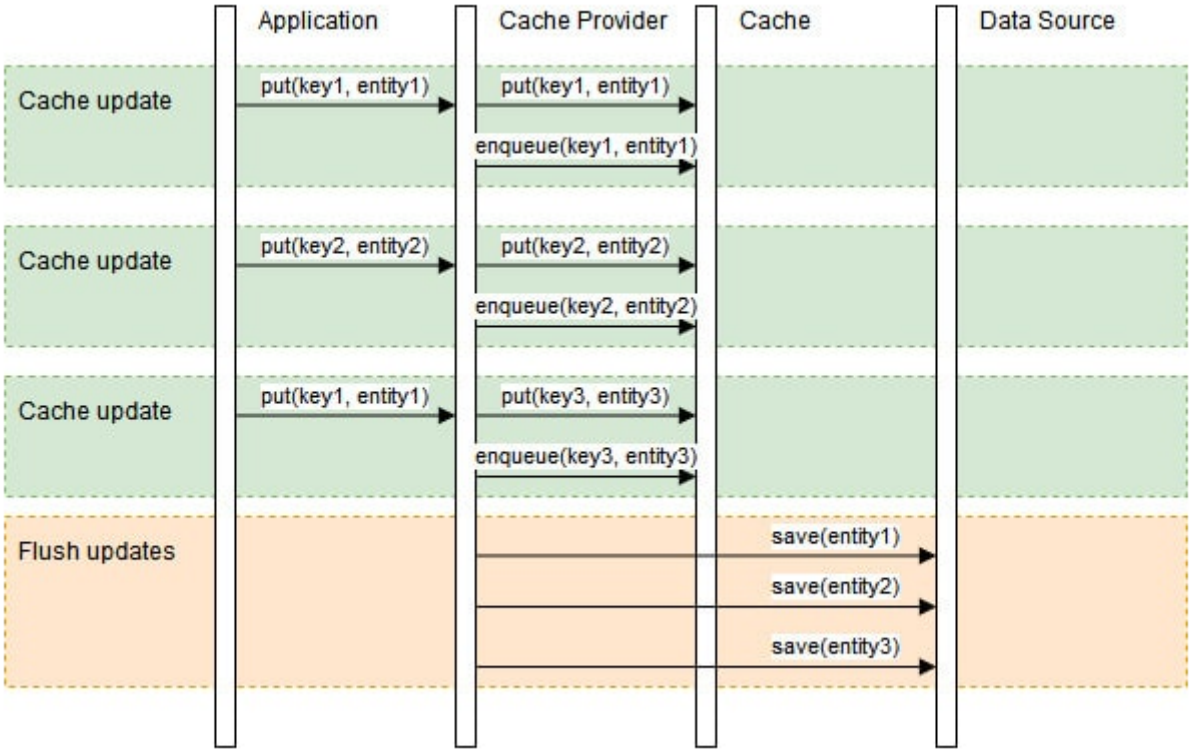
这个跟 Cache-Aside 策略很类似，但是 Read-Through 逻辑上通常是由独立**缓存提供程序**支持，而 Cache-Aside 是直接跟数据源打交道。所以说其实就是 Cache-Aside 的封装

- Write-Through 策略



配合 Read-Through 使用

- Write-Behind 策略



该策略可以理解成把缓存当主要数据源，通过异步定期flush的方式持久化到数据库。不过这可能因为某些原因造成最后一次写入的数据丢失。

缓存三兄弟

缓存穿透： 请求的数据在数据库和缓存中都不存在，这样的请求永远打到数据库上

缓存击穿： 某个热点key缓存突然失效，导致大量请求瞬间打到数据库

缓存雪崩： 大量key同时失效，或redis宕机，导致大量请求打到数据库

实现功能

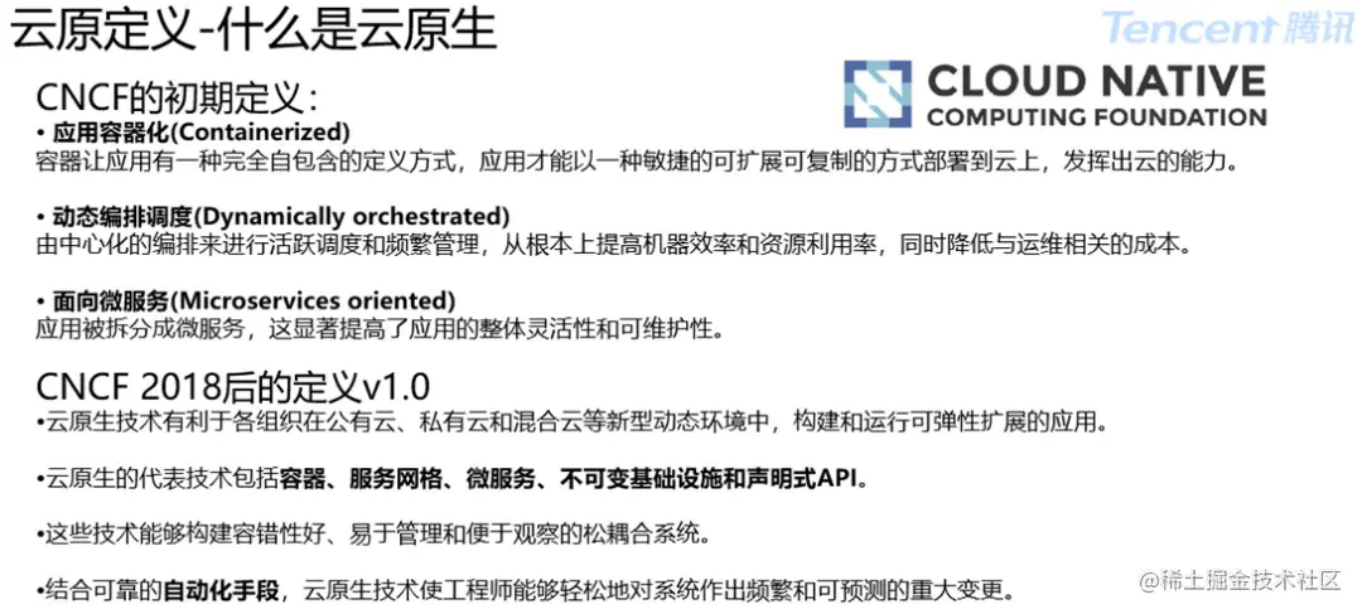
后面大家做项目来实现功能的时候，如果遇到以下场景，希望可以想到Redis：

- 缓存（建议）
- 消息队列（不太建议）

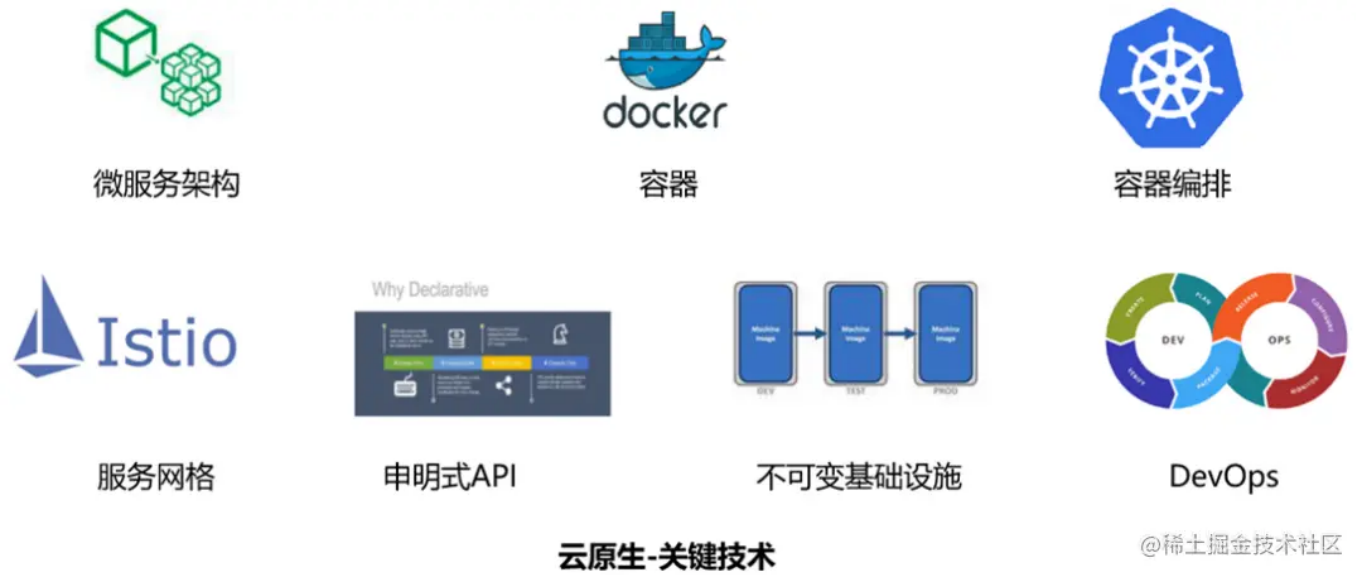
- 延迟消息（不太建议）
- 排行榜（建议）
- 计数器（建议）
- 分布式ID（可以）
- 分布式锁（建议）
- 地理位置应用（建议）
- 分布式限流（可以）
- 分布式Session（建议）
- 布隆过滤器（建议）
- bitmap状态统计（可以）
- 共同关注（建议）
- 推荐关注（可以）

第二部分：了解云原生

云原生是什么



云原生的定义，业界也是“百家争鸣”各持观点，从技术视角理解云原生会相对清晰



关键技术

微服务： 一种用于构建应用的架构方案。将一个复杂的应用拆分成多个独立自治的服务，服务与服务间通过"高内聚低耦合"的形式交互

容器： 容器是一种打包应用的方式，可以打包应用中的所有软件和软件所依赖的环境，并可实现跨平台部署。容器关键技术：namespace命名空间，cgroup控制组 ，UnionFS联合文件系统

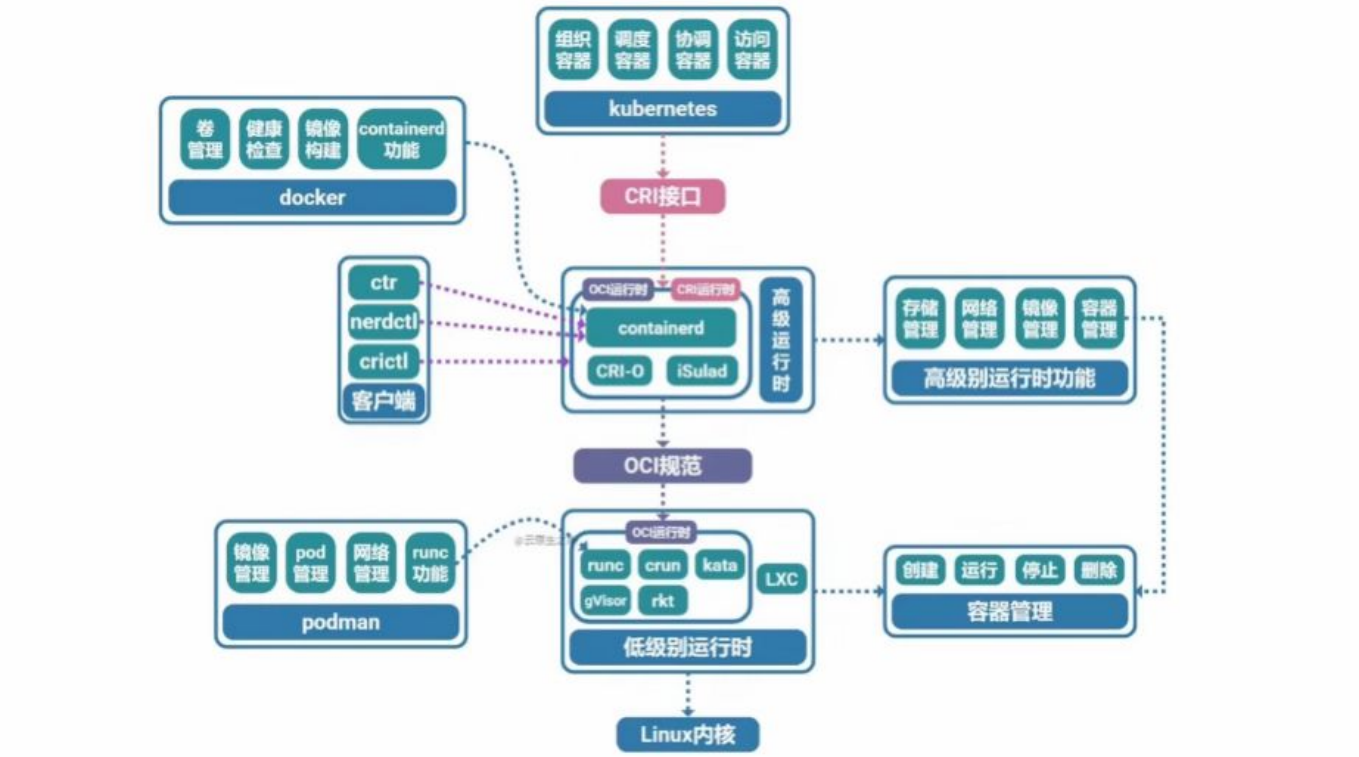
容器编排： 自动化管理和协调容器的系统，专注于容器的生命周期管理和调度

服务网格： Service Mesh 致力于解决服务间通讯的基础设施层

不可变基础设施： 任何基础设施实例（服务器、容器等各种软硬件）一旦创建之后便成为一种只读状态，不可对其进行任何更改。如果需要修改或升级实例，唯一方式是创建一批新实例以替换

声明式API： 可声明期望的状态，系统将不断地调整实际状态，直到与期望状态保持一致

DevOps： 缩短开发周期，增加部署频率，更可靠地发布



Docker核心概念

Docker官网

Image：

- 一个只读模板
- 由一堆只读层（read-only layer）重叠
- 统一文件系统（UnionFileSystem）整合成统一视角，相当于是一个root文件系统

Container：

- 从面向对象角度，利用容器并通过镜像创建的相互隔离的运行实例

- 最上面那一层可读可写层
- 一个可读写的统一文件系统，加上隔离的进程空间，以及包含在其中的应用进程

Repository：

- 全球最大的docker仓库 Docker Hub
- 集中存放镜像文件的地方
- Docker Registry 可包含多个仓库（Repository），每个仓库可包含多个标签（Tag），每个标签对应一个镜像

Docker使用

Docker Engine 安装 Docker Desktop安装

Pull & Run

```
docker pull nginx:latest
```

```
docker run -d --name my-nginx -p 8080:80 nginx:latest
```

Build & Push

1. 创建项目目录

```
mkdir nginx-docker && cd nginx-docker
```

2. 创建 index.html

```
mkdir html
echo "<h1>Welcome to SAST!</h1>" > html/index.html
```

3. 创建 nginx.conf

```
mkdir conf
vim conf/nginx.conf
```

```
worker_processes  1;
events {
    worker_connections 1024;
}
```

```
http {
    server {
        listen 80;
        server_name localhost;

        location / {
            root /usr/share/nginx/html;
            index index.html;
        }
    }
}
```

4. 创建 Dockerfile

```
# 使用官方 Nginx 基础镜像
FROM nginx:latest

# 复制自定义的 Nginx 配置文件
COPY conf/nginx.conf /etc/nginx/nginx.conf

# 复制 HTML 页面
COPY html/index.html /usr/share/nginx/html/index.html

# 暴露 80 端口
EXPOSE 80

# 运行 Nginx
CMD ["nginx", "-g", "daemon off;"]
```

5. 构建并运行自定义 Nginx 容器

```
docker build -t my-nginx .
docker run -d --name custom-nginx -p 8080:80 my-nginx
```

6. 打 Tag 并推送到 Docker Hub（可选）

```
docker tag my-nginx myusername/my-nginx:latest
docker push myusername/my-nginx:latest
```

一些别的命令 当然上面只是简单的使用示例，Docker的重点还包括，**docker数据卷，docker network**等

Docker-Compose

官网安装

只使用dockerfile进行创建镜像会有很多的问题，比如当项目部署的时候有redis，mysql这样的镜像启动顺序，或者说容器间的ip地址发生了变化，映射出错，要么生产ip写死，要么通过服务调用。因此使用docker-compose可以做到集中管理，一套带走

```
services:  # 定义应用的服务
  web:  # 服务名称
    image: nginx:latest  # 使用的镜像
    ports:  # 定义端口映射
      - "8080:80"  # 将容器的 80 端口映射到主机的 8080 端口
    volumes:  # 定义卷挂载
      - ./web-content:/usr/share/nginx/html  # 将主机上的 ./web-content 目录挂载到容器的 /usr/share/nginx/html 目录
    networks:  # 定义网络
      - frontend
    environment:  # 设置环境变量
      - NGINX_VERSION=latest

  db:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=root_password
      - MYSQL_DATABASE=my_database
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
    volumes:
      - db-data:/var/lib/mysql
    networks:
      - frontend
      - backend

networks:  # 定义网络
  frontend:
  backend:

volumes:  # 定义卷
  db-data:
  web-content:
```

- `services`：定义应用的各个服务，每个服务包含一个或多个容器。
- `image`：指定服务所使用的镜像。
- `ports`：定义端口映射，将容器内的端口映射到主机上的端口。
- `volumes`：定义卷挂载，将主机上的目录或卷挂载到容器内。
- `networks`：定义服务使用的网络，以及服务之间的连接关系。
- `environment`：设置容器的环境变量。
- `volumes`：定义卷，用于持久化数据。

编写好之后，使用docker-compose up -d 可以一次性的把需要的容器给创建好，就省去了很多麻烦

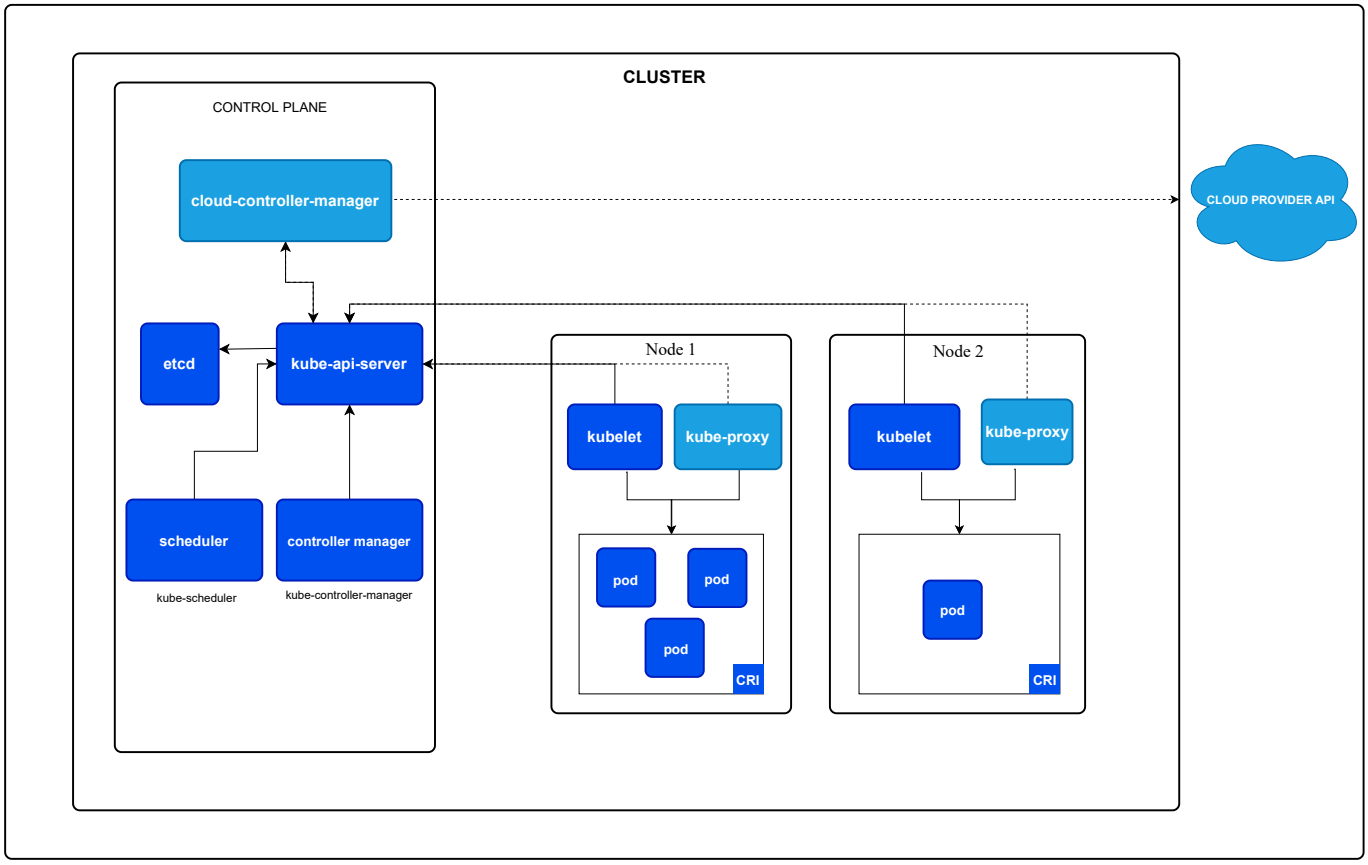
K8S

Kubernetes 是 Google 基于十多年的生产环境运维经验，开发出的一个生产级别的容器编排系统。在 Kunernetes 文档中，这样描述 Kubernetes：

> [Success] "an open-source system for automating deployment, scaling, and management of containerized appcations".
> “一个自动化部署、可拓展和管理容器应用的开源系统”

架构

很推荐跟着官方中文文档进行学习



k8s中有非常多的资源对象，深入了解学习它们是为了解决什么问题，是怎么协同工作的

学习环境

可以使用 minikube、kind 做单机学习环境，或者购买云服务器、白嫖 3个月 Google Cloud ，也可以使用电脑创建多个虚拟机或者多台电脑来搭建节点，或者使用线上学习环境等。