



# Pimp your Continuous Delivery Pipeline with Jenkins workflow

---

Cyrille Le Clerc, CloudBees

# About Me



@cyrilleleclerc

Open Source



Cyrille Le Clerc

Xebia

CTO

CloudBees®

Product Management

DevOps, Infra as Code,  
Continuous Delivery



# About you



*Jenkins? CI? CD? Code Jenkins plugin?*

# Agenda

- Continuous Delivery
- Jenkins & Continuous Delivery
- Workflows in Jenkins today
- The new Jenkins Workflow Engine
- Jenkins Workflow Syntax Card
- Possible future

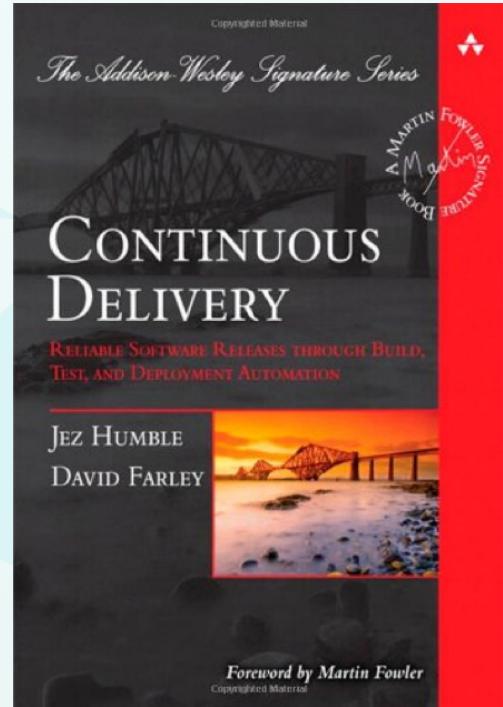


# Continuous Delivery

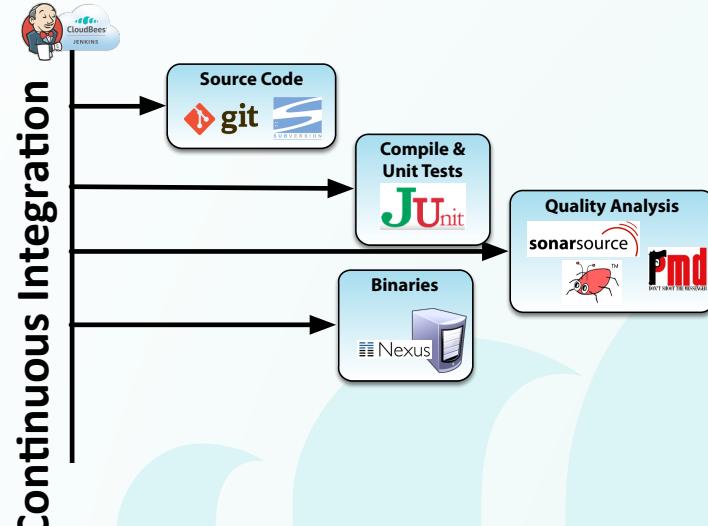
---

# Continuous Delivery

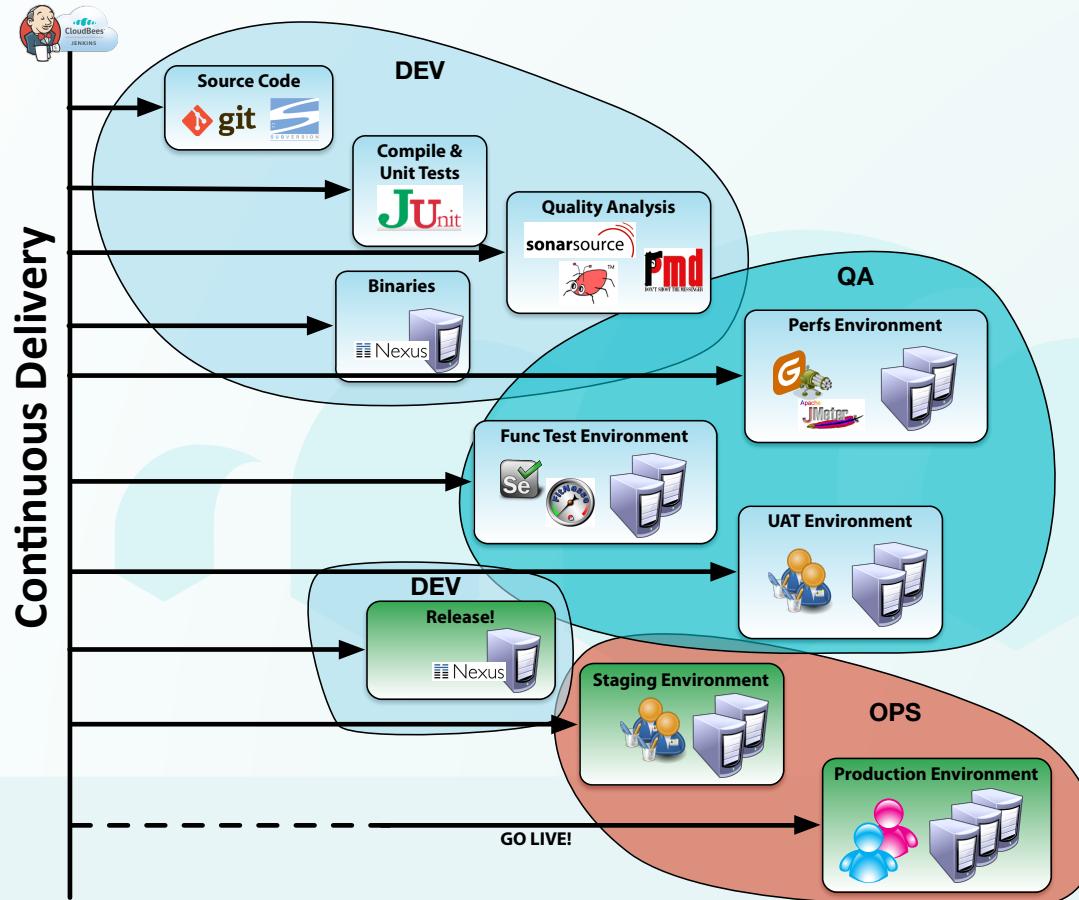
- “*Reliable Software Releases through Build, Test, and Deployment Automation*”
- DONE = Shippable into Production



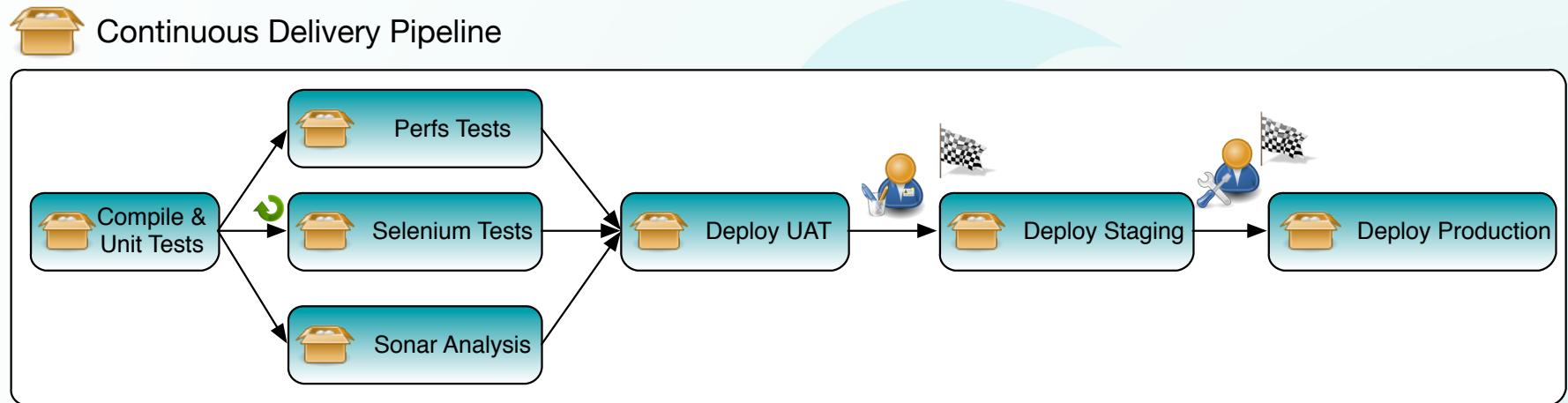
# Continuous Integration Steps



# Continuous Delivery Steps

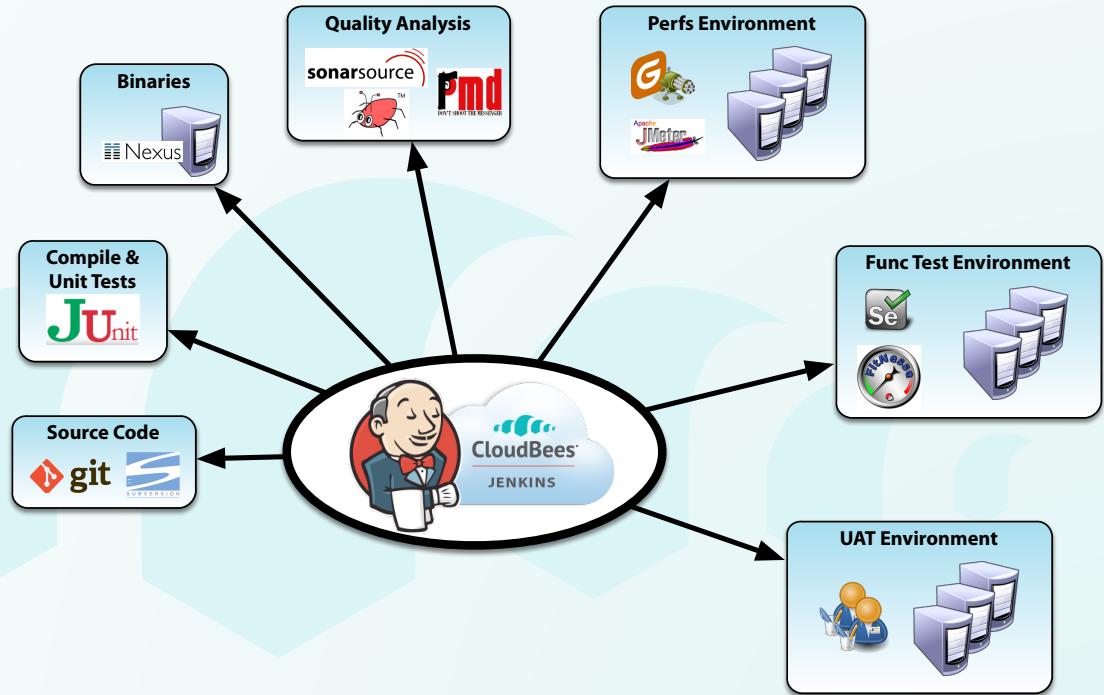


# Continuous Delivery Pipeline

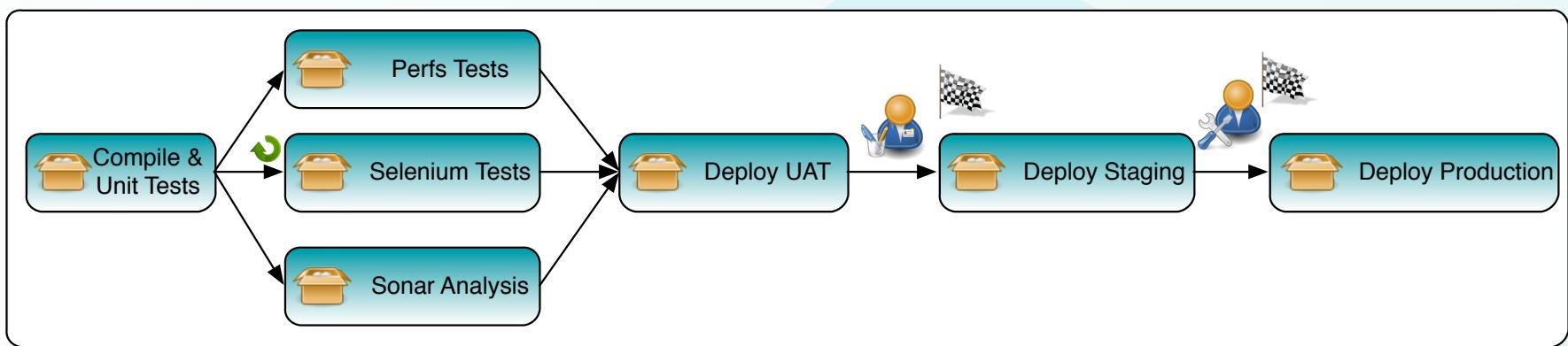


# Jenkins: Hub of Continuous Delivery

- Connect the dots
- It is the process



# CD Pipeline → Workflow

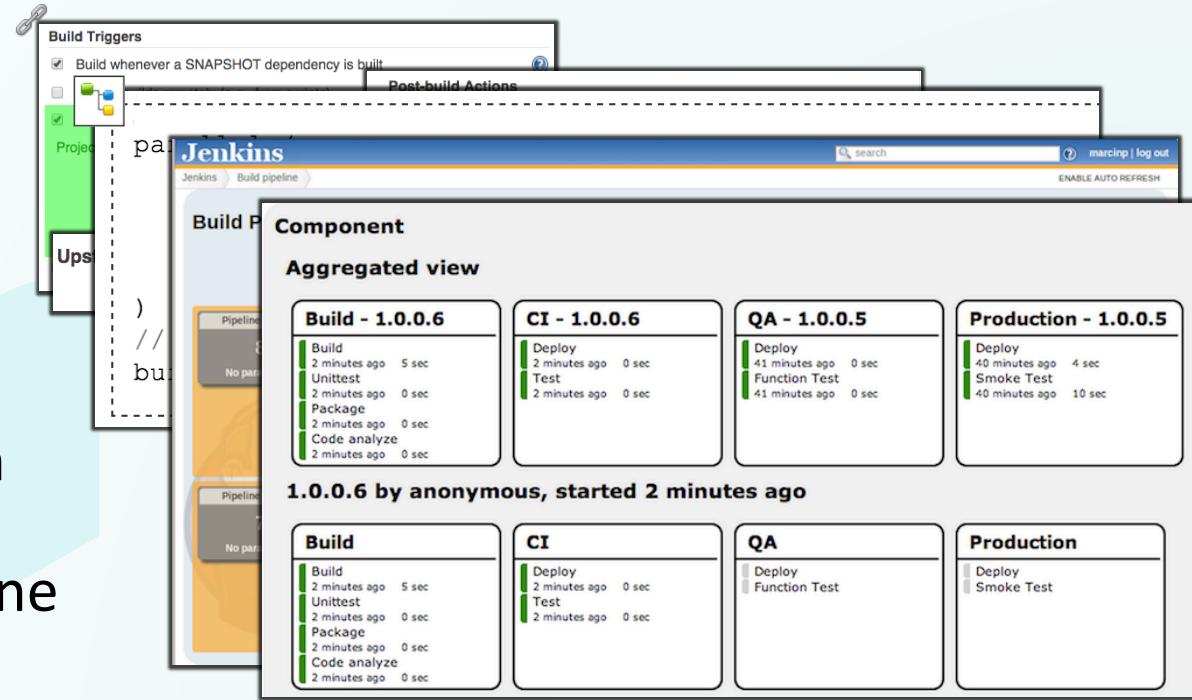


# What we need

- **Complex pipelines** involving multiple stages
- **Non-sequential logic** such as loops, forks ...
- **Long-running builds** must survive outages
- **Interactions with humans** including pauses, input
- **Restartable builds** in case of transient errors
- **Reusable definitions** to avoid duplication
- **Comprehensible scripts** with one clear definition

# Workflow until today

- Job chaining
- Build Flow Plugin
- Build Pipeline Plugin
- Build Delivery Pipeline



# Workflow until today

- Many atomic jobs
- Hard to share variables/state between jobs
- Limited logic
- Mix build triggers, parameterized build ...



# Build Flow Plugin

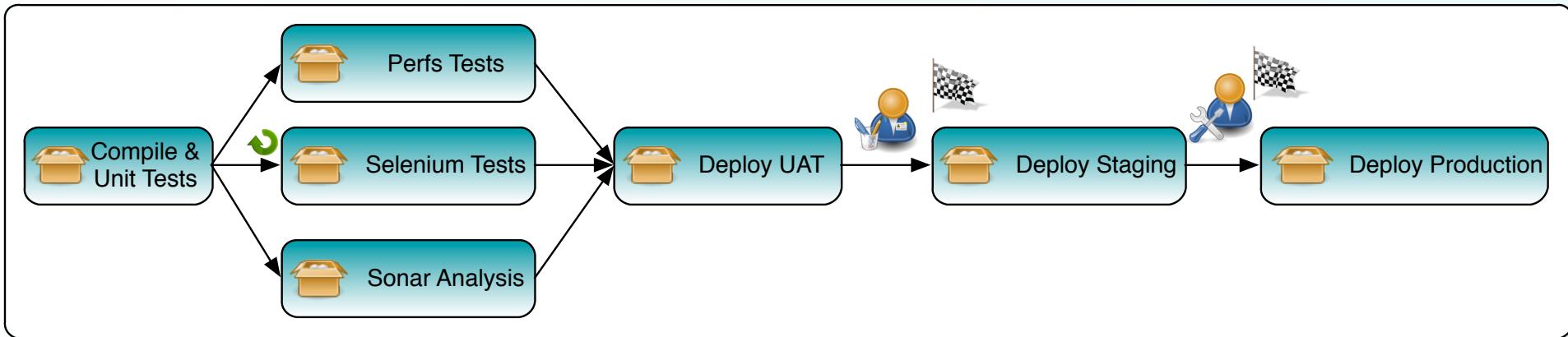
- Did have scriptability and extensibility
- Did not address configuration “sprawl”
- Disjointed view of what really ran
- No ability to survive restarts
- Almost good enough but could not go further



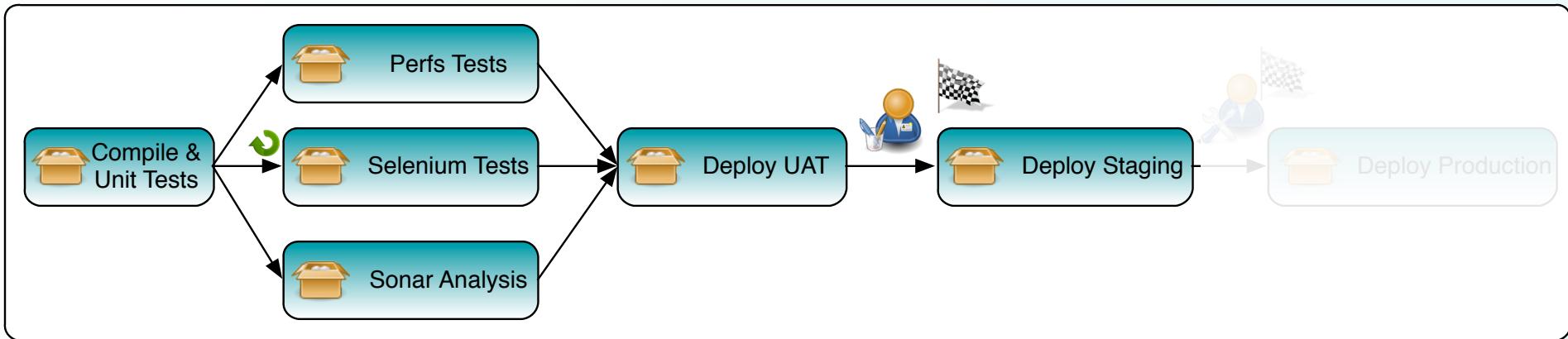
# Jenkins Workflow Engine

---

# Simplifying our sample workflow



# Simplifying our sample workflow



# What we need

- **Complex pipelines** involving multiple stages
- **Non-sequential logic** such as loops, forks ...
- **Long-running builds** must survive outages
- **Interactions with humans** including pauses, input
- **Restartable builds** in case of transient errors
- **Reusable definitions** to avoid duplication
- **Comprehensible scripts** with one clear definition



# New Workflow Syntax

```
1 stage 'DEV'
2 node('linux') {
3     // COMPILE AND JUNIT
4     git url: 'https://github.com/cyrille-leclerc/spring-petclinic.git'
5     sh 'mvn -o clean package'
6     archive 'target/petclinic.war, src, pom.xml'
7     step $class: 'hudson.tasks.junit.JUnitResultArchiver', testResults: 'target/surefire-reports/*.xml'
8 }
9
10 parallel(qualityAnalysis: {
11     // RUN SONAR ANALYSIS
12     node('linux') {
13         stage name: 'QUALITY_ANALYSIS', concurrency: 1
14         unarchive mapping: ['src': '.', 'pom.xml': '.']
15         sh 'mvn -o sonar:sonar'
16     }
17 }, performanceTest: {
18     // DEPLOY ON PERFS AND RUN JMETER STRESS TEST
19     node('linux') {
20         stage name: 'PERFS', concurrency: 1
21         unarchive mapping: ['src': '.', 'pom.xml': '.', 'target/petclinic.war': 'petclinic.war']
22         deployApp 'petclinic.war', perfsCatalinaBase, perfsHttpPort
23         sh 'mvn -o jmeter:jmeter'
24         shutdownApp(perfsCatalinaBase)
25     }
26 })
27
28 checkpoint 'ENTER QA'
29 input message: "Deploy to QA?", ok: "DEPLOY TO QA!"
30 // DEPLOY ON THE QA SERVER
31 node('linux') {
32     stage name: 'QA', concurrency: 1
33     unarchive mapping: ['target/petclinic.war': 'petclinic.war']
34     deployApp 'petclinic.war', qaCatalinaBase, qaHttpPort
35 }
```

# Key features

- Entire flow is one concise Groovy script
  - For loops, try-finally, fork-join ...
- Can restart Jenkins while flow is running
- Allocate slave nodes and workspaces
  - As many as you want, when you want
- *Stages* throttle concurrency of builds
- Human input/approval integrated into flow
- Standard project concepts: SCM, artifacts, plugins

# Groovy DSL vs. Graphical Workflow

- Familiar control flow construction
- Familiar “tools” for building abstractions
  - Functions, classes, variables, ...
- Workflow in version control
- As opposed to:
  - Graphical workflow designer

# Project setup

- One workflow is defined as a job
- Single script for all steps
- Build triggers & parameters like regular projects
- Each workflow execution is a regular Jenkins build displayed in regular Jenkins views
- Graphical visualization of actual build possible
  - Not of visualization job definition but of build execution



# Non-sequential logic

```
parallel(qualityAnalysis: {  
    // RUN SONAR ANALYSIS  
    // ...  
, performanceTest: {  
    // DEPLOY ON PERFS AND RUN JMETER STRESS TEST  
    // ...  
})
```

```
try {} catch {}
```

```
for (def hostname: hostnames) {  
    sh "ssh $hostname -c /opt/tomcat/bin/shutdown.sh"  
}
```

```
retry(count: 5) {  
    sh "sleep 5 && curl http://localhost:$httpPort/health-check.jsp"  
}
```



# Interaction with humans

```
42  input message: "Does staging app http://localhost:$qaHttpPort/ look good? ",  
43          ok: "DEPLOY TO STAGING!", submitter: "petclinic-qa-group"
```

A screenshot of a Jenkins build interface. The top navigation bar shows 'Jenkins > petclinic > #53 > Paused for Input'. On the right, there is a green button labeled 'DEPLOY TO STAGING!' with an 'Abort' button next to it. A large green arrow points from the bottom left towards this button. On the left, there is a sidebar with links: Back to Project, Status, Changes, Console Output, Edit Build Information, Git Build Data, No Tags, Test Result, Checkpoints, Paused for Input (which is highlighted in green), and Previous Build. At the bottom left, a message box says: 'Running: Input Does staging app <http://localhost:8081/> look good? If yes, we deploy [DEPLOY TO STAGING!](#) or [Abort](#)'.

# Restartable build / checkpoint



```
36  
37  checkpoint 'CHOOSE TO ENTER STAGING'  
38  
39  input message: "Does staging app http://localhost:$qaHttpPort/ look good? " +  
40          "If yes, we deploy on staging.", ok: "DEPLOY TO STAGING!"
```

### Checkpoints

Checkpoints capture the state of a running workflow in such a way that it can be restarted as a separate run.

Name	Actions
CHOOSE TO ENTER STAGING	

# Visualization



## Workflow petclinic

Last Successful Artifacts

Recent Changes

### Pipeline Stage View

	DEV	QUALITY_ANALYSIS	PERFS	QA	STAGING	
	19s (avg.)	2s (avg.)	43s (avg.)	1min 35s (avg.)	14s (avg.)	
#52 Oct 31 17:58			48ms	2min 8s	14s	
#51 Oct 31 17:56	20s	36ms	45s	33s	14s	
#50 Oct 31 17:48	26s	45ms	48s	51s	14s	
#49 Oct 31 15:37	16s	28ms	43s	2min 49s	14s	
#48 Oct 31 15:04	31s	13s	1min 20s failed			

This Workflow run can be restarted from the following Checkpoint(s):

Delete

Restart

ENTER QA

Delete

Restart

CHOOSE TO ENTER STAGING

Produced the following 2 artifact(s):

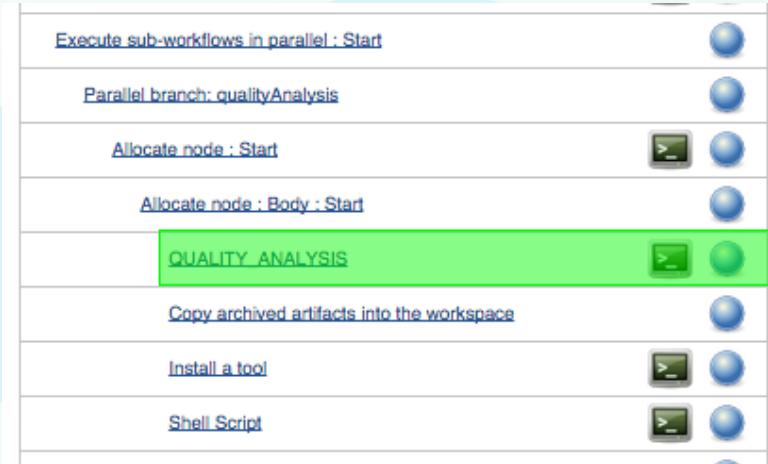
- src.tar (417.50KB)
- petclinic.war (28.04MB)

# Stages

```
stage name: 'QUALITY_ANALYSIS', concurrency: 1  
unarchive mapping: ['src/': '.', 'pom.xml': '.']  
sh 'mvn -o sonar:sonar'
```



- Semaphore
- Visualization



# Comprehensible scripts



```
83  /**
84  * @param war      The path to the war file to deploy
85  * @param catalinaBase The base directory of the Catalina server
86  * @param httpPort The port number to bind the application to
87  */
88  def deployApp(war, catalinaBase, httpPort) {
89      sh "${catalinaBase}/bin/shutdown.sh || :" // use "|| :" to ignore exception if server is not started
90      sh "rm -rf ${catalinaBase}/webapps/ROOT"
91      sh "rm -rf ${catalinaBase}/webapps/ROOT.war"
92      sh "cp -rf ${war} ${catalinaBase}/webapps/ROOT.war"
93      sh "${catalinaBase}/bin/startup.sh"
94      echo "$catalinaBase server restarted with new webapp $war, see http://localhost:$httpPort"
95      retry(count: 5) {
96          sh "sleep 5 && curl http://localhost:$httpPort/health-check.jsp"
97      }
98  }
99
100 }
101
102 /**
103 * @param catalinaBase The base directory of the Catalina server
104 */
105 def shutdownApp(catalinaBase) {
106     sh "${catalinaBase}/bin/shutdown.sh || :" // use "|| :" to ignore exception if server is not started
107     echo "$catalinaBase server is stopped"
108 }
```



# Demo time!

---





# Workflow Syntax Card



# Workflow Syntax Card

- ***stage***: Enter a new stage
- ***node***: Allocate node
- ***ws***: Allocate workspace
- ***parallel***: Execute sub-workflow in parallel
- ***retry***: Retry the body up to N times
- ***catchError***: Catch error and continue
- ***input***: Input / human interaction
- ***load***: Evaluate a Groovy source file into the workflow script



# Workflow Syntax Card

- ***step***: General build step
- ***sh***: Shell script
- ***bat***: Windows batch script



# Workflow Syntax Card

- ***archive***: Archive artifacts
- ***unarchive***: Copy archived artifact into the workspace
- ***echo***: Print message
- ***dir***: Change directory
- ***readFile***: Read file from workspace
- ***writeFile***: Write file to workspace



# Workflow Syntax Card

- ***git***: Git
- ***svn***: Subversion
- ***scm***: General SCM
- ***tool***: install a tool
- ***build***: Build a job

# Workflow Syntax Card



- ***checkpoint***: capture the state of the execution so that it can be restarted later



# Possible Future

---



# Possible Future: probably OSS

- More plugin interoperability
- Multi-branch project (& SCM API) integration
- More advanced SCM
- In-IDE editing support
- Concise syntax

# Possible Future: probably Jenkins Enterprise

- More visualizations
- Validated merge integration
- Deployment, incl. blue/green with rollback
- Freestyle and/or Build Flow import
- Templates integration

# Get the code

- <https://gist.github.com/cyrille-leclerc/796085e19d9cec4a71ef>
  - workflow.groovy

W-JAX 14- Pimp Your Continuous Delivery Pipeline with the New Jenkins Workflow Engine

workflow.groovy

Raw

```
1 def qaCatalinaBase = '/opt/apache-tomcat-8-qa'
2 def qaHttpPort = 8081
3
4 def stagingCatalinaBase = '/opt/apache-tomcat-8-staging'
5 def stagingHttpPort = 8082
6
7 def perfsCatalinaBase = '/opt/apache-tomcat-8-perfs'
8 def perfsHttpPort = 8084
9
10 def productionCatalinaBase = '/opt/apache-tomcat-8-production'
11 def productionHttpPort = 8083
12
13
```

