Docker镜像构建实践

基于Ubuntu平台的容器化解决方案，从基础构建到优化瘦身的完整实践

小组成员：王奥林，杨志炜，李翔，邱柏豪，赵子航

汇报人：赵子航

项目概览

01

核心技术

Docker 24.0.5 + Docker Compose 2.4 + Python Flask应用栈

02

关键功能

双版本Dockerfile、端口映射、卷挂载、环境变量传递

03

优化成果

多阶段构建、slim镜像、非root用户运行实现镜像瘦身

背景分析

云原生时代的挑战

当前痛点

- 镜像臃肿：构建工具残留导致体积过大
- 安全风险：root权限运行存在提升风险
- 配置混乱：缺乏统一版本管理规范
- 验证困难：难以保证镜像质量
- 复现性差：异构环境难以复现

解决目标

- 编写可复现的Dockerfile
- 完整演示运行参数配置
- 建立规范版本管理机制
- 实现标准化验证体系
- 确保跨环境一致性

核心技术原理

01 容器化与虚拟化

轻量级虚拟化技术，共享主机内核，实现快速启动和低资源开销

02 镜像分层与缓存

分层存储结构，通过缓存机制和多阶段构建优化镜像体积

03 最小权限原则

非root运行、命名空间隔离、资源限制确保容器安全

系统架构

整体方案设计

基础构建

编写Dockerfile，配置依赖环境

镜像优化

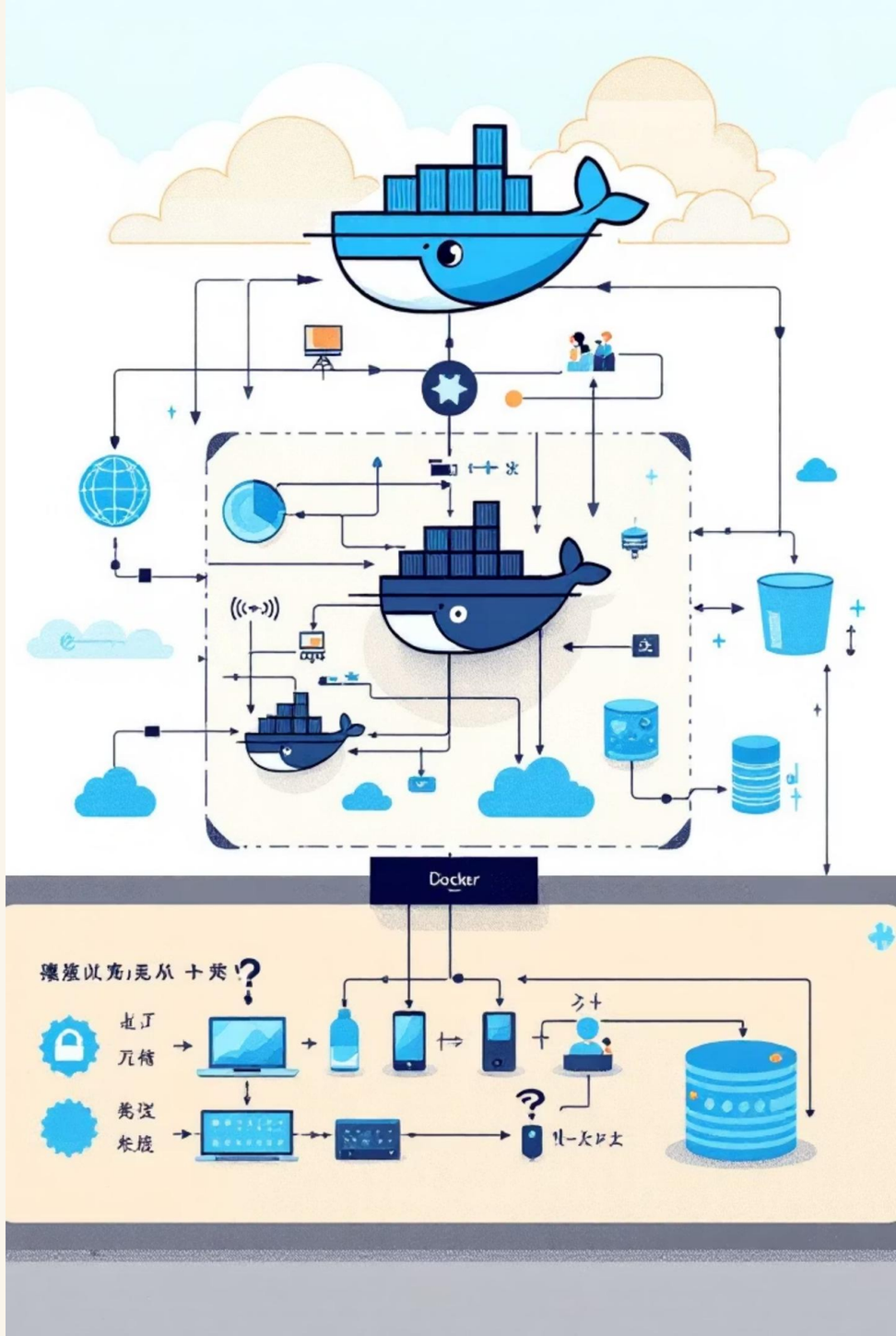
多阶段构建，减少层数

参数配置

端口映射、卷挂载、环境变量

验证部署

自动化测试，确保质量



实验环境配置

硬件资源

- 机器数量：1台
- CPU：4核
- 内存：8GB
- 网络：校园网环境

软件版本

- 虚拟化：VMware
- Docker：24.0.5
- Docker Compose：2.4
- Python：Python3

实验结果

关键验证成果

1

镜像构建验证

成功构建基础版和优化版两个镜像，通过docker images命令确认

2

端口映射验证

8081和8082端口可正常访问，curl测试返回正确JSON响应

3

环境变量验证

环境变量正确传递至容器，/env端点返回app_name值

4

卷挂载验证

主机logs和config目录与容器目录成功同步

5

镜像瘦身验证

优化镜像体积显著小于基础镜像

6

健康检查验证

/health端点返回status正常状态

7

安全特性验证

优化容器使用appuser非root用户运行

运维问题与解决

权限问题

现象：非root用户无法写入日志文件

解决：在Dockerfile中正确设置目录权限和用户所有权

镜像层过多

现象：镜像体积较大，构建层数过多

解决：合并RUN指令，清理apt缓存，使用.dockerignore

健康检查失败

现象：容器启动后健康检查立即失败

解决：增加start-period参数，给应用足够启动时间

端口冲突

现象：多容器运行时端口被占用

解决：使用Docker Compose管理端口映射，避免硬编码

总结与展望

核心收获

- 掌握容器化技术核心原理与分层机制
- 实践云原生应用安全与可靠性设计
- 建立自动化运维与持续验证体系

改进方向

- 微服务架构拆分与API增强
- 集成CI/CD流水线和监控告警
- 安全扫描、密钥管理与零信任网络
- 使用Alpine镜像进一步瘦身至50MB以下