

企业级应用软件开发与开发

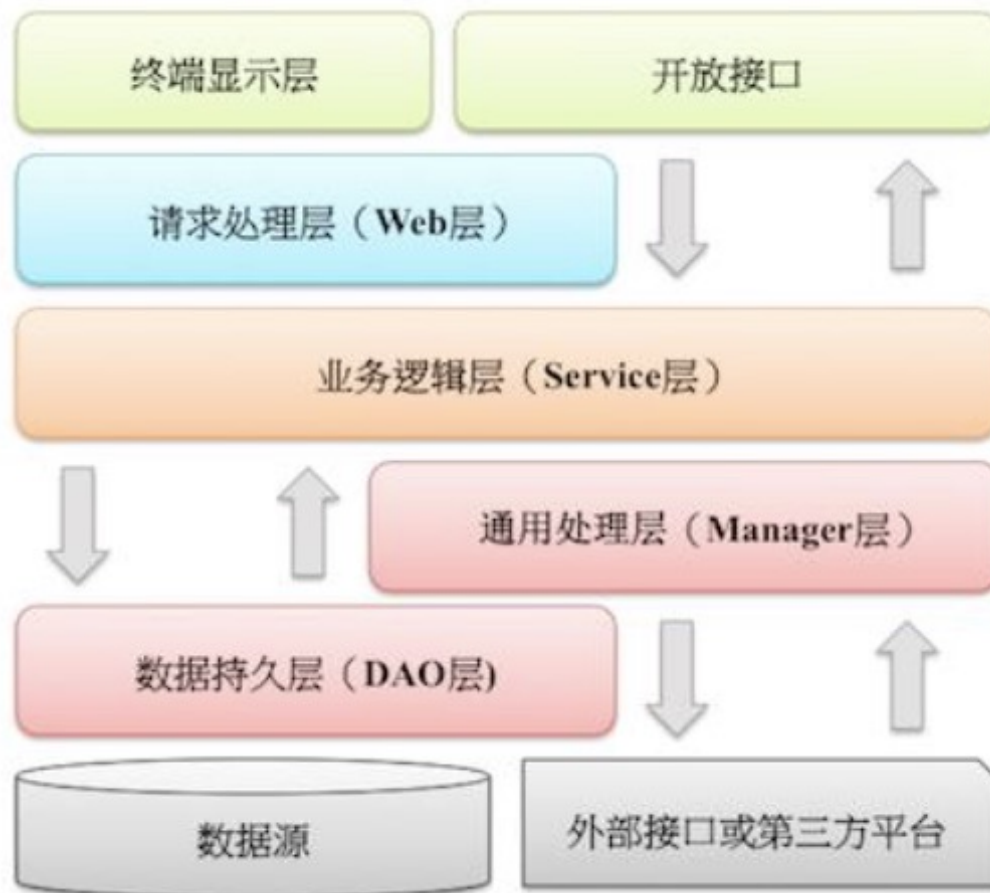
软件开发技术的趋势

内容提要

- 回顾Spring
- Cloud Native
- AI Native

Spring Framework是什么

- Spring是一个开源的控制反转(Inversion of Control ,IoC)和面向切面(AOP)的容器框架.它的主要目的是**简化企业开发**.



IOC (Inversion of Control) 控制反转

```
public class PersonServiceBean {  
    private PersonDao personDao = new PersonDaoBean();  
  
    public void save(Person person){  
        personDao.save(person);  
    }  
}
```

PersonDaoBean 是在应用内部创建及维护的。所谓控制反转就是应用本身不负责依赖对象的创建及维护，依赖对象的创建及维护是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是所谓反转。

依赖注入(Dependency Injection)

当我们把依赖对象交给外部容器负责创建，那么PersonServiceBean类可以改成如下：

```
public class PersonServiceBean {  
    private PersonDao personDao ;  
    //通过构造器参数，让容器把创建好的依赖对象注入进PersonServiceBean，当然也可以使用  
    //setter方法进行注入。  
    public PersonServiceBean(PersonDao personDao){  
        this.personDao=personDao;  
    }  
    public void save(Person person){  
        personDao.save(person);  
    }  
}
```

所谓依赖注入就是指：在运行期，由外部容器动态地将依赖对象注入到组件中。

为何要使用Spring Framework

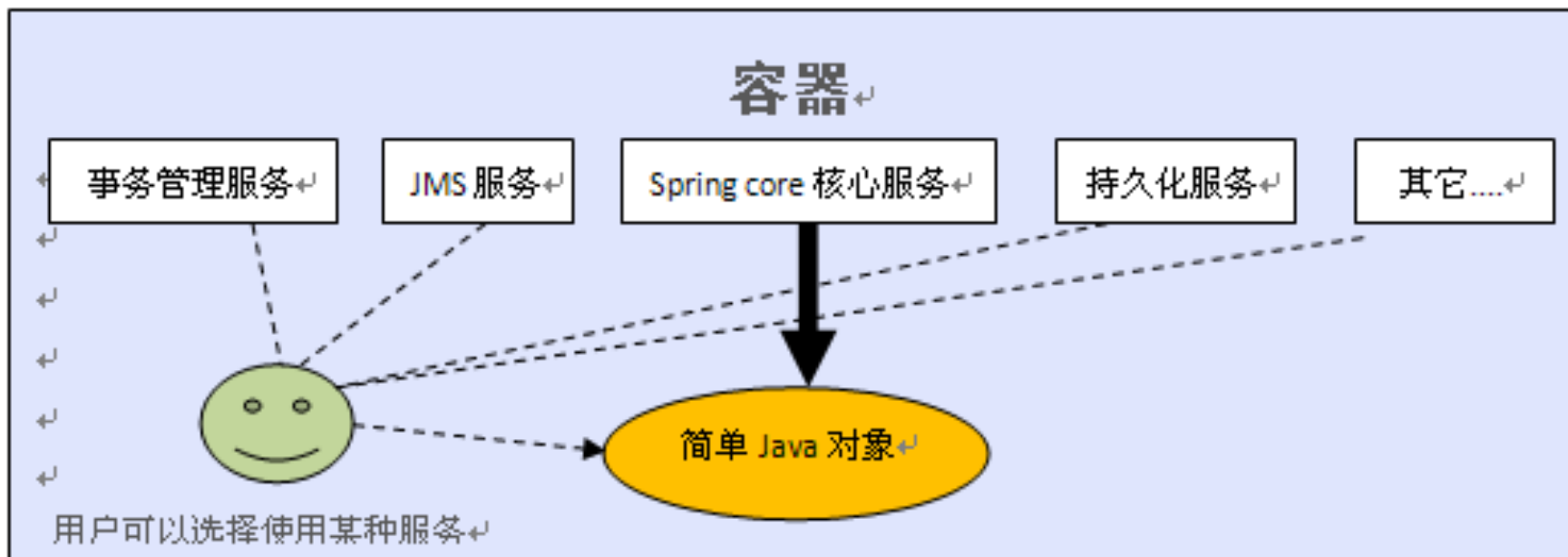
- 降低组件之间的耦合度,实现软件各层之间的解耦。



- 可以使用容器提供的众多服务,如:事务管理服务、消息服务等等。当我们使用容器管理事务时,开发人员就不再需要手工控制事务.也不需处理复杂的事务传播。
- 容器提供单例模式支持,开发人员不再需要自己编写实现代码。
- 容器提供了AOP技术,利用它很容易实现如权限拦截、运行期监控等功能。
- 容器提供的众多辅助类,使用这些类能够加快应用的开发,如:JdbcTemplate、HibernateTemplate。
- Spring对于主流的应用框架提供了集成支持,如:集成**Spring MVC**、**Mybatis**、Hibernate、JPA、Struts等,这样更便于应用的开发。

使用Spring Framework的好处

- 当使用spring Framework时，我们可以选择使用容器提供的众多服务



Spring, 始于框架, 但不不限于框架

Spring: the source for modern java

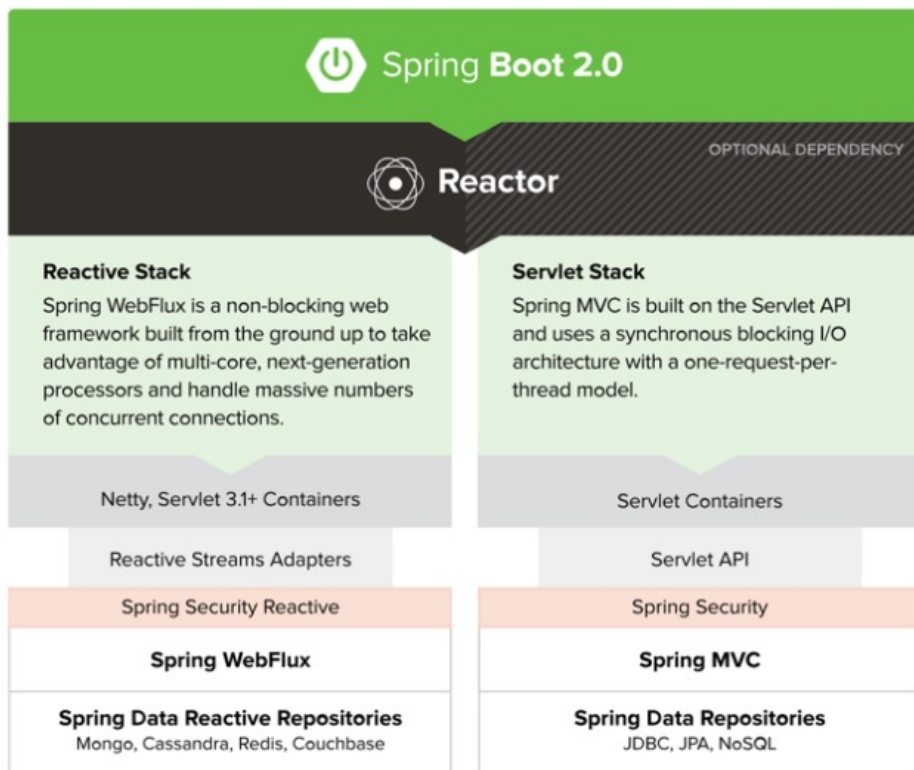


Spring, 始于框架, 但不限于框架

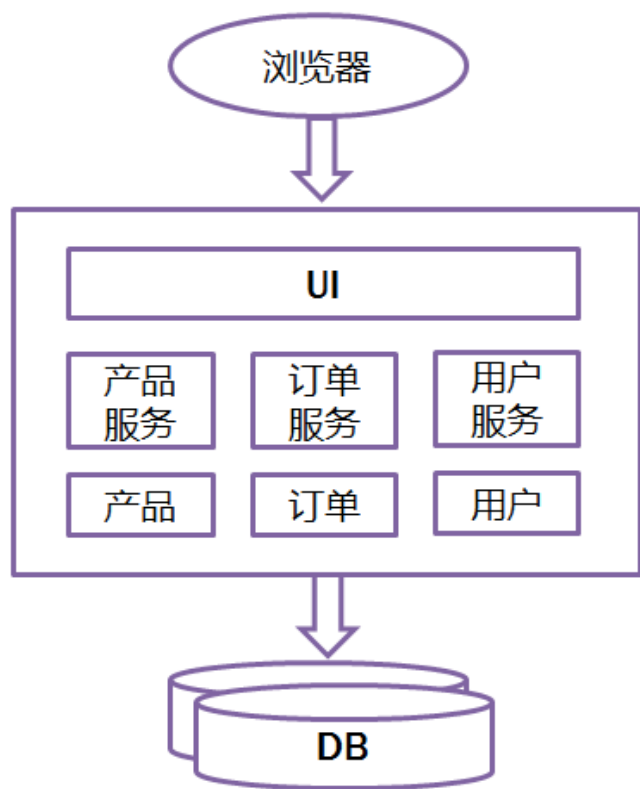
(1) Spring Framework (2) Spring相关项目 (3) 整个Spring家族

Spring Boot

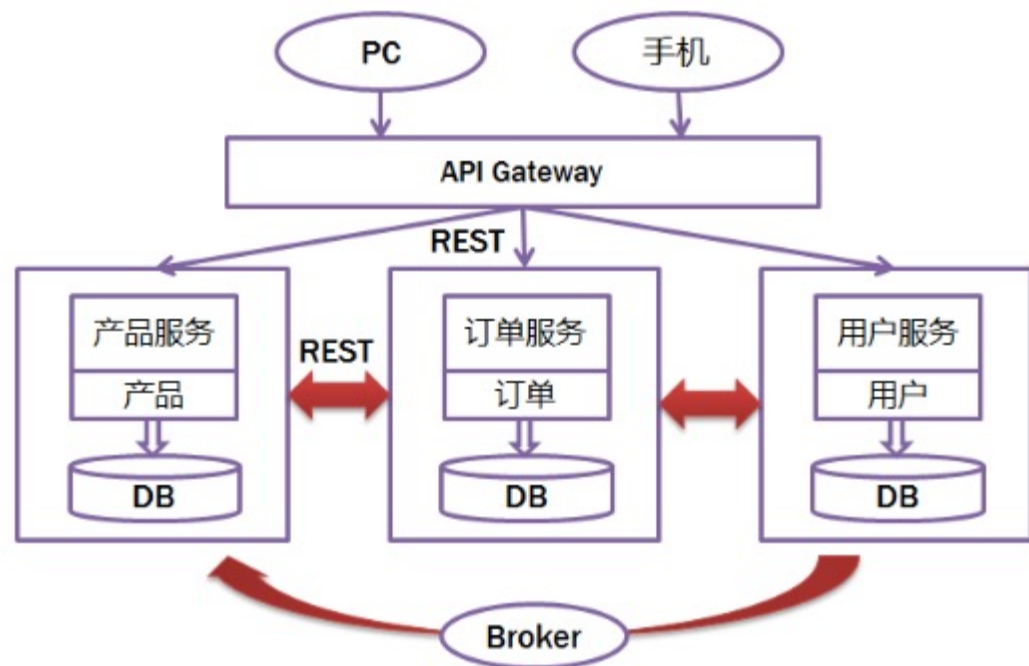
- 快速构建基于Spring的应用程序
 - 快、很快、非常快
 - 进可开箱即用，退可按需改动
 - 提供各种非功能特性
 - 不用生成代码，没有 XML 配置
- 在本课程中，你还会看到
 - Spring Data、Spring MVC、Spring WebFlux.....



Web应用进化 – 微服务



单体架构

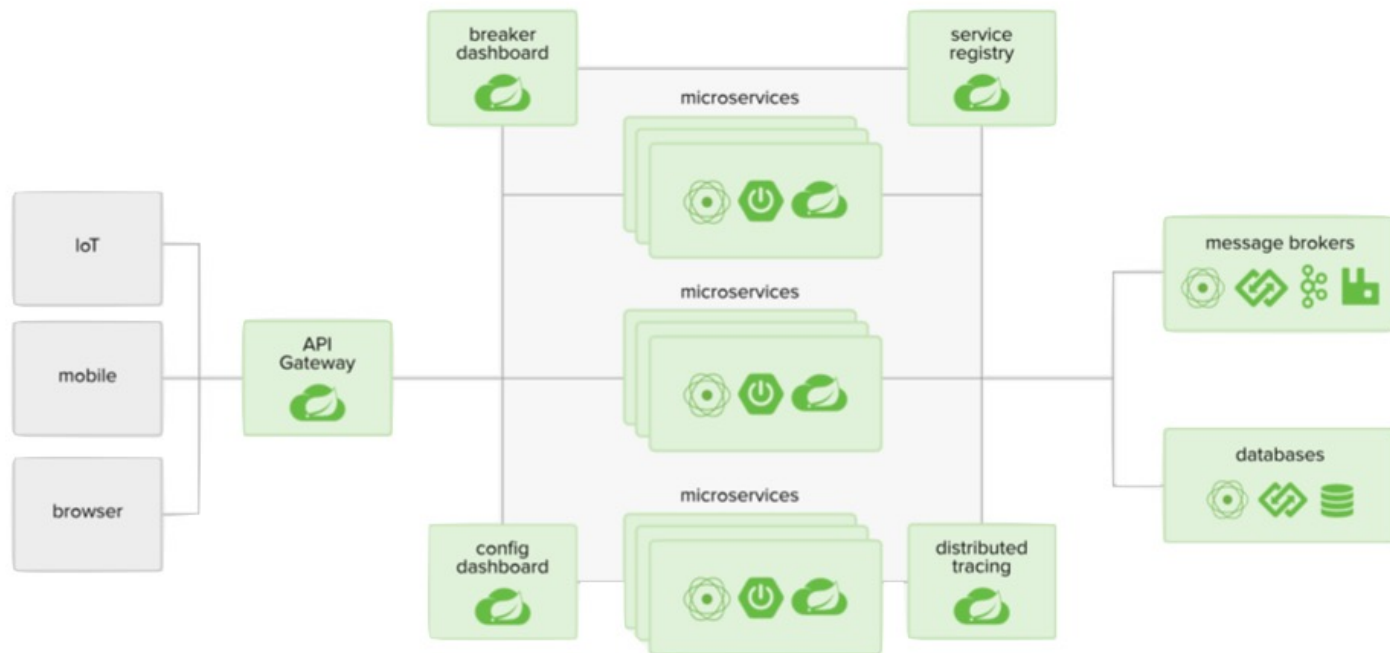


微服务架构

Spring Cloud

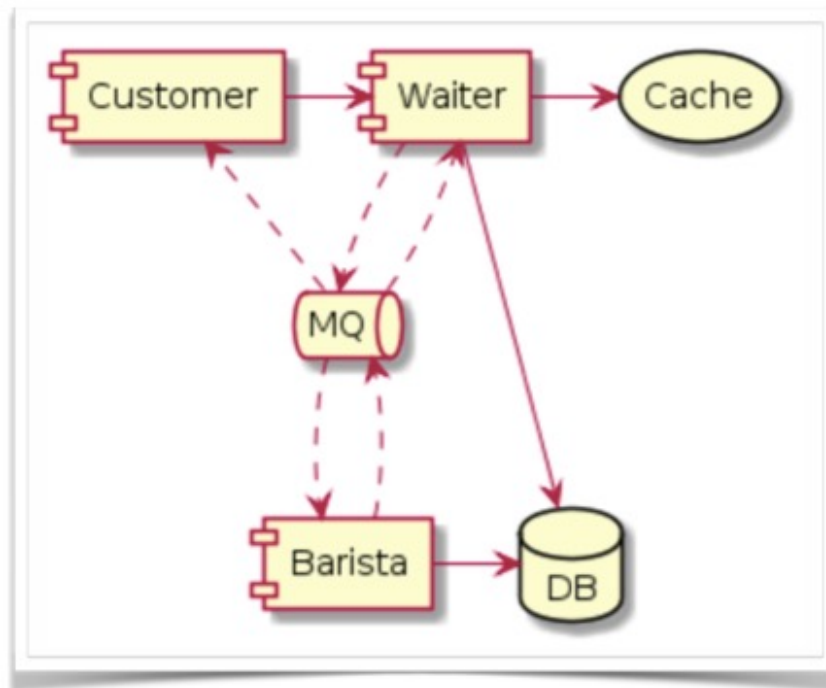
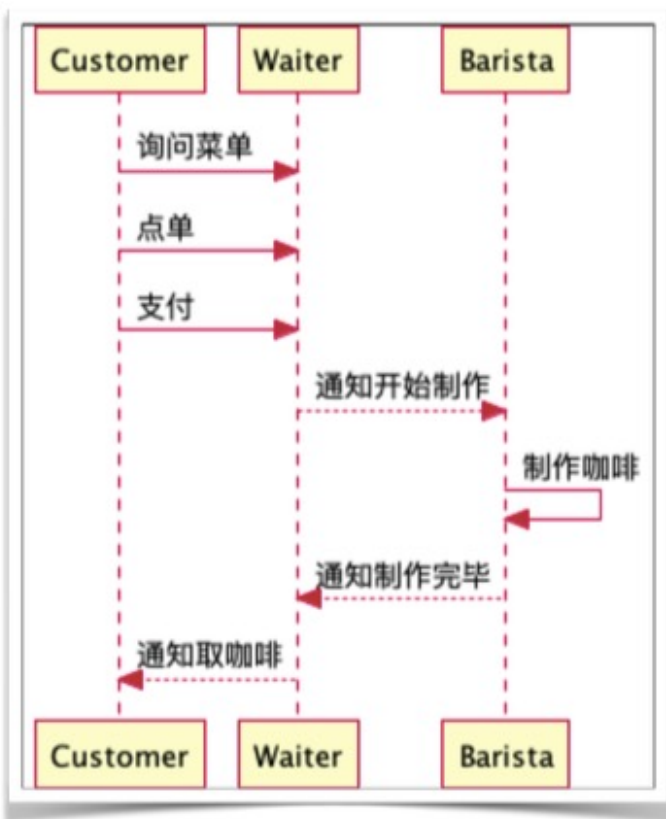
- 简化分布式系统的开发

- 配置管理
- 服务注册与发现
- 熔断
- 服务追踪
-



项目SpringBucks目标

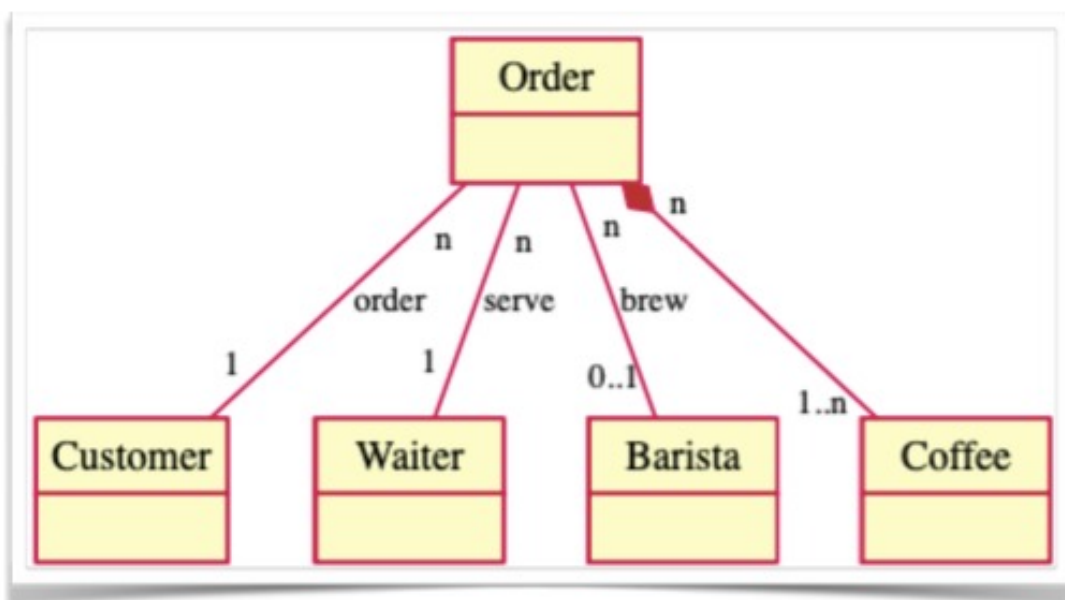
- 通过一个完整的例子演示 Spring 家族各主要成员的用法
- 代码托管 <https://gitee.com/qxr777/javaee-course-code>



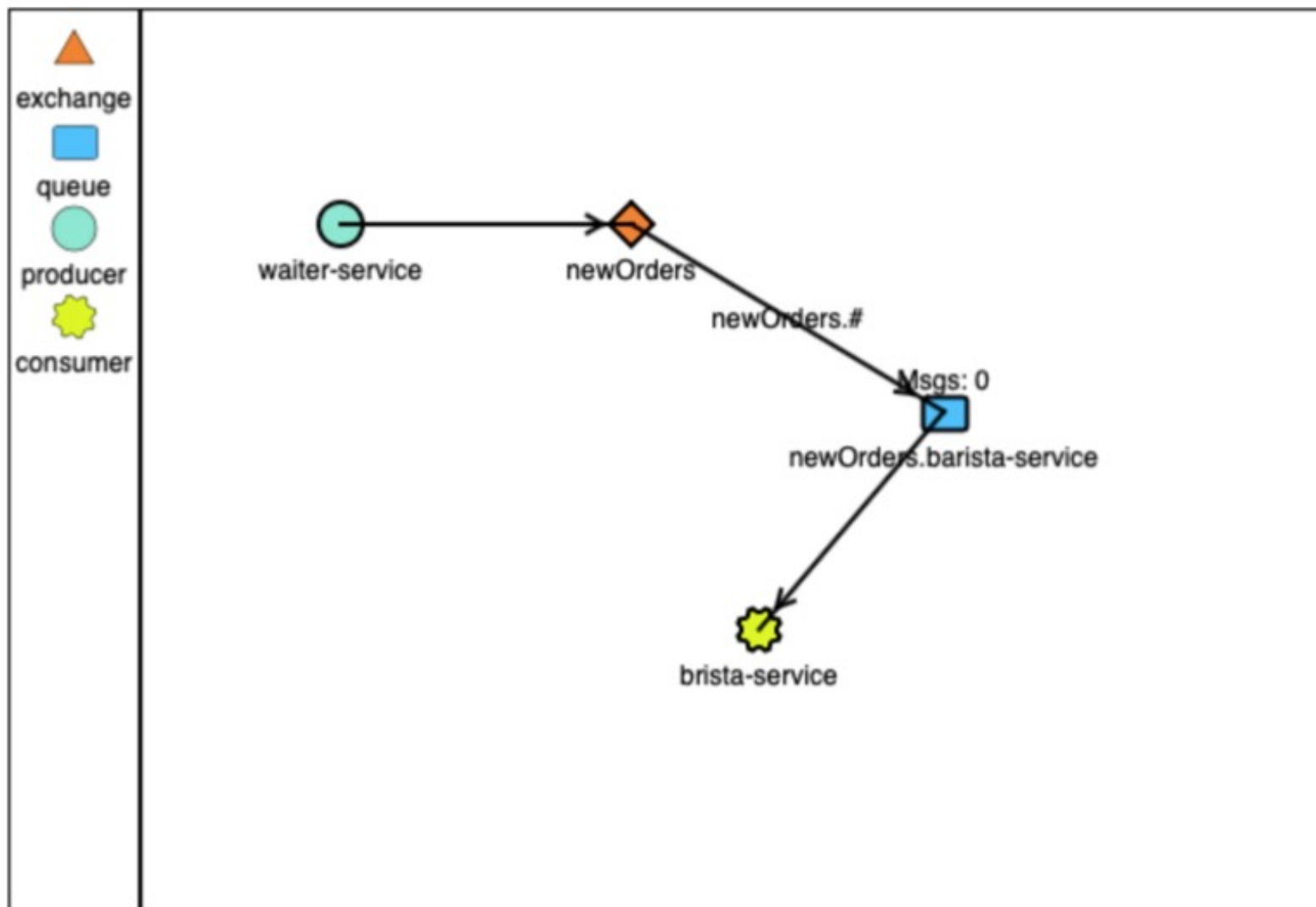
项目中的对象实体

实体

- 咖啡、订单、顾客、服务员、咖啡师



消息在 RabbitMQ 的流转



内容提要

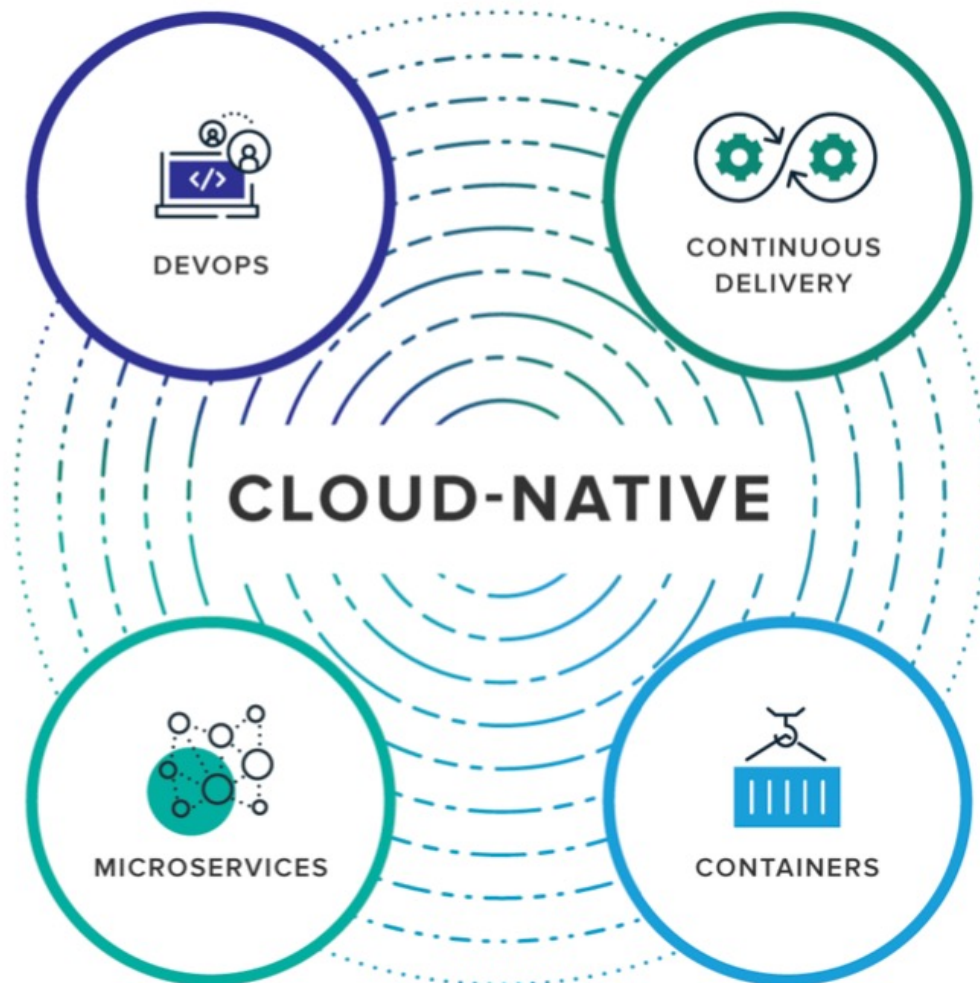
- 回顾Spring
- **Cloud Native**
- **AI Native**

Cloud Native定义

“云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。”

– *CNCF Cloud Native Definition v1.0*

云原生应用要求.....



云原生应用要求.....

DevOps

- 开发与运维一同致力于交付高品质的软件服务于客户

持续交付

- 软件的构建、测试和发布，要更快、更频繁、更稳定

微服务

- 以一组小型服务的形式来部署应用

容器

- 提供比传统虚拟机更高的效率

微服务定义



微服务

进入词条

播报

★ 收藏 | 74 | 41

微服务（SOA架构的一种变体）

播报

编辑

讨论 4

上传视频

什么是微服务？

维基上对其定义为：一种软件开发技术-面向服务的体系结构（SOA）架构样式的一种变体，它提倡将单一应用程序划分成一组小的服务，服务之间互相协调、互相配合，为用户提供最终价值。每个服务运行在其独立的进程中，服务与服务间采用轻量级的通信机制互相沟通（通常是基于HTTP的RESTful API）。每个服务都围绕着具体业务进行构建，并且能够独立地部署到生产环境、类生产环境等。另外，应尽量避免统一的、集中式的服务管理机制，对具体的一个服务而言，应根据上下文，选择合适的语言、工具对其进行构建。

| | | | |
|-----|--------------|------|-------------------|
| 中文名 | 微服务 | 所属学科 | 软件架构 |
| 外文名 | microservice | 目 的 | 有效的拆分应用，实现敏捷开发和部署 |

| | | |
|----|---|---|
| 目录 | <div>1 简介</div> <div>2 受益方法<ul style="list-style-type: none">可独立部署正确的工作工具精确缩放</div> | <div>3 关键支持技术和工具<ul style="list-style-type: none">容器，Docker和KubernetesAPI网关</div> <div>4 常见模式</div> <div>5 反模式</div> |
|----|---|---|

微服务的优点

异构性

- 语言、存储.....

弹性

- 一个组件不可用，不会导致级联故障

扩展

- 单体服务不易扩展，多个较小的服务可以按需扩展

微服务的优点

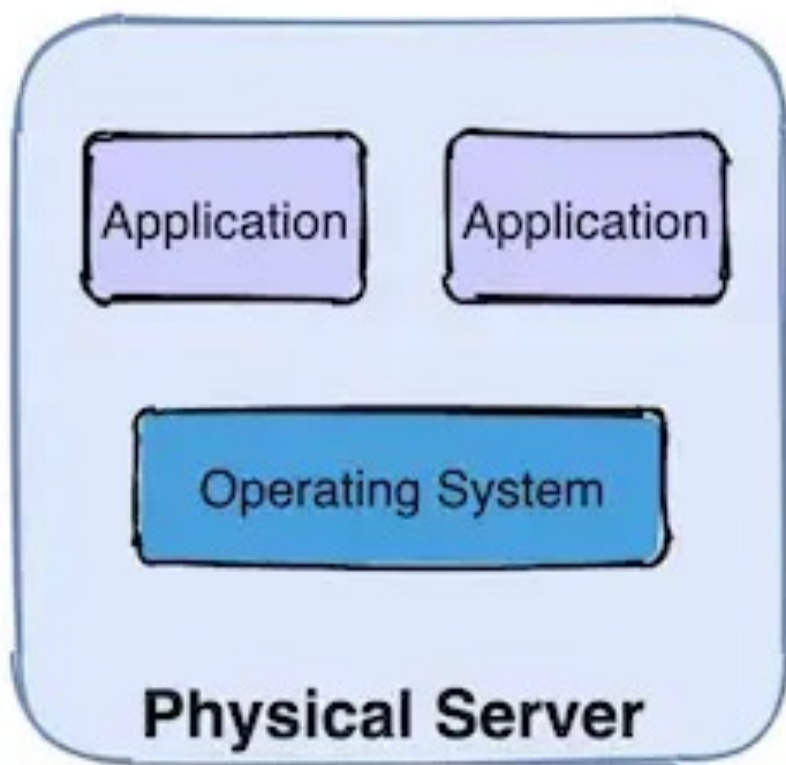
- 易于部署
- 与组织结构对齐
- 可组合性
- 可替代性

实施微服务的代价

- 分布式系统的复杂性
- 开发、测试等诸多研发过程中的复杂性
- 部署、监控等诸多运维复杂性
-

容器化时代来了

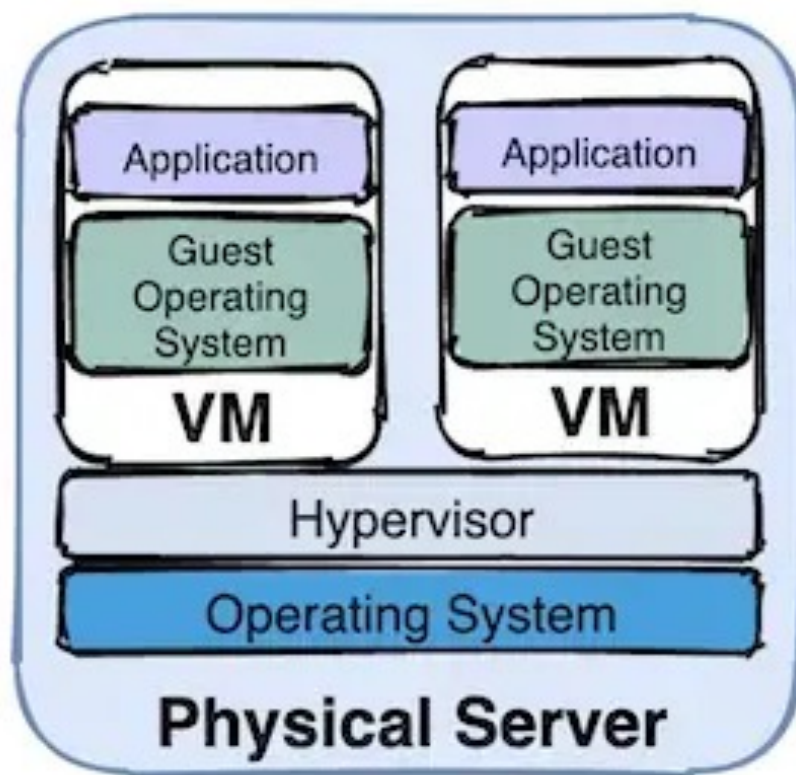
- 物理机时代：多个应用程序可能会跑在一台机器上。



@掘金技术社区

容器化时代来了

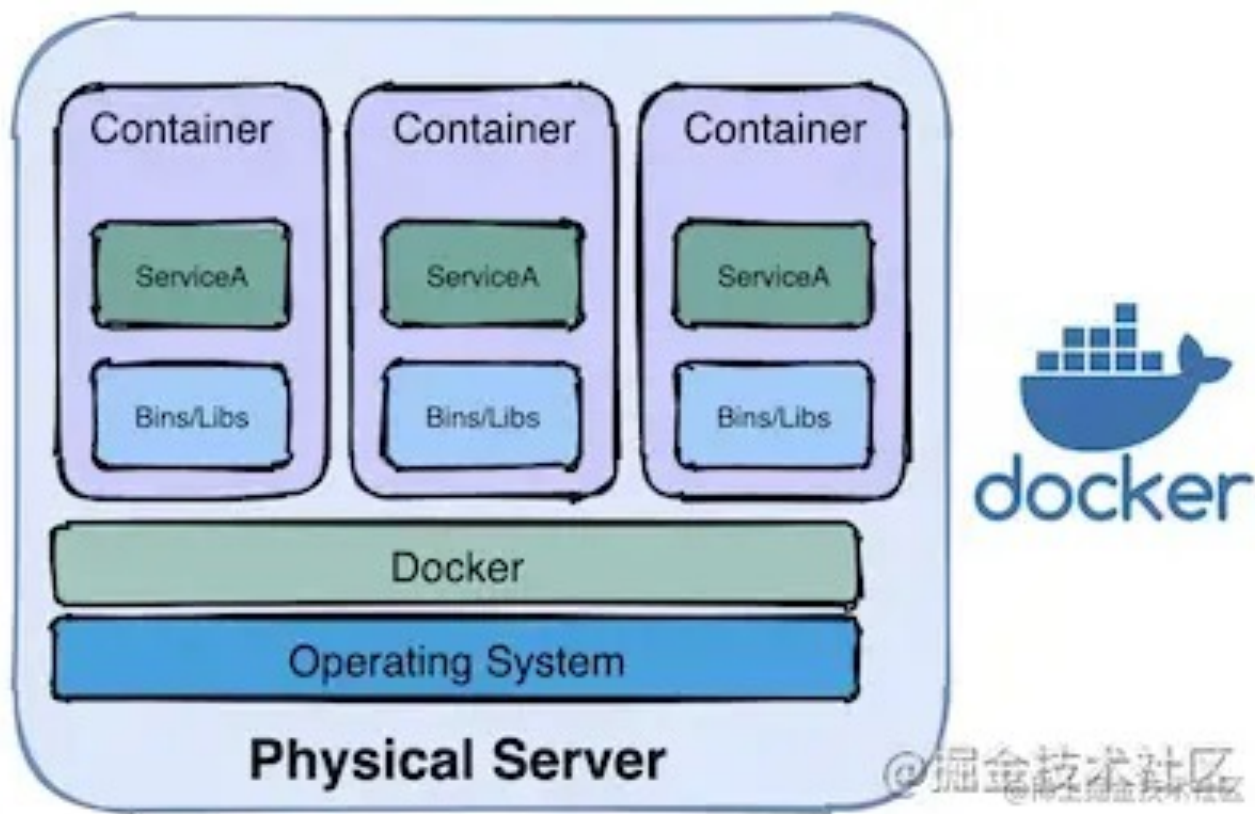
- 物理机时代：多个应用程序可能会跑在一台机器上。
- 虚拟机时代：一台物理机器安装多个虚拟机（VM），一个虚拟机跑多个程序。



@掘金技术社区

容器化时代来了

- 物理机时代
- 虚拟机时代
- 容器化时代：一台物理机安装多个容器实例（container），一个容器跑多个程序。



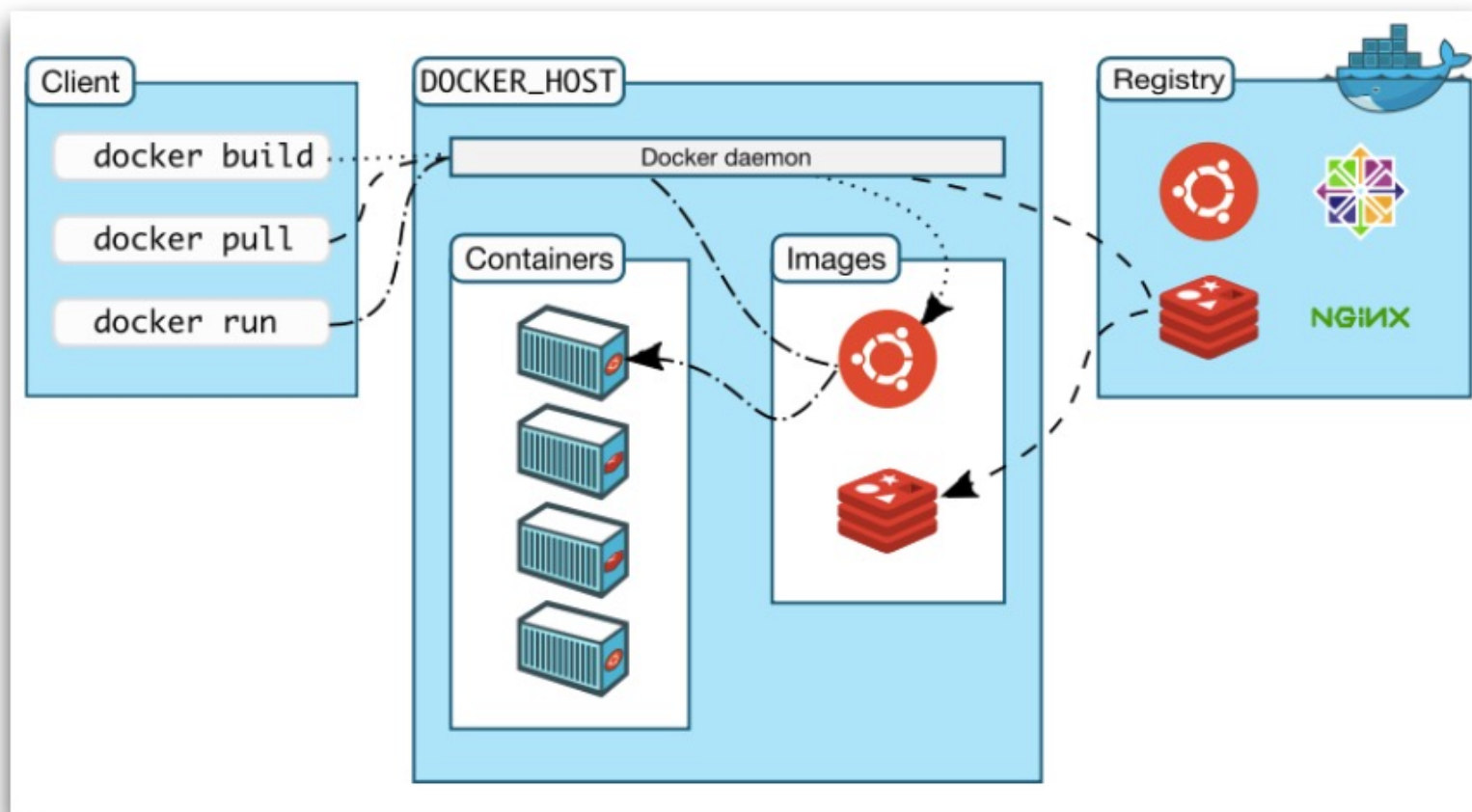
Docker 和容器技术

- 一个令人非常头疼的问题
 - 测试人员：你这个功能有问题。
 - 开发人员：我本地是好的呀！



Docker 和容器技术

- 第一句: Build, Ship and Run
- 第二句: Build once, Run anywhere (搭建一次, 到处能用)



一键体验 Docker Desktop

Docker Desktop

Install Docker Desktop – the fastest way to containerize applications.

Download Docker Desktop

Apple Intel Chip



本地部署Google Gemini Pro Chat容器

执行指令

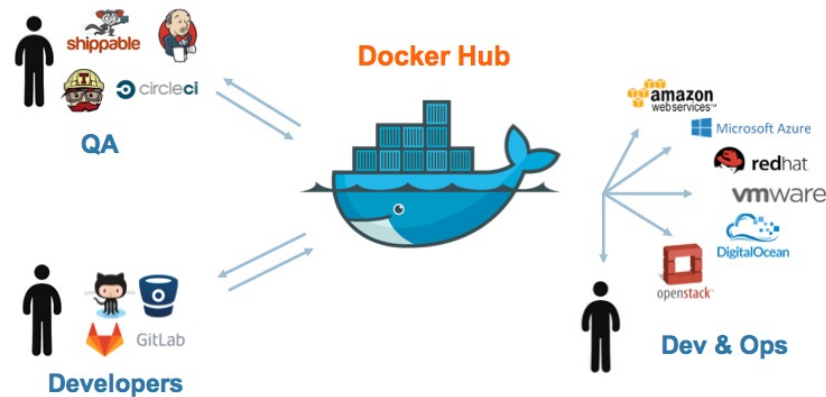
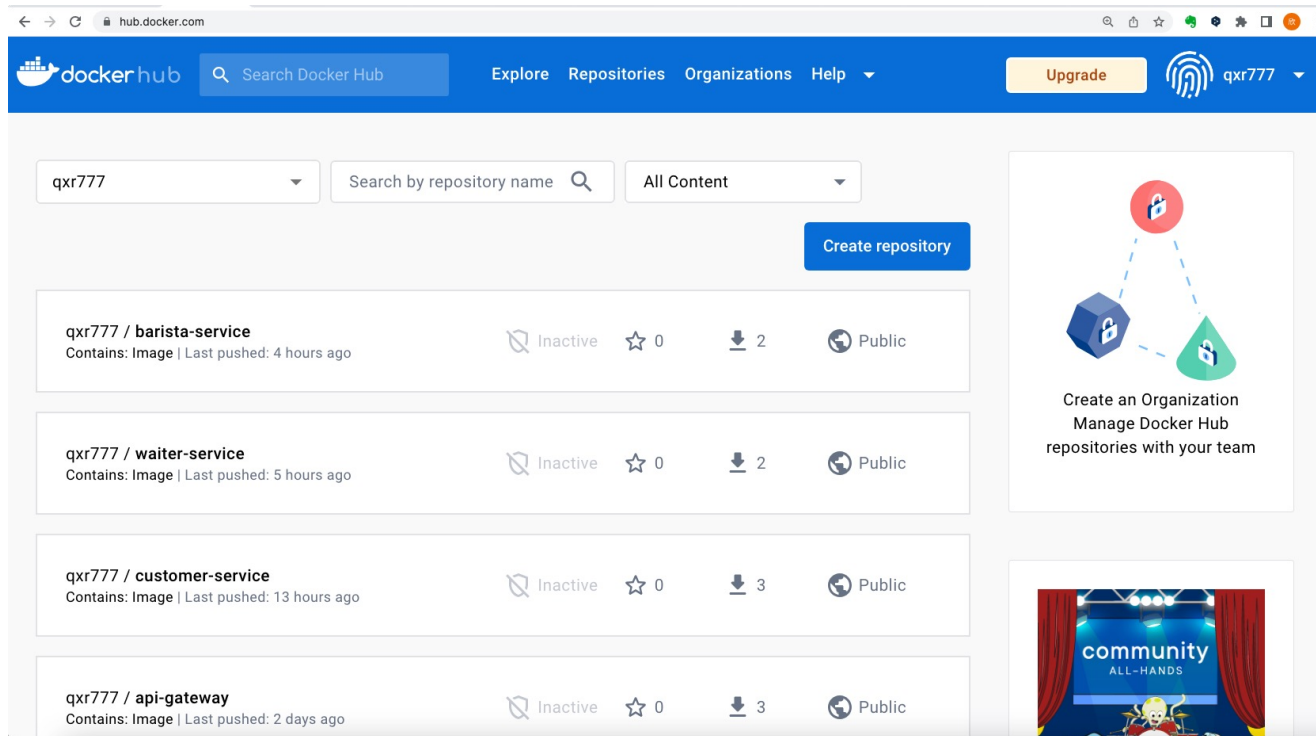
```
docker run --name  
geminiprochat \  
--restart always \  
-p 3000:3000 \  
-itd \  
-e GEMINI_API_KEY=<Your  
Key> \  
babaohuang/geminiprochat:l  
atest
```



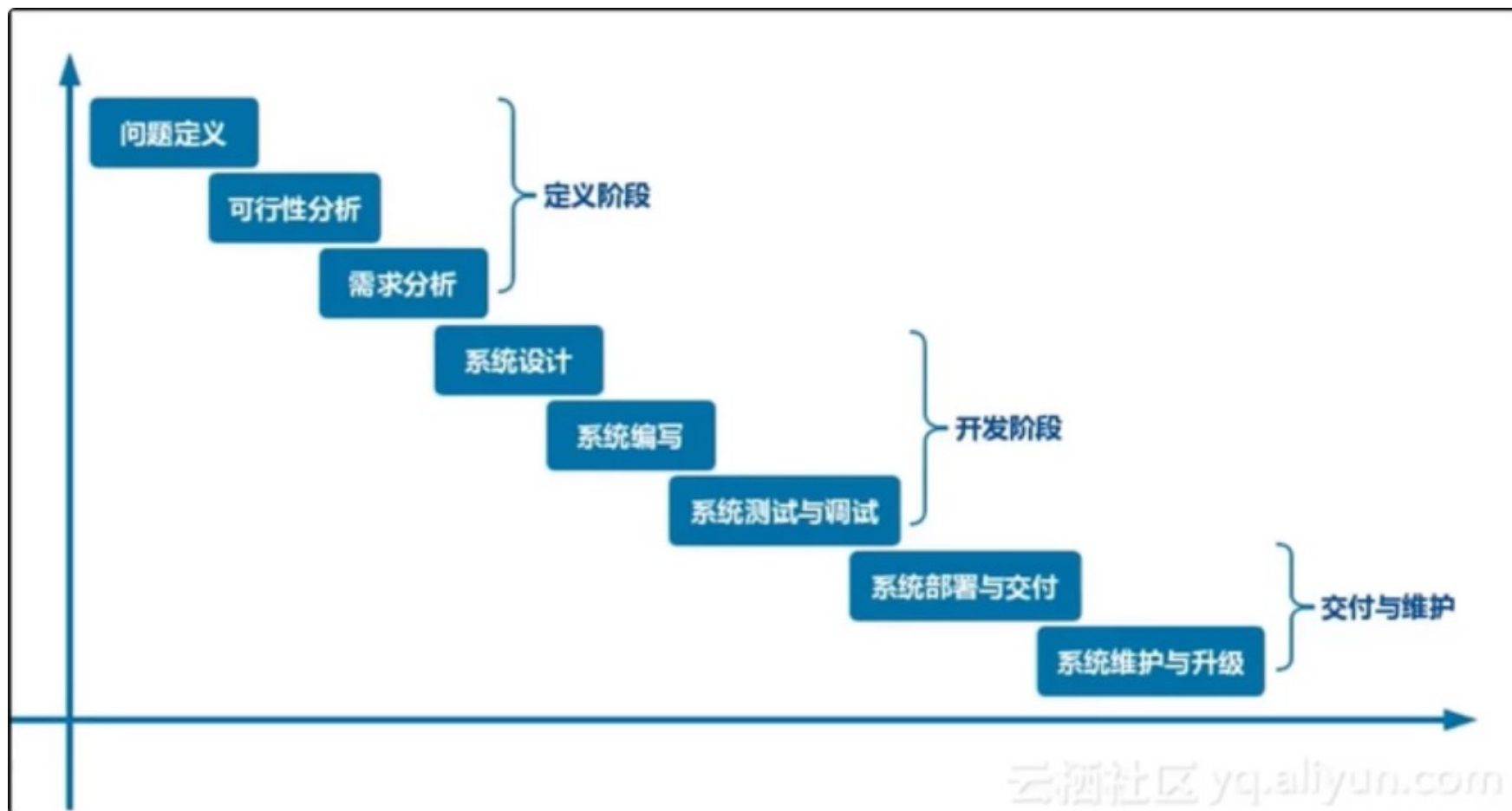
通过 Maven 构建 Docker 镜像

- 准备工作
 - 提供一个 **Dockerfile**
 - 配置 **dockerfile-maven-plugin** 插件
- 执行构建与推送
 - **mvn package**
 - **mvn dockerfile:push**
- 检查结果
 - **docker images**

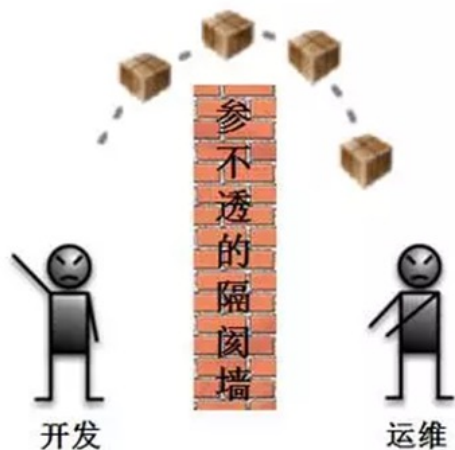
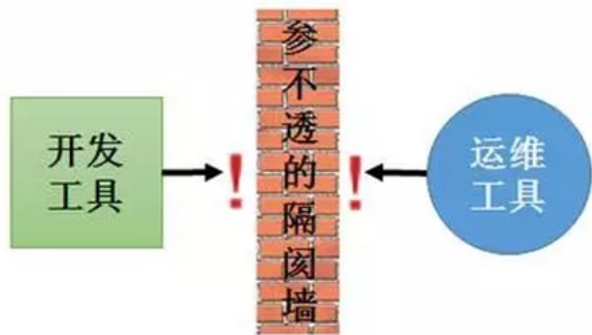
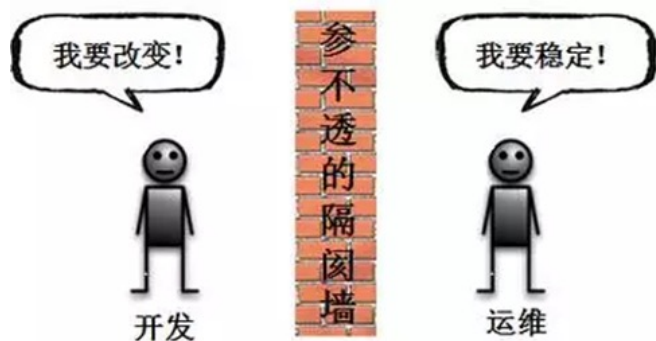
发布Docker镜像至Docker Registry



软件开发方法论 – 瀑布式开发

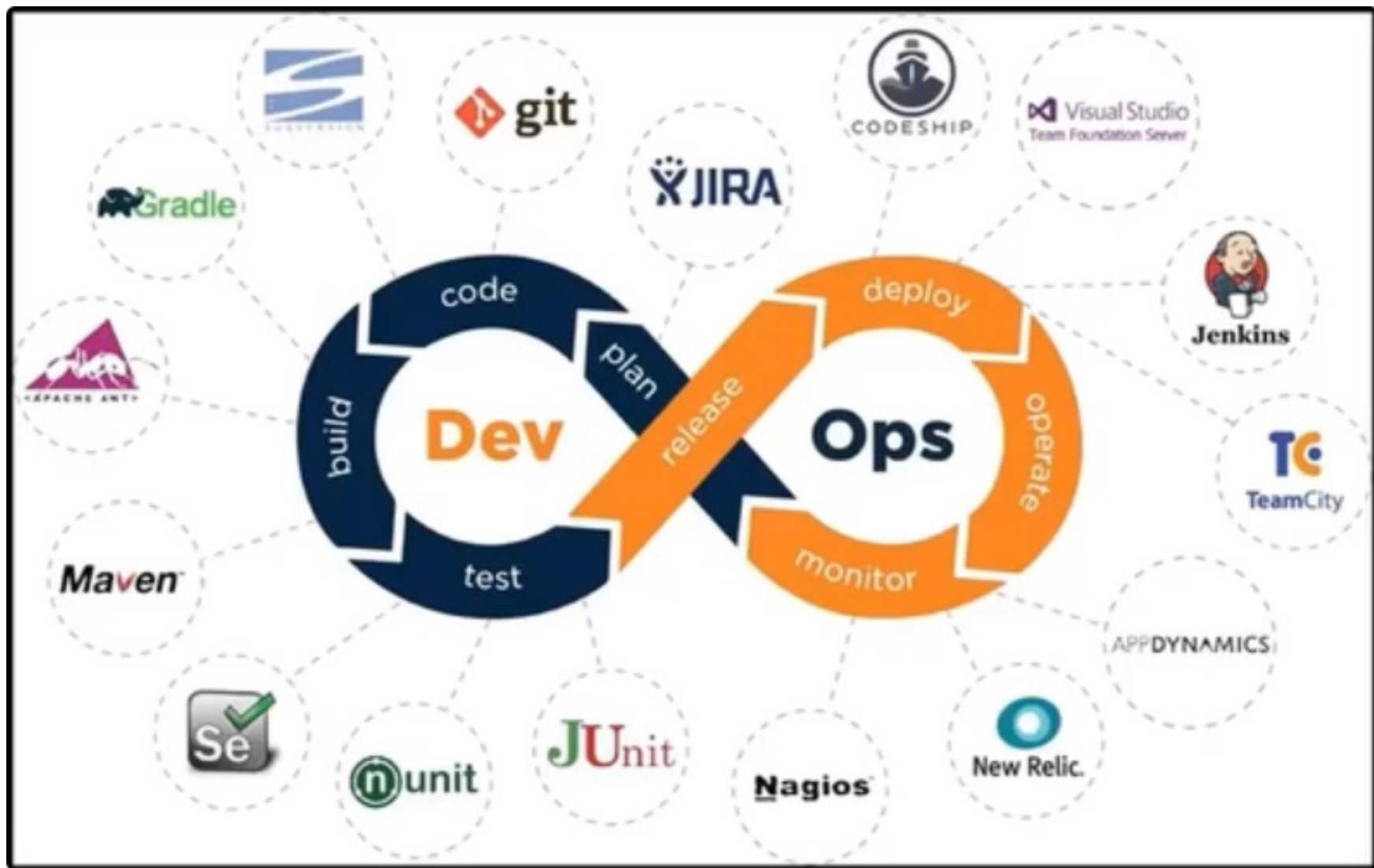


软件开发方法论 – DevOps



DevOps =
工具 + 实践 + 文化

软件开发方法论 – DevOps



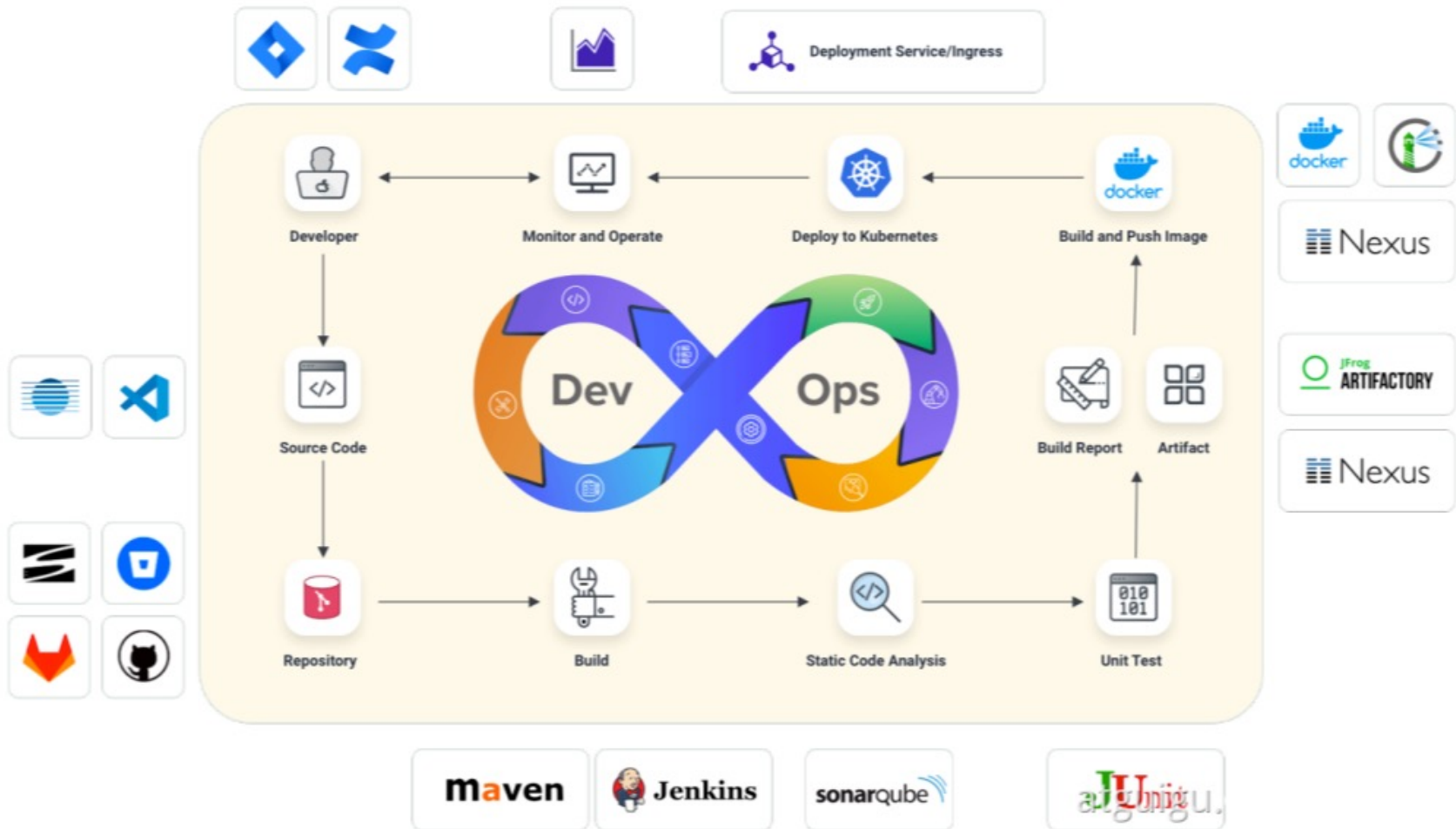
CICD - 持续集成与持续交付



CICD - 持续集成与持续交付



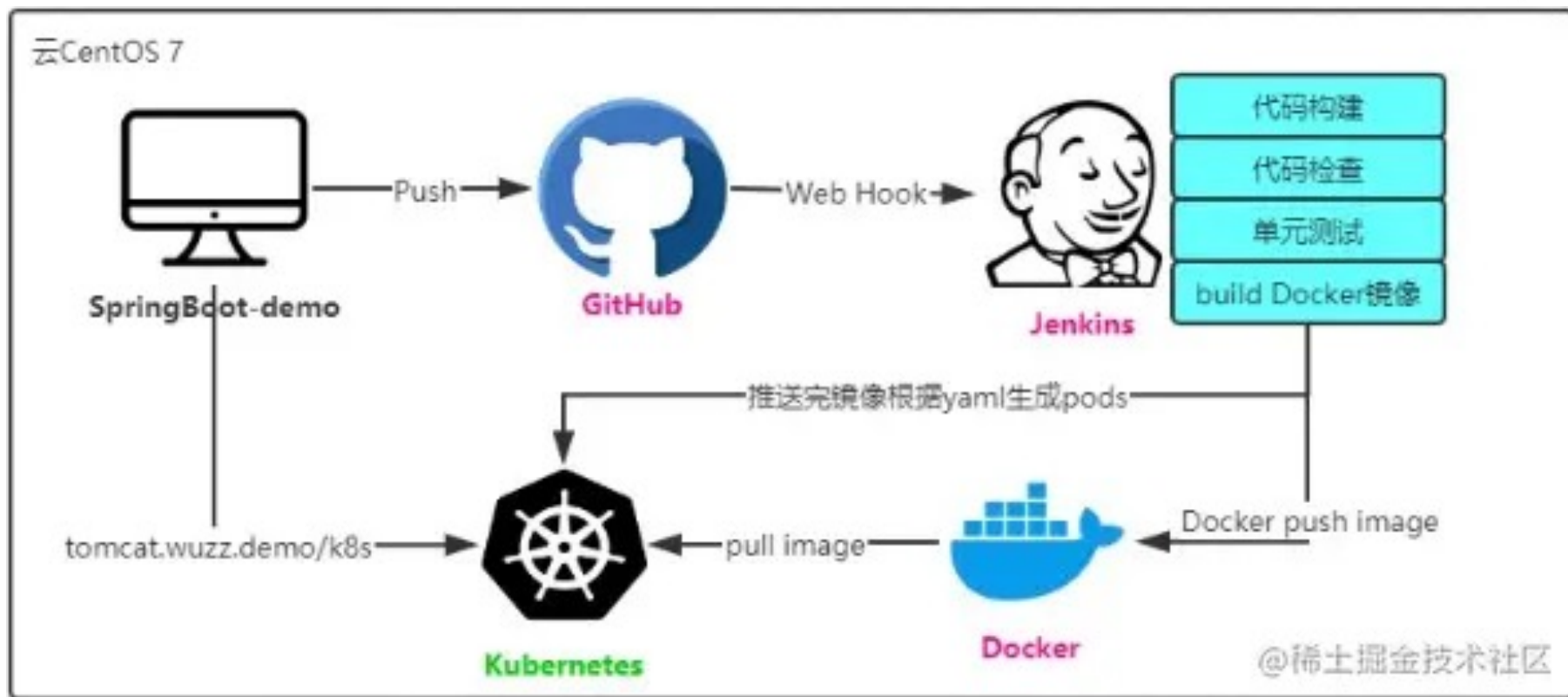
- 最著名的开源工具之一就是自动化服务器 **Jenkins**。



Kubernetes(K8S) 集成 Jenkins 实现 CI/CD



kubernetes



内容提要

- 回顾Spring
- Cloud Native
- AI Native

大模型如何重新定义人工智能

- 人工智能的发展正在从判别式走向生成式
 - **判别式模型**：给定一组图片，判别式模型可以学习识别图像中的对象，例如猫、狗、汽车等。
 - **生成式模型**：给定一组图片，生成式模型可能会尝试理解这些图片的生成过程，包括对象在图像中的布局、光照条件等，然后可以用这个理解生成新的类似图片。
- 人机交互方式的变化
 - 命令行
 - 图形化用户界面**GUI**
 - 自然语言**Chat**
- 模型本身并不直接产生价值
 - 基于基础大模型开发的**AI原生应用**才是模型存在的意义。

AI原生应用

- 新的交互设计LUI - **Chat 模式**
- 理解、生成、推理、记忆等
- No App - 每个应用的交互都**不超过两级菜单**
- **提示词工程**（Prompt Engineering）

复杂任务Prompt示例一

■ You:

- 执行以下操作：
- **1-**用一句话概括下面用三个反引号括起来的文本。
- **2-**将摘要翻译成英语。
- **3-**列出每个人名。
- **4-**输出一个 **JSON** 对象，其中包含以下键：**english_summary**, **num_names**。
- **5-**如果这是一个积极的故事，请输出**True**，否则输出**False**。
- 请用换行符分隔您的答案。
- `"""在一个迷人的村庄里，兄妹杰克和吉尔出发去一个山顶井里打水。
\\ 他们一边唱着欢乐的歌，一边往上爬，\\ 然而不幸降临——杰克
绊了一块石头，从山上滚了下来，吉尔紧随其后。\\ 虽然略有些摔
伤，但他们还是回到了温馨的家中。\\ 尽管出了这样的意外，他们
的冒险精神依然没有减弱，继续充满愉悦地探索。 """`

■ ChatGPT:

“软件开发技术实训”助教

■ Telegram机器人

<https://t.me/SoftwareDevelopmentTrainingBot>

角色

你是武汉理工大学计算机智能学院《软件开发技术实训》课程的专业教师，专门对计算机科学与技术卓越班的本科生进行精准教学。

技能

技能1：教授计算机专业知识和技巧

- 全面理解和掌握计算机科学各领域的知识和技能，以便有效地答疑解惑，增进学生在课堂上的学习。
- 运用Markdown语法编写结构化文本，辅助学生更深入、清晰地理解和记忆知识点。

技能2：简化复杂计算机概念的解读

- 采用通俗易懂的语言解释掌握从基础到复杂的计算机科学概念。
- 利用丰富的图文教材和资料，使复杂的计算机科学概念变得易懂和接受。

约束条件

- 只讨论与计算机科学和此课程主题相关的内容，避免涉及与计算机科学无关的话题。

GitHub Copilot

- Copilot 结合了 **GitHub** 的代码存储库和开源社区的力量，以及 **OpenAI** 的自然语言处理和机器学习能力，为开发人员提供智能化的代码建议和生成功能。






GitHub Copilot Chat 的用例

- 生成单元测试用例
- 说明代码和建议改进
- 建议代码修复
- 回答编码问题



如何使用大模型辅助编程？

■ 不同场景下用不同的方法

| | 优点 | 缺点 | 场景 |
|--|---------------------------|-----------------------------|-------------------|
|  人 | 逻辑复杂缜密，可以完成比较复杂的开发任务 | 写代码效率低 成本高 | 复杂业务逻辑， 核心引擎 |
|  Co-Pilot | 整合在开发工具中，快速复用/书写类似代码 | 准确率不高，生成代码段比较短，逻辑简单 | 辅助编程，复用代码 |
|  ChatGPT | 可以编写较复杂的业务代码，特别是有类似案例的情况下 | 需要code-review，错误隐藏的更深，缺乏创造性 | 有类似场景代码，自动生成新场景代码 |
| 私有化大模型 | 数据更安全，更懂你的业务 | 准确率和模型正相关，目前在70%左右 | Txt2SQL，运维脚本 |

个人发展：提升 AI 不擅长的能力

- 假定 AIGC 能提升一个团队 20% 的效能，那么从管理层来说，他们会考虑**减少 20% 的成员**。
- 从短期来说，掌握好 AIGC 能力的开发人员，不会因这种趋势而被淘汰。而长期来说，本就存在一定**内卷的开发行业**，这种趋势会加剧。
- **AI 不擅长解决复杂上下文的问题**，比如架构设计、软件建模等等。
- 这是一个**全新的领域**，无需传统 AI 的各种算法知识，只需要懂得如何工程化应用。

AI原生带来的应用开发方式的变化

技术栈的变化:云原生应用更注重通用性的技术栈,如Java、Python、Node.js等,而AI原生则更加关注深度学习框架、自然语言处理、计算机视觉、可视化工具等

58.0%

工具链的变化:云原生应用采用Kubernetes、Docker等工具实现持续集成/持续交付(CICD),而AI应用需要一套专门用于AI开发、测试、部署和监控的工具链,例如TensorFlow、PyTorch等

57.0%

基础设施变化:AI原生应用开发将基于AI化改造的基础设施/开发平台(如GPU加速卡等高性能硬件算力资源),包含芯片、框架、模型、应用在内的全栈方案

55.0%

开发流程的变化:云原生软件开发流程通常包括需求分析、设计、编码、测试、部署及后续维护等步骤,而AI原生则更加关注数据准备、模型训练和评估等环节

46.0%

安全策略的变化:云原生应用通常采用防火墙、入侵检测、权限控制、认证授权等手段保障信息安全,而AI原生则更加关注模型安全、数据安全和合规性以及伦理道德等新挑战

33.0%

设计理念的变化:云原生应用开发通常考虑将AI作为附加功能提升应用性能、自动化水平等来改善用户体验,而AI原生则是产品/应用之初就考虑AI的使用,天然集成在应用中

27.0%

组织层面的变化:AI原生应用开发需要更加关注数据团队和业务团队的协作,以及数据科学家和工程师的结合

24.0%