# 概述

- 参考教材
  - [Cloud Native Pattern][1] 第一章
- 当前已经进入了云时代
  - 数字化、网络化深入人心
  - 越来越多的服务运行在云上
  - 网络就是云，云就是网络
  - 网络与水、电、气一样重要
- 但是云有时也出错
  - 2015年9月20日，Amazon Web Service（AWS）发生了宕机事件，停运时间超过5小时
    - Netflix, Airbnb, Nest等一大批公司受到影响
    - 由于采取了冗余机制，在宕机发生后，Netflix 在几分钟内便恢复了服务
      - 采取了备份机制，当东部地区的服务宕机后，能够快速将请求转发到其他地区
      - 视失败或改变为习惯：treat change or failure as the rule.
- **TODO 应该加入十二要素应用**

# 云时代企业应用的需求（云应用）

- 零故障时间：7 * 24 小时服务
  - 对于关键应用，用户越来越不能忍受服务掉线
  - 即便是由于系统升级造成的短暂不可用也不能忍受
  - 对于运维工程师：保证基础设施的可用性
  - 对于开发工程师：设计松耦合的系统
    - 设计、开发的松耦合
    - 部署、运行时的松耦合
- 快速的反馈周期：必须具备快速的发布能力
  - 第一时间满足用户的需求
  - **单块（monolithic）应用**做不到快速反馈
- 移动和物联网的多设备支持
  - 多终端设备：手机、平板、桌面、电视等
  - 物联网设备：智能家居，智慧交通等
  - 无时不在获取数据，数据请求规模激增
  - 数据量越来越大，需要强大的计算能力
- 数据驱动
  - 不需要巨大的、集中式的数据库
  - 需要小的，分布式的，本地化的数据库
  - 智能化：通过数据挖掘价值

# 单块应用

- 所有的功能被打包成 WAR 或 JAR 包，并且运行在一个进程中
  - 开发时，一个大工程包含了所有的模块代码，共享同一个版本
  - 部署时，打包为单独的 WAR 或 JAR
  - 运行时，运行在一个 JVM 进程中（包括服务器进程或单独进程）

- 单块应用的好处
  - 开发简单：有许多先进的 IDE 全面支持单个应用的开发
  - 适应性强：可以很容易修改代码，数据库设计，然后构建和部署
  - 容易测试：很容易写出端到端的测试代码
  - 容易部署：打包然后放到 Tomcat 的部署目录
  - 容易扩展：可以方便地运行多个实例
- 但是，随着时间的推移，这些好处会逐渐消失
  - 随着功能的不断加入，应用的功能逐渐增多，应用变得越来越复杂，应用的尺寸越来越大
  - 开发速度越来越慢，
    - IDE 的速度，
    - 代码的构建速度
    - 应用的启动速度（测试时需要运行应用）
  - 部署和交付过程很痛苦
    - 部署过程：提交代码到 VCS （版本控制服务器） -> 构建（编译、测试）-> 部署 -> 运行
    - 目标：追求快速反馈，一日部署多次（持续交付）
    - 但由于代码过大，部署过程很慢，无法实现快速反馈
    - 在项目工期紧张的情况下，往往仓促交付
  - 扩展困难
    - 应用过大，并且还要运行在一个进程，需要的硬件资源很难满足
  - 技术栈单一
    - 无法使用新技术
- **问题：当前所能够开发的应用无法满足这些需求**
  - 虽然进行了前后端分离
  - 后端部分还属于单块应用
- **解决办法：微服务（Microservice）架构**
  - 三个维度解释微服务
    - 分解功能：把单块应用的功能分解为一个一个的服务（暂时可以理解为单体应用中的模块）
    - 分解数据：把集中式的数据库按服务分割为多个数据库，每个数据库对于其所属服务是私有的
    - 水平扩展：当需要扩展时，可以动态部署多个服务实例
      - 如：双11，12306，疫情期间的网络教学
  - 好处
    - 允许持续交付（Continuous Delivery）大型、复杂应用
    - 服务很小且容易维护
    - 服务可独立开发
    - 服务可独立扩展
    - 开发服务的团队能够自治
    - 允许使用新技术
    - 能够进行错误隔离
  - 坏处
    - 设计合适的服务集有一定难度 -> 领域驱动设计
    - 分布式系统很复杂，给开发，测试，部署带来挑战 -> 云原生
      - 如何通讯：如何发现服务
      - 如何保证可靠性、一致性、安全性?

- 横切关注点：如何记录日志与审计，如何负载均衡，如何验证与授权，如何异常处理

[1]:Cloud Native Pattern. 2019.5