ThoughtWorks® | IT大咖说 知识共享平台

ThoughtWorks®

持续集成下的 PIPELINE代码化实施实践

Pipeline As Code In CI/CD Practices

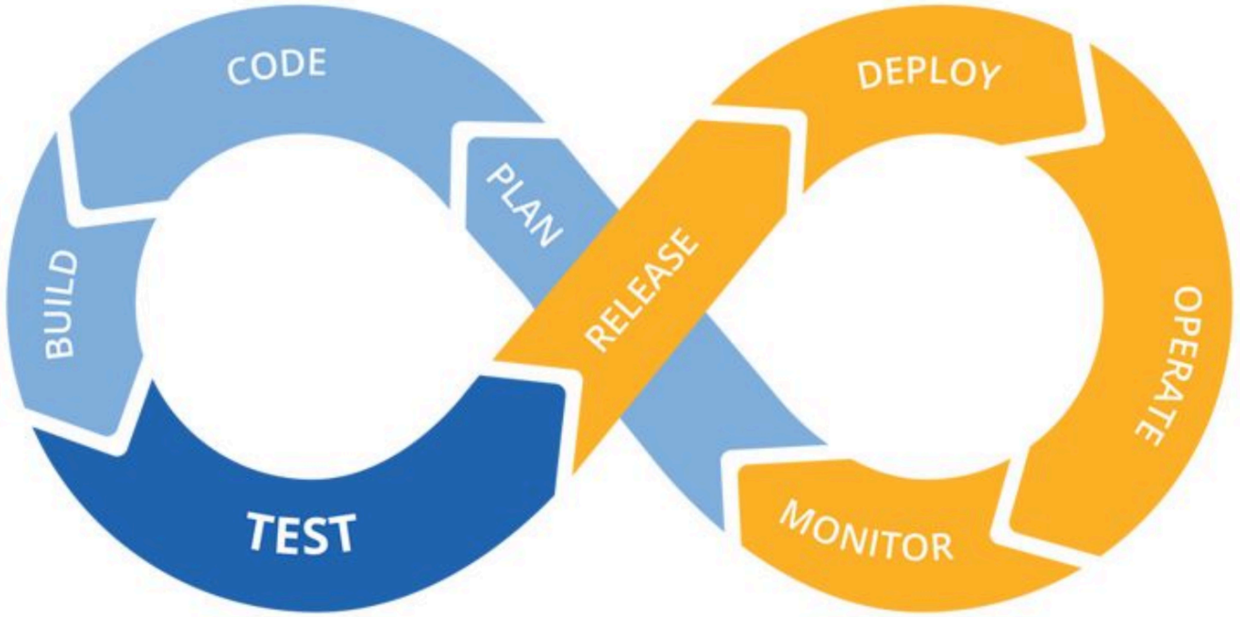
宋琦(俊毅)@ThoughtWorks

持续集成与部署流水线

THOUGHTWORKS

思特沃克

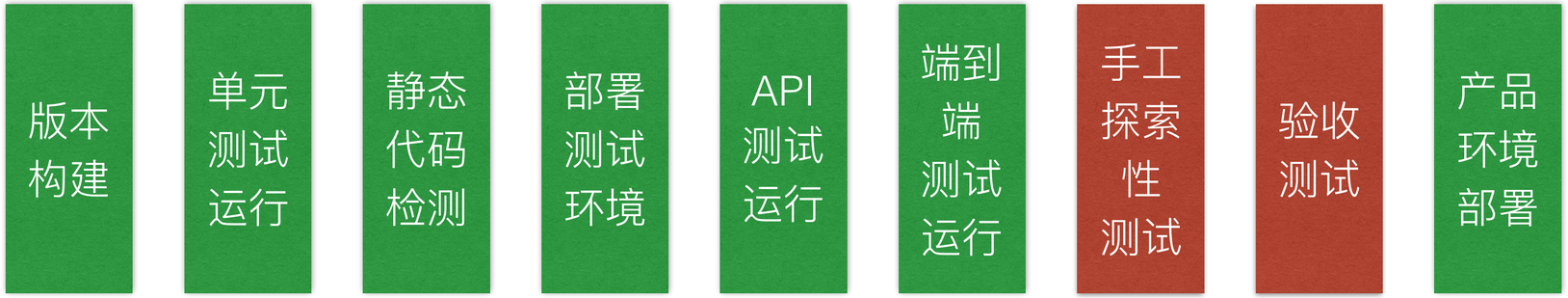
问题 - 如何提升或打造产品的高竞争力

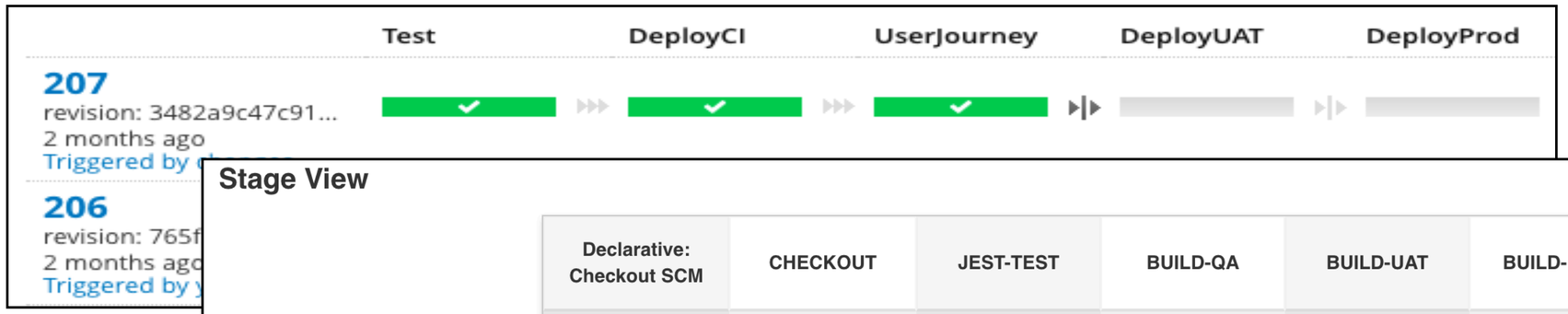


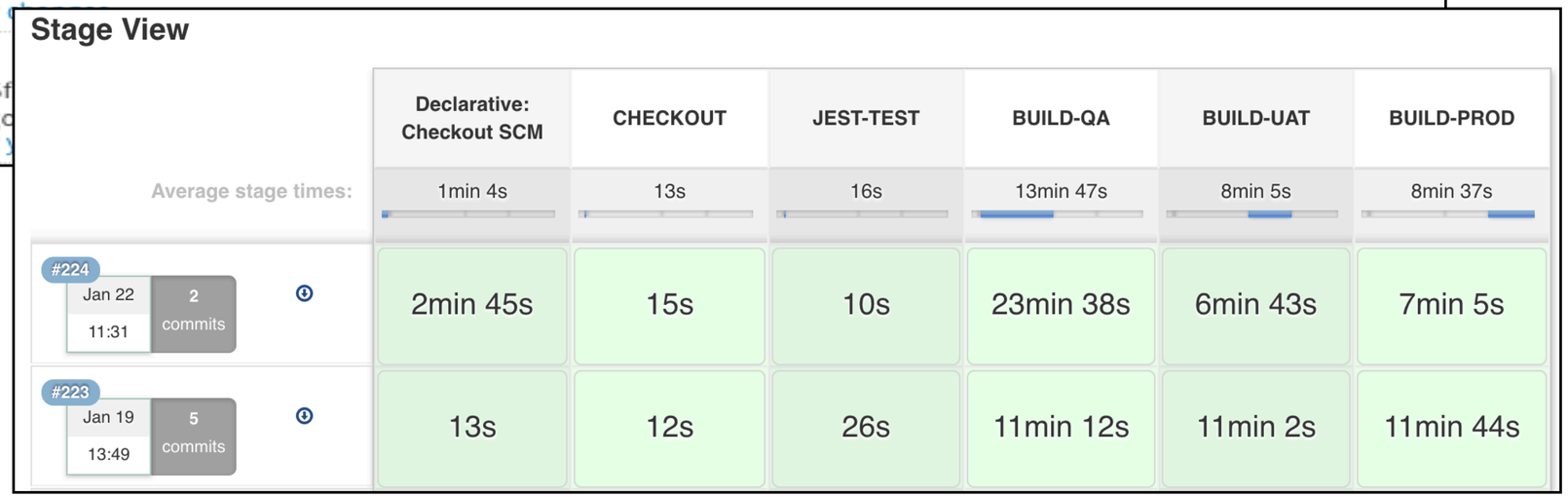
构建快速价值交付能力 - 持续集成

部署流水线（Deployment Pipeline）

是一个应用程序或软件系统从构建、部署、测试和发布整个过程的自动化实现

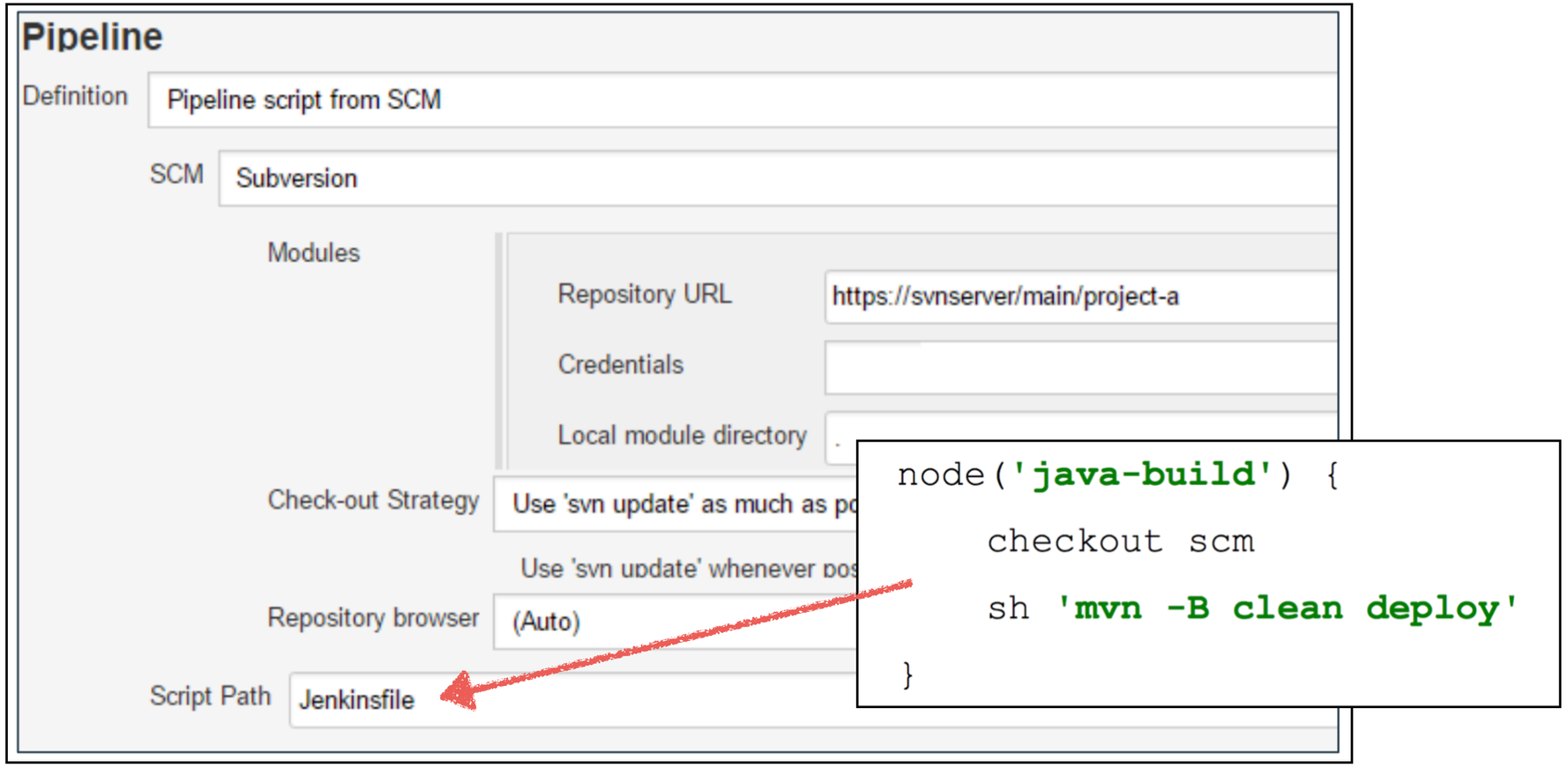






基于Jenkins 2.0 使用DSL语言进行步骤定义

Jenkins 从2.0版本开始提供了Pipeline功能，支持通过Jenkinsfile文件创建pipeline。



Jenkins Pipeline 步骤定义常用关键字定义如下

- **stage**：定义开始一个新的Pipeline阶段
- **node**：定义一个阶段中的可执行步骤
- **checkout**：由SCM(Git或SVN)中检出源码
- **sh**：运行命令行或shell脚本
- **bat**：运行Windows命令行或batch脚本
- **junit**：发布JUnit测试结果

```
stage 'Compile'
node('java-build') {
    checkout scm
    sh 'mvn -B clean deploy'
    junit testResults: 'build/test-results/*.xml'
}
stage 'Automated tests'
node('qa-environment') {
    …
```

Jenkins Web项目实例

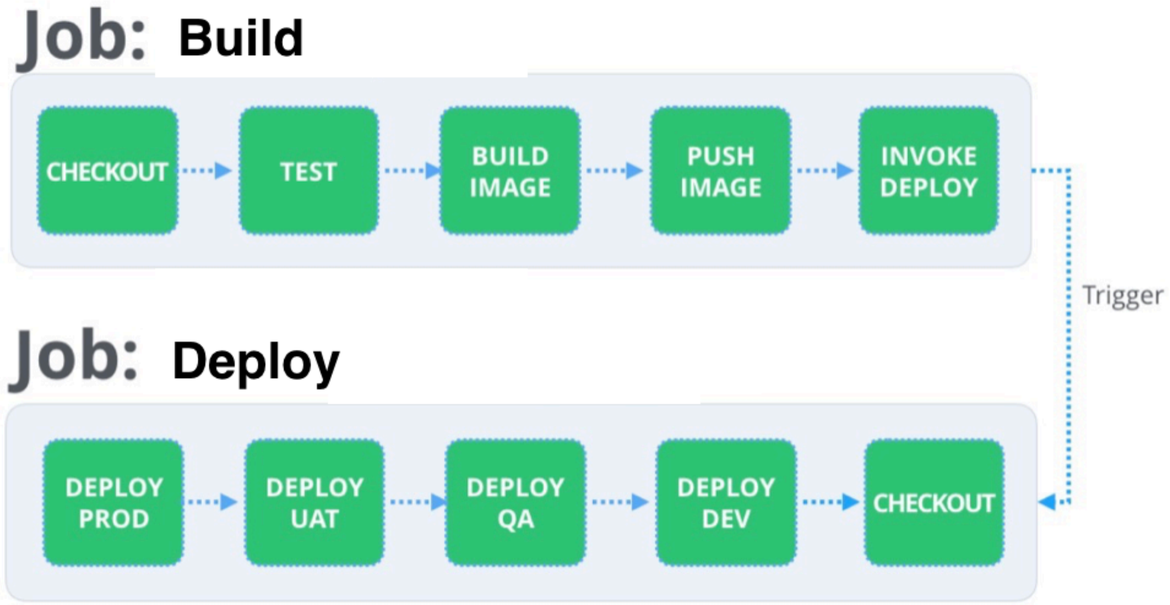
Stage View

	CHECKOUT	TEST	BUILD_IMAGE	PUSH_IMAGE	INVOKE_DEPLOY
Average stage times: (Average full run time: ~6min 36s)	5s	2min 48s	1min 32s	2s	35s
#626 Jan 23 15:30 No Changes	5s	2min 51s	2min 32s	280ms	1min 2s

Stage View

	Declarative: Checkout SCM	CHECKOUT	DEPLOY_DEV	DEPLOY_QA	DEPLOY_UAT	DEPLOY_PROD
Average stage times: (Average full run time: ~34s)	3s	3s	14s	6s	17ms	18ms
#645 Jan 26 16:28 No Changes	3s	3s	29ms	15s	17ms	17ms

Jenkins 编译部署策略



需要如下参数用于构建项目:

IMAGE_ID	01
	Upstream BUILD_NUMBER
IMAGE_NAME	web-release
	Upstream IMAGE NAME
ADCC_ENV	dev_web

开始构建

Jenkins File - Build


```
pipeline {
    agent any
    stages {

        stage('CHECKOUT') {
            steps {
                git  url: "${env.SCM_URL}", credentialsId: "${env.SCM_CREDENTIALS}", branch: "${env.SCM_BRANCH}"
            }
        }

        stage('TEST') {
            steps {
                ...
                sh 'docker run --rm -i xxxx bitnami/node:7.7.2-r0 sh -c \'npm run lint && npm test\''
            }
        }

        stage('BUILD_IMAGE') {
            steps {
                sh 'docker run --rm -i -v $(pwd):/data -w /data bitnami/node:7.7.2-r0 sh -c \'npm run build\''
                ...
                sh './auto/build ${JOB_NAME} ${BUILD_NUMBER}'
            }
        }

        stage('PUSH_IMAGE') {
            steps {
                sh "./auto/pushadcc ${JOB_NAME} ${BUILD_NUMBER} ${env.SCM_NEXUS}"
            }
        }

       stage('INVOKE_DEPLOY') {
            steps {
                build job: 'xxx-deploy', parameters: [string(name: 'IMAGE_NAME', value: String.valueOf(BUILD_NUMBER))...]
            }
        }
    }
}
```

Jenkins File - Deploy


```
pipeline {
    agent any

    parameters {
        string(name: 'IMAGE_ID', defaultValue: '01', description: 'Upstream BUILD_NUMBER')
        string(name: 'IMAGE_NAME', defaultValue: 'web-release', description: 'Upstream IMAGE NAME')
    }

    stages {
        stage('CHECKOUT') {
            steps {
                git url: "${env.SCM_URL}", credentialsId: "${env.SCM_CREDENTIALS}", branch: "${env.SCM_BRANCH}"
            }
        }

        stage('DEPLOY_DEV') {
            steps {
                ...
                sh './web-deploy/auto/deploydev'
            }
        }

        stage('DEPLOY_QA') {
            steps {
                ...
                sh './web-deploy/auto/deployqa'
            }
        }

        stage('DEPLOY_UAT') {
            steps {
                ...
                sh './web-deploy/auto/deployuat'
            }
        }

        stage('DEPLOY_PROD') {
            steps {
                ...
                sh './web-deploy/auto/deployprod'
            }
        }
    }
}
```

我们还缺少些什么？

持续交付 - 使用DSL语言定义基础设施

使用DSL语言进行基础设施定义与配置。使用DSL更容易通过描述性的语言定义基础设施，也有助于代码重用。团队成员能建立起使用DSL更容易通过描述性的语言定义基础设施，也有助于代码重用。团队成员能建立起共同理解，从而维护脚本。共同理解，从而维护脚本。不要直接登录服务器上用命令改变基础设施环境与配置。

```
---
- hosts: local
  tasks:
    - name: Install Nginx
      apt: pkg=nginx state=installed update_cache=true
      notify:
        - Start Nginx

  handlers:
    - name: Start Nginx
      service: name=nginx state=started
```

持续交付 - 基础设施的质量内建

在编写环境代码的配置时，也要编写对环境的测试。确保所有服务器都进行了正确的配置，遵守了所有的安全规则，也对网络连通性等进行了验证。我们一般提倡将测试代码和配置代码放在一起维护。这样配置代码更新时，能保证测试代码也被及时更新。
一些典型的基础设施自动化测试工具有ServerSpec、Testinfra等。

```
def test_passwd_file(host):
    passwd = host.file("/etc/passwd")
    assert passwd.contains("root")
    assert passwd.user == "root"
    assert passwd.group == "root"
    assert passwd.mode == 0o644


def test_nginx_is_installed(host):
    nginx = host.package("nginx")
    assert nginx.is_installed
    assert nginx.version.startswith("1.2")
```

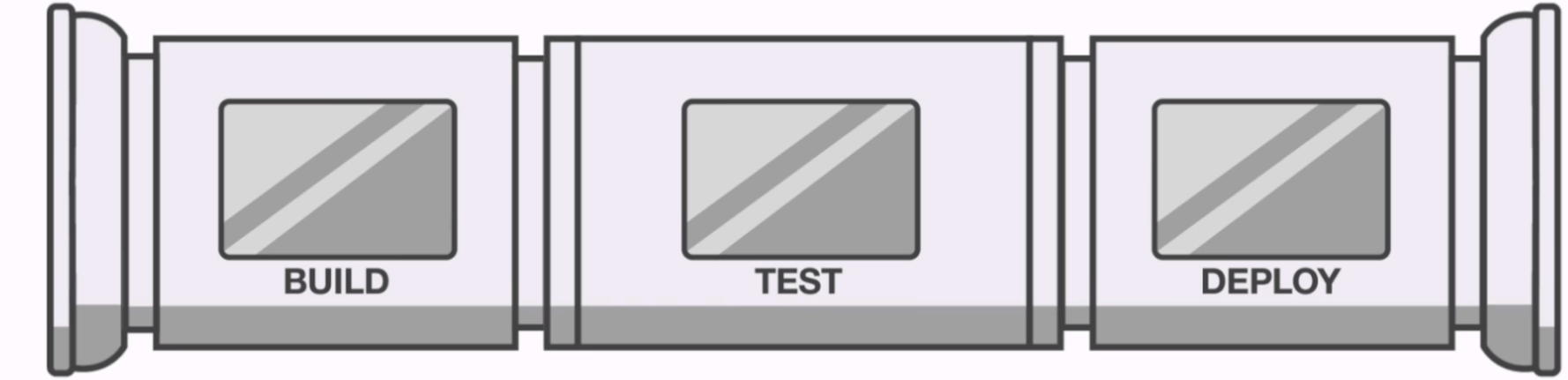
持续交付 - 基础设施变更版本化

在基础环境使用DSL语言进行定义并实现对环境的控制后，需要将所有基础设施代码使用版本工具管理起来。对每一次提交与变更做好变更管理，便于对变更的团队协作、记录与回溯。如使用GitLib、GitHub、Subversion等版本管理工具。

github
SOCIAL CODING

持续交付 - 基础设施的变更自动化

搭建基础设施配置pipeline对基础设施代码变更进行构建与验证。所有的环境变更都应该先修改环境定义脚本，由环境定义脚本触发pipeline上的Job进行对环境的变更与验证。
登录到服务器执行一些临时性命令是被坚决禁止的，因为这极有可能会破坏环境的一致性。
如使用Jenkins搭建Pipeline，并在deploy步骤后加入TestInfra、ServerSpec等工具支撑的验证步骤。



分阶段落地交付流水线

THOUGHTWORKS

思特沃克

Pipeline 建立各过渡阶段

初始阶段

关注点：

构建项目CI流水线

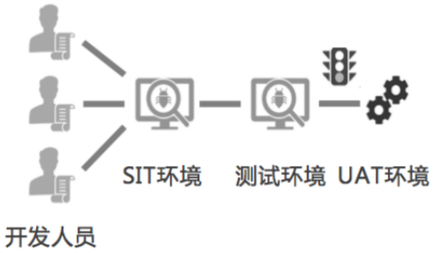
落地建议：

- 建立代码审核与质量检测流程与体系
  - 搭建CI环境
  - 主干开发、小步提交
  - 明确代码质量检测指标与标准
- 建立一键化部署脚本
- 建立标准化部署环境

过渡阶段

关注点：

完善测试结构，构建CD流水线



落地建议：

- 进一步完善测试结构
- 缩小单次发布体积，加快发布频率
- 在Pipeline中增加UAT环境，蓝绿部署策略
- 加入部分监控，如性能、稳定性等

最终阶段

关注点：

关注用户价值快速交付

落地建议：

- 基于用户最小价值快速交付
- 加入用户行为日志分析
- 加快用户需求反馈速率

完善Pipeline 交付体系

测试数据服务

作用：

提高测试效率

落地建议：

- 完善的初始化脚本
- 通过API接口实现相关功能调用

Mock服务

作用：

降低耦合

落地建议：

- 给每个第三方应用建立一套Mock服务
- 根据请求不同参数返回预先设计Mock API返回数值

日志服务

作用：

快速定位问题

落地建议：

- 使用日志框架输出日志并做好日志分级管理
- 参考ELK系统实现原理：

  ElasticSearch: 支持分布式全文搜索引擎

  Logstash: 支持收集、分析日志，并将其存储共以后使用

  Kibana: 汇总、分析和搜索重要数据日志

全方位的监控服务

作用：

第一时间通知与恢复服务

落地建议：

- 识别系统每个关键节点并监控
- 关键服务采用自动化脚本自动恢复或隔离
- 第一时间通知相关人员进行问题修复

全球视野 本土智慧

ThoughtWorks®

THANK YOU