总览

1. 需求

   给开发人员提供一套环境用以部署代码以及静态资源并达到以下效果

   - 对开发友好，减少开发操作
   - 开发部署自动化，并能及时得到反馈
2. 环境
   - 系统：
     - [Linux（Ubuntu）]: https://www.ubuntu.com/download/server
   - 基础环境：
     - [java（JDK1.8）]: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
     - [maven]: http://maven.apache.org/download.cgi
     - [git]: https://git-scm.com/downloads
3. 工具
   - jenkins

     [jenkins]: https://jenkins.io/download/

     ```
     Jenkins可以做到持续编译和发布软件项目，这使得开发者很容易把他们的改动集成到项目中，还让用户能更加便利的获取编译和测试版本；
     ```

   - docker

     [docker]: https://www.docker.com/docker-ubuntu

     ```
     Docker是一个新的容器化的技术，它轻巧，且易移植，能够帮助你快速地测试、快速地编码、快速地交付，并且缩短你从编码到运行应用的周期。
     build once, configure once and run anywhere
     ```

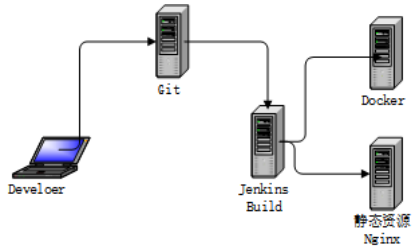
设计

1. 阶段

   为尽可能的解放开发人员，不同人员之间互相解耦，本例中分为代码提交、部署、验证三个阶段。
2. 部署
   1. 通过git拉取分支代码（可输入参数确定拉去分支/tag）
   2. 代码编译（本例使用java代码，通过maven -P 参数确定读取配置文件用以区分环境）
   3. 上传静态资源
   4. docker容器、镜像销毁并重新根据Dockerfile进行build，通过新image启动容器。
   5. 部署静态资源
   6. 完成部署，返回结果（可通过邮件、钉钉等形式）
3. 验证

   通过测试人工访问页面进行验证。另可通过shell、监控工具等进行访问获取http状态码，并同时监控进程死活

架构



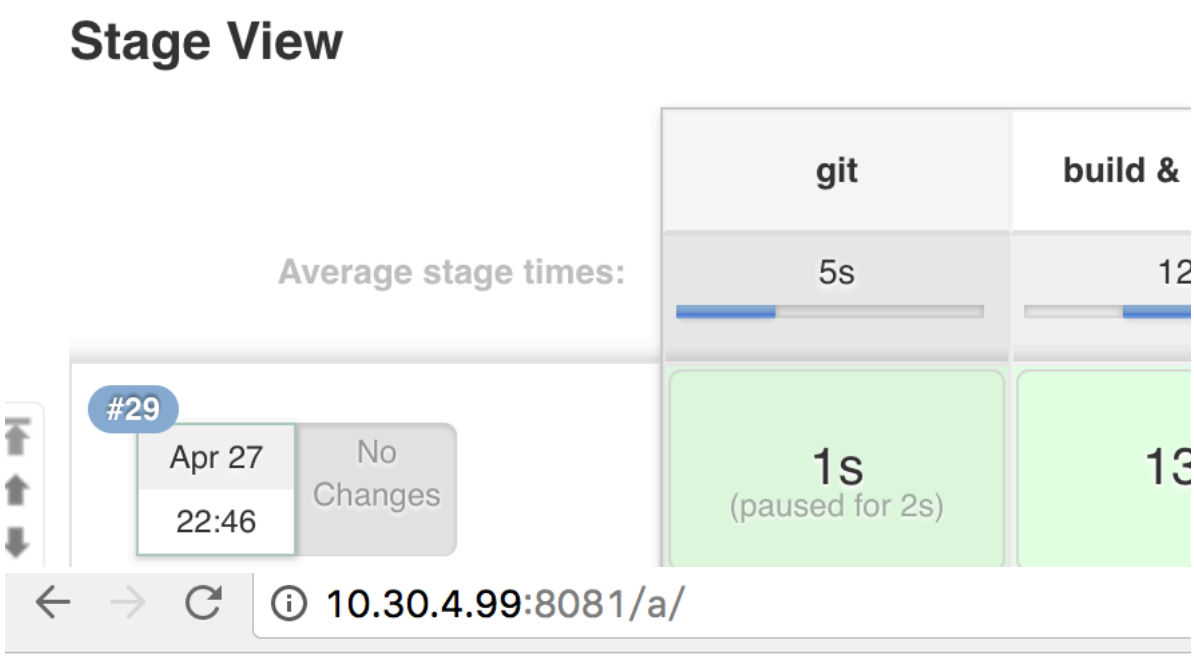
实现

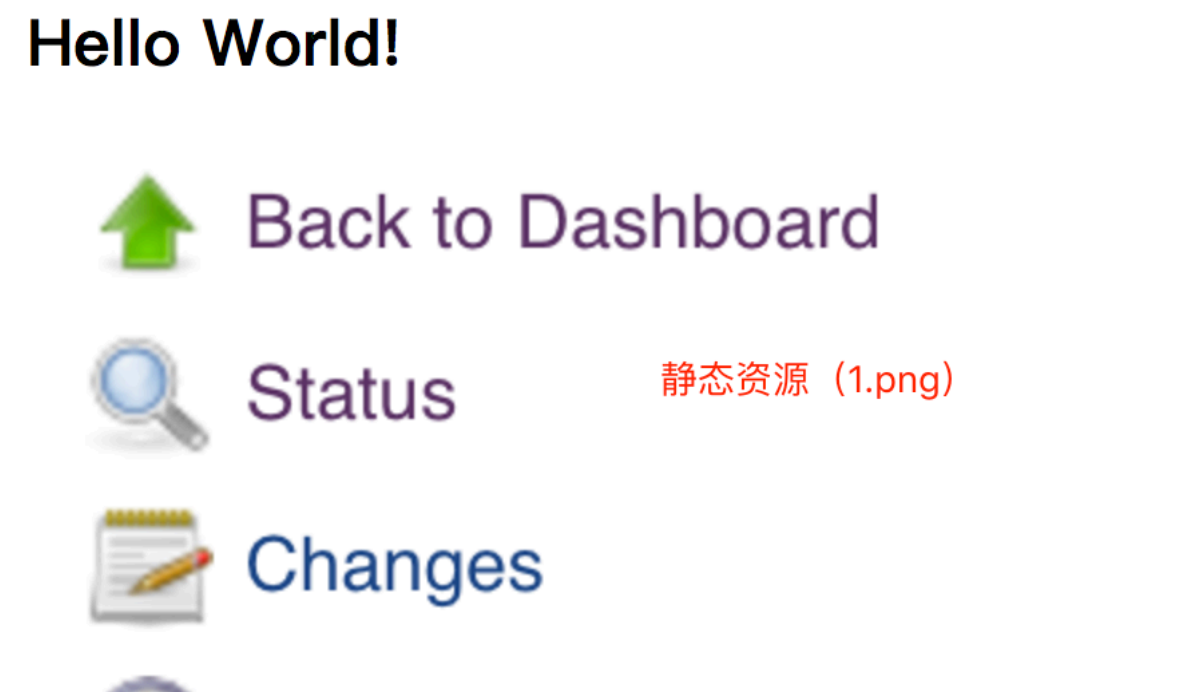
1. 代码提交至git（本例中使用[GitHub]:https://github.com/zhangrandl/testtest）
2. 将需要部署的静态资源上传至服务器指定目录（此例中为/home/zhang/static）

3. 进入jenkins点击Build with Parameters
4. 依次输入相应参数，点击开始构建

```
tag: git对应分支/tag
env: 选择对应环境（prod、test）
version: 版本号
```

5. 等待部署完成





附:

- pipeline代码

```
node("master"){
   stage 'git'
   git url:'https://github.com/zhangrandl/testtest.git', branch:'${tag}'
   input "是否开始测试？"    // 这里会阻塞等待用户的输入

   stage 'build & Deploy'
   echo "开始部署编译java程序"
   sh 'cd /var/lib/jenkins/workspace/test && mvn clean package install -Dmaven.test.skip=true'
   sh "rm -rf /home/jenkins/a.war"
   sh "mv /var/lib/jenkins/workspace/test/target/a.war /home/jenkins/test"
   echo "--------------------"
   sh "/home/jenkins/develop.sh"
   sleep 3

   echo "开始部署静态资源"

   sh "unzip -o /home/jenkins/static/static.zip -d /home/jenkins/${version}/"
   sh "rm -rf /usr/share/nginx/html"
   sh "ln -s /home/jenkins/${version} /usr/share/nginx/html"

   echo "部署成功."
 }
```

- develop.sh

```
docker stop docker ps | grep test|awk '{print $1}'

docker rm docker ps -a | grep test|awk '{print $1}'

docker rmi docker images | grep none | awk '{print $3}'

docker build -t test:1.0 /home/jenkins/test

docker run -v /var/log/tomcat/test/://usr/local/tomcat/logs/ -d -it -p 8081:8080 test:1.0
```

- Dockerfile

```
FROM docker.io/tomcat

COPY a.war /usr/local/tomcat/webapps/

EXPOSE 8080

CMD ["catalina.sh", "run"]
```

高可用

通常一个系统的压力瓶颈往往是在程序本身或者是数据库上，所以在高可用上，我建议前端nginx使用upstream建立集群，将压力分散在多节点。数据库方面会基于系统重要性做双机热备、主从等。如果是公有云，建议直接接入现有产品，比如阿里云的RDS，AWS的RDS等。如果读写压力较高经常造成服务器死锁，建议新增只读库用来缓解主库压力。程序方面尽量做多节点，通过nginx的upstream形成集群。如果前端压力非常大，则可以增加nginx节点来缓解，或者增加H5设备做均衡。

监控/报警

私有云建议使用zabbix监控系统，Zabbix 是一个高度集成的网络监控解决方案，可以提供企业级的开源分布式监控解决方案。对于系统监控，建议从多维度，多角度来考虑，比如监控项目是否可访问，可以从进程死活、监控页面返回是否正常、服务器数据流量是否异常等多项指标来综合衡量。

公有云建议使用产品本身自带的监控为主，zabbix为辅，使得监控更加准确及时

日志

由于项目存在多节点，日志建议进行统一格式进行输出，可以从配置复杂度、查看难易程度等方面衡量

1.输出至各自节点

优点：方便无需配置

缺点：查看麻烦，出现单节点故障需要去该节点进行查看。

2.输出至统一网络存储

优点：日志集中，容易查看，方便管理

缺点：多节点对存储进行读写容易达到性能瓶颈

3.输出至elasticsearch/Hadoop

优点：日志集中，统计方便，可利用日志进行多种计算

缺点：成本高

建议使用elk作为日志存储，其中logstash进行日志读取发送，elasticsearch作为存储，kiabana作为前端展示。其中kiabana由于没有权限控制，可以使用grafana替代作为前端展示。如果日志量大的话，可以将logstash替换为heka，在用redis做缓存队列。