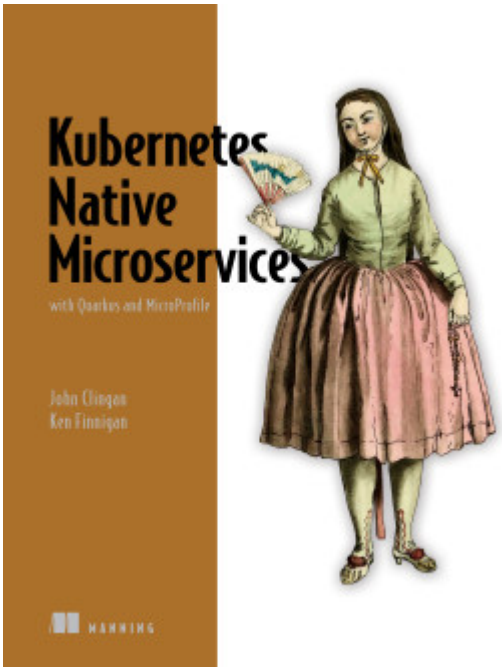
# Table of Contents

# 使用Quarkus和MicroProfile进行云原生开发




Manning Book

KNM笔记

源书PDF

## 内容

- Deploy enterprise Java applications on Kubernetes
- Develop applications using the Quarkus runtime
- Compile natively using GraalVM for blazing speed
- Create efficient microservices applications
- Take advantage of MicroProfile specifications

## 知识点

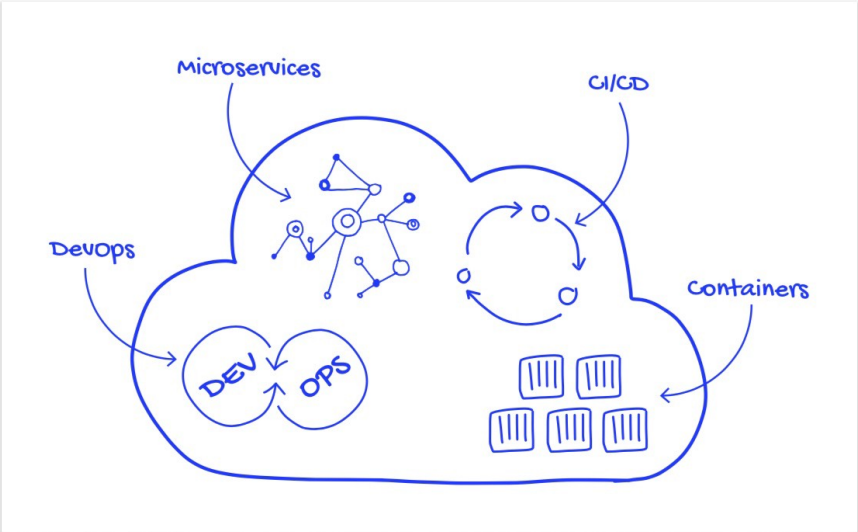
- JDK11+
- Graalvm
- Maven
- Quarkus
- MicroProfile
- Docker
- Kubernetes(or MiniKube)

# 介绍

以Quarks作为开发框架，开发基于云原生的应用程序，包括MicroProfile规范API的使用和在Kubernetes云平台中部署Quarkus应用程序。

# Cloud-Native

- 云原生是基于分布部署和统一运管的分布式云
- 以容器、微服务、DevOps等技术为基础建立的一套云技术产品体系。




# 技术基础

## JDK

- 采用版本: 11/17，OpenJDK, Zuul JDK, Temurin JDK, Oracle JDK都可
- Native包构建应采用Graalvm，选择安装相应平台及版本
- SDKMAN: 管理多版本JDK
- 推荐书箱：Java8 in Action

## Maven

- Java最流行的构建工具，相比Gradle更好用
- 了解依赖管理、常用插件、构建命令
- 推荐书箱：Mavan in Action，有中文翻译版

## 适用范围

- 有一定的Java基础开发者
- 编写过Spring应用的开发者
- 希望了解并增强云原生开发知识

# 术语解释

# 什么是微服务?



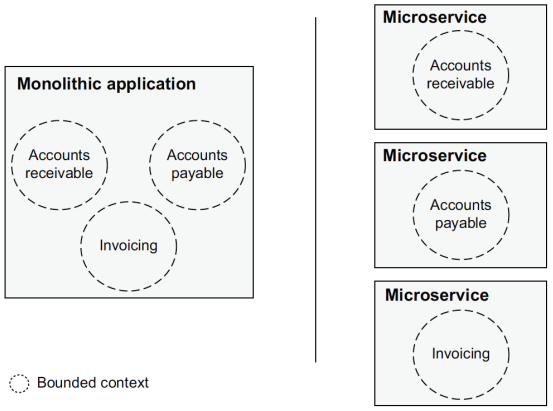



Figure 1.1 Bounded context: monolith vs. microservices

作者未从软件架构层面讲述微服务，要了解架构方面内容，推荐Enterprise Java Microservices

# 解决的问题

- JVM没有内置资源管理
- 应用更新与修补对其它应用的影响
- 各应用版本控制独立性
- 单应用的稳定性决定了整体项目应用程序
- 失去了对特别基础设计优化的条件

# 不仅是拆分，还需遵循隔离原则

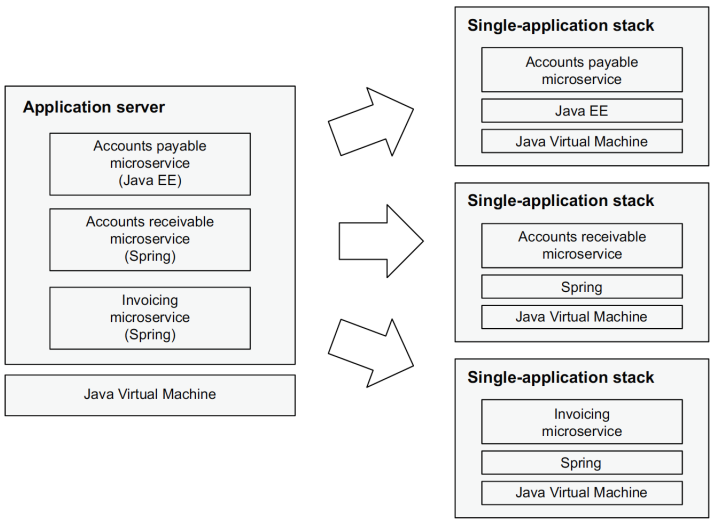



Figure 1.2 Application servers vs. single-application stacks

# 建议

- 因地制宜的使用微服务架构
- 小团队，小项目没有必要
- 微服务架构应结合完整的CI/DI流程，如Gitlab、Github、Jenkins等
- 生产资源在架构计时就应考虑

# Docker & Kubernetes

# Docker: 开源的应用容器引擎

依托Linux内核功能的虚拟技术， 能把应用程序部署到容器的引擎。

- 应用隔离
- 交付软件更容易
- 标准化交付
- 易水平扩展

## 应用隔离

每个容器间是隔离状态，互相不影响

## 交付软件更容易

- 传统交付软件过程：安装（包管理工具或源码编译）-> 配置 -> 运行
- 基于容器：拉取镜像 -> 运行
- 不同应用程序的依赖或配置不冲突

# 标准化交付

- 交付镜像包含了应用程序及所依赖的运行环境，简化了应用交付模式，可跨平台使用。
- 只需维护好Dockfile

# 易水平扩展

- 当需要扩展时，使用编排工具增加扩展数
- 如: kubectl scale --replicas=3 landing/web:v1.1

# Kubernetes

- 目前最流行,应用最广的云平台容器集群管理系统
- 用于自动部署、扩展和管理容器化应用程序
- 通过CLI或API能方便的管理及调度资源

主要功能有:

- 服务发现
- 水平扩展
- 负载均衡
- 自我修复
- 配置管理

# 为什么是Quarkus?

Quarkus官网，最新版v2.14.2

## Java开发常见问题

- 不够轻量
- 也不够快速

## 开发体验

- 代码实时运行时改动
- 无需重启JVM

## 社区规范支持

MicroProfile 4.0

## Reactive响应式编程

- 依赖Vert.x和Netty
- 非阻塞I/O交互

## 运行效率

### 启动到第一个HTTP响应用时

| Application | Traditional | Quarkus JVM | Quarkus Native |
| --- | --- | --- | --- |
| REST Application | 4.3(s) | 0.934(s) | 0.016(s) |
| CRUD Application | 9.5(s) | 2.03(s) | 0.042(s) |

### 内存使用

| Application | Traditional | Quarkus JVM | Quarkus Native |
| --- | --- | --- | --- |
| REST Application | 136(mb) | 73(mb) | 12(mb) |
| CRUD Application | 209(mb) | 145(mb) | 28(mb) |

## Native AOT Compiler(二进制可执行文件)

- JIT: Just-In-Time

- AOT: Ahead-Of-Time，缺点是构建Native包花费更多时间

## 对比其它

- Spring Native
- Micronaut

# MicroProfile

MicroProfile, Optimizing Enterprise Java for a Microservices Architecture

# MicroProfile规范

- 2016年由IBM,Eclipse,RedHat等联合定义
- 为微服务架构优化企业JAVA(Java EE)
- 常见的API可以被多个框架和实现或运行时使用

# 基础编程模型(MicroProfile 5.0 now)

- Config
- Fault Tolerance
- Health
- JWT RBAC
- Metrics
- Open API
- OpenTracing
- REST Client

# 区别Spring

- CDI（Context and Dependency Injection）基于“Java的上下文和依赖项注入”规范
- 而Spring是围绕依赖项注入容器的完整生态系统

# 常用

- JAX-RS (RESTful Services)
- CDI (Contexts and Dependency Injection)
- JSON-P (JSON Processing)

# 容器与云

对Docker、Kubernetes或Minikube进行初步认识与实践，详细内容或参考官方文档及以下书箱

- Docker In Action
- Kubernetes In Action

# Docker

## Windows/MacOS

Windows及MacOS系统推荐安装Docker Desktop，最新预览版可进行WASM(WebAssembly)。

## Linux

(略)

## docker cli

- docker version: 查阅版本
- docker info: 显示docker系统信息，重点关注Cgroup Driver: cgroupfs/systemd，Kubernetes要与之一致
- docker pull: 拉取镜像
- docker run: 运行镜像
- docker exec: 执行容器中的命令

## 国内镜像配置

```
{
"registry-mirrors": [
    "https://registry.cn-hangzhou.aliyuncs.com"
  ]
}
```

- in Docker Engine
- in /etc/docker/daemon.json

## docker buildx/build

### Dockerfile

```
FROM registry.access.redhat.com/ubi8/openjdk-11:1.14

COPY --chown=185 target/quarkus-app/lib/ /deployments/lib/
COPY --chown=185 target/quarkus-app/*.jar /deployments/
COPY --chown=185 target/quarkus-app/app/ /deployments/app/
COPY --chown=185 target/quarkus-app/quarkus/ /deployments/quarkus/

EXPOSE 8080
ENTRYPOINT exec java -jar /deployments/quarkus-run.jar
```

## buildx/build

```
docker buildx build -f src/main/docker/Dockerfile.jvm -t quarkus-mp/account-se
```

# Kubernetes

## Kubernetes in docker desktop

- 修改为国内镜像, Docker Engine registry-mirrors

## minikube

官网下载

## Kubernetes Cluster

(略)

## kubernetes cli

# 第一个Quarkus应用

- 创建Quarkus项目
- Live Coding
- 测试
- 运行与部署

# 创建项目骨架

通过以下三种方式创建Quarkus项目骨架，注意：Quarkus基于JDK 11或17，Maven 3.8.1+

# 在线项目生成器

官网在线生成

- 打开网站
- 修改项目配置信息：Group、Artifact、Version、Java Version、Build Tool
- 选择所需依赖项，最精简项选择: RESTEasy Classic 或 RESTEasy Classic JSON-B/RESTEasy Classic Jackson
- 生成并下载代码文件
- 使用IDE打开项目

# 使用Maven插件

```
mvn io.quarkus:quarkus-maven-plugin:2.1.3.Final:create \
-DprojectGroupId=quarkus \
-DprojectArtifactId=account-service \
-DclassName="quarkus.accounts.AccountResource" \
-Dpath="/accounts"
```

# 手动

对Maven项目管理及Quarkus组件较熟悉可以使用该方法

# 运行

- mvn quarkus:dev
- mvn package & java -jar target/quarkus-app/quarkus-run.jar
- 访问 http://localhost:8080

# REST API

在已创建有Quarkus应用骨架下增加REST资源访问，完成获取账户服下账户列表、账户对象，以JSON格式输出，观察启动控制台日志，它是Live Coding。

# 增加REST资源

## 准备Account对象

```
public class Account {
    public Long accountNumber;
    public String customerName;
    public Account(Long accountNumber, String customerName) {
        this.accountNumber = accountNumber;
        this.customerName = customerName;
    }
    public Long getAccountNumber() {
      return accountNumber;
    }
    public void setAccountNumber(Long accountNumber) {
      this.accountNumber = accountNumber;
    }
    public String getCustomerName() {
      return customerName;
    }
    public void setCustomerName(String customerName) {
      this.customerName = customerName;
    }
}
```

```
@Path("/accounts")
public class AccountResource {
    Set<Account> accounts = new HashSet<>();
    @PostConstruct // A
    public void setup(){
      accounts.add(new Account(1L, "Mary"));
    }
}
```

- Class 类
- 以@Path标注REST资源路径
- 通常以Resource结尾(类似Spring应用以Controller结尾)

## GetAccounts查询账户列表

```
@GET // A
@Produces(MediaType.APPLICATION_JSON) // B
public Set<Account> getAccounts() { // C
    return return accounts;
}
```

- A - 指定请求方法
- B - 表明响应体为JSON格式
- C - 返回一个账户对象集合
- 未指定Path，则为主类Path请求路径

### GetAccount查询账户对象

```
  @GET
  @Path("/{accountNumber}")
  @Produces(MediaType.APPLICATION_JSON)
  public Account getAccount(@PathParam("accountNumber") Long accountNumber){
    Optional<Account> account = accounts.stream().filter(acct -> acct.getAccou
    return  account.orElseThrow( () -> new WebApplicationException(String.form
  }
```

## JAX-RS 说明

所在包路径：javax.ws.rs.*

- @Path
- @GET
- @PathParam
- @Produces
- @PostConstruct

## 验证

使用curl或Postman请求以上定义的路径，能正常响应内容。

# 打包

Quarkus提供基于JVM运行的jar文件打包和基于Graalvm的二进制可执行文件打包。

# JVM

## Docker

- mvn clean package -Dquarkus.container-image.build=true
- 自动生成容器镜像, docker images 查看

# Native

使用AOT编译方式打包为一个二进制可执行文件

## Requirement

- Graalvm，选择对应版本，需对应操作系统平台、Java版本及CPU架构，Release Notes查阅对应Java版本
- JAVA_HOME及PATH需要相应更改，推荐使用sdkman管理
- Native-image: gu install native-image

## 配置

- Maven中增加名称为native的profile，并配置构建设置，示例如下代码块
- mvn clean package -Pnative
- 运行target目录下可执行文件

```
<profile>
    <id>native</id>
    <activation>
        <property>
            <name>native</name>
        </property>
    </activation>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-failsafe-plugin</artifactId>
                <version>3.0.0-M7</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>integration-test</goal>
                            <goal>verify</goal>
                        </goals>
                        <configuration>
                            <systemPropertyVariables>
                                <native.image.path>${project.build.directory}/
                            </systemPropertyVariables>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
    <properties>
        <quarkus.package.type>native</quarkus.package.type>
    </properties>
</profile>
```

## 构建Docker镜像

- mvn clean package -Pnative -Dquarkus.native.container-build=true
- docker buildx build -f src/main/docker/Dockerfile.native -t quarkus/account-service-native .
- docker images

# Deploy & Running

## Jar

## Docker

## Kubernetes

- 验证K8s环境
-

## 实际过程

- 开发人员编码
- 提交仓库: Github、Gitlab
- CI/DI: [测试]、构建镜像
- 部署工具持续部署: Helm、Kind等

# 开发微服务

# 可观测性