基于Docker的全栈持续集成

一、课前准备

- 预习Linux基础知识
- 预习PM2基础知识

二、课堂主题

使用Docker部署一个全栈程序

- 前端Vue-element-admin
- 反向代理Nginx

三、课堂目标

- 掌握Docker容器基本操作
- 掌握Docker部署全栈应用的方法
- 掌握利用Github Webhook完成持续集成

四、知识要点

1. Docker是什么

- 操作系统层面的虚拟化技术
- 隔离的进程独立于宿主和其它的隔离的进程 - 容器
- GO语言开发

2. 特点

- 高效的利用系统资源
- 快速的启动时间
- 一致的运行环境
- 持续交付和部署
- 更轻松的迁移

3. 对比传统虚拟机总结

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 `MB`	一般为 `GB`
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

4. 核心概念

- 镜像
- 容器
- 仓库

5. Docker使用

构建一个Nginx服务器

1. 拉取官方镜像

```
# 拉取官方镜像
docker pull nginx

# 查看
docker images nginx

# 启动镜像
mkdir www
echo 'hello docker!!' >> www/index.html

# 启动
# www目录里面放一个index.html
docker run -p 80:80 -v $PWD/www:/usr/share/nginx/html -d nginx

# 查看进程
docker ps
docker ps -a // 查看全部

# 伪终端 ff6容器的uuid
docker exec -it ff6 /bin/bash

# 停止
docker stop ff6

# 删除镜像
docker rm ff6
```

6. Dockerfile 定制镜像

```
#Dockerfile
FROM nginx:latest
RUN echo '<h1>Hello, Kaikeba!</h1>' > /usr/share/nginx/html/index.html
```

```
# 定制镜像
docker build -t nginx:kaikeba .

# 运行
docker run -p 80:80  nginx:kaikeba
```

构建前端应用

- 前端静态化

```
# nginx/conf.d/default.conf(名字千万别写错)
server {
    listen       80;
    # server_name  www.josephxia.com;
    location / {
        root   /var/www/html;
        index  index.html index.htm;
    }
}
```

```
#Dockerfile
FROM nginx:latest
ADD ./dist /var/www/html
ADD ./nginx/conf.d /etc/nginx/conf.d
EXPOSE 80
```

```
# deploy.sh
docker build -t kkb-frontend git@gitlab.kaikeba.com:web_dev/docker_ci.git
docker stop kkb-frontend
docker rm kkb-frontend
docker run -p 80:80 -d --name kkb-frontend kkb-frontend
```

9. Github WebHook实现CI持续集成

启动NodeJS监听

```
var http = require('http')
var createHandler = require('github-webhook-handler')
var handler = createHandler({ path: '/webhooks', secret: 'myHashSecret' })
// 上面的 secret 保持和 GitHub 后台设置的一致
```

```
function run_cmd(cmd, args, callback) {
    var spawn = require('child_process').spawn;
    var child = spawn(cmd, args);
    var resp = "";

    child.stdout.on('data', function (buffer) { resp += buffer.toString(); });
    child.stdout.on('end', function () { callback(resp) });
}

http.createServer(function (req, res) {
    handler(req, res, function (err) {
        res.statusCode = 404
        res.end('no such location')
    })
}).listen(3000)

handler.on('error', function (err) {
    console.error('Error:', err.message)
})


handler.on('*', function (event) {
    console.log('Received *', event.payload.action);
    //   run_cmd('sh', ['./deploy-dev.sh'], function(text){ console.log(text) });
})

handler.on('push', function (event) {
    console.log('Received a push event for %s to %s',
        event.payload.repository.name,
        event.payload.ref);
        // 分支判断
        if(event.payload.ref === 'refs/heads/master'){
            console.log('deploy master..')
        }
    //   run_cmd('sh', ['./deploy-dev.sh'], function(text){ console.log(text) });
})


handler.on('issues', function (event) {
    console.log('Received an issue event for % action=%s: #%d %s',
        event.payload.repository.name,
        event.payload.action,
        event.payload.issue.number,
        event.payload.issue.title)
})
```

五、拓展点、未来计划、行业趋势

- Docker与云应用 和分布式技术结合
- K8s与Rancher
- Docker与DevOps Jenkins 自动化测试

六、总结

- Docker
- 服务器端接口调用
- 密码学 - SHA1
- 全局票据管理

七、作业

- 实现消息接口
- 调用一个服务器端接口