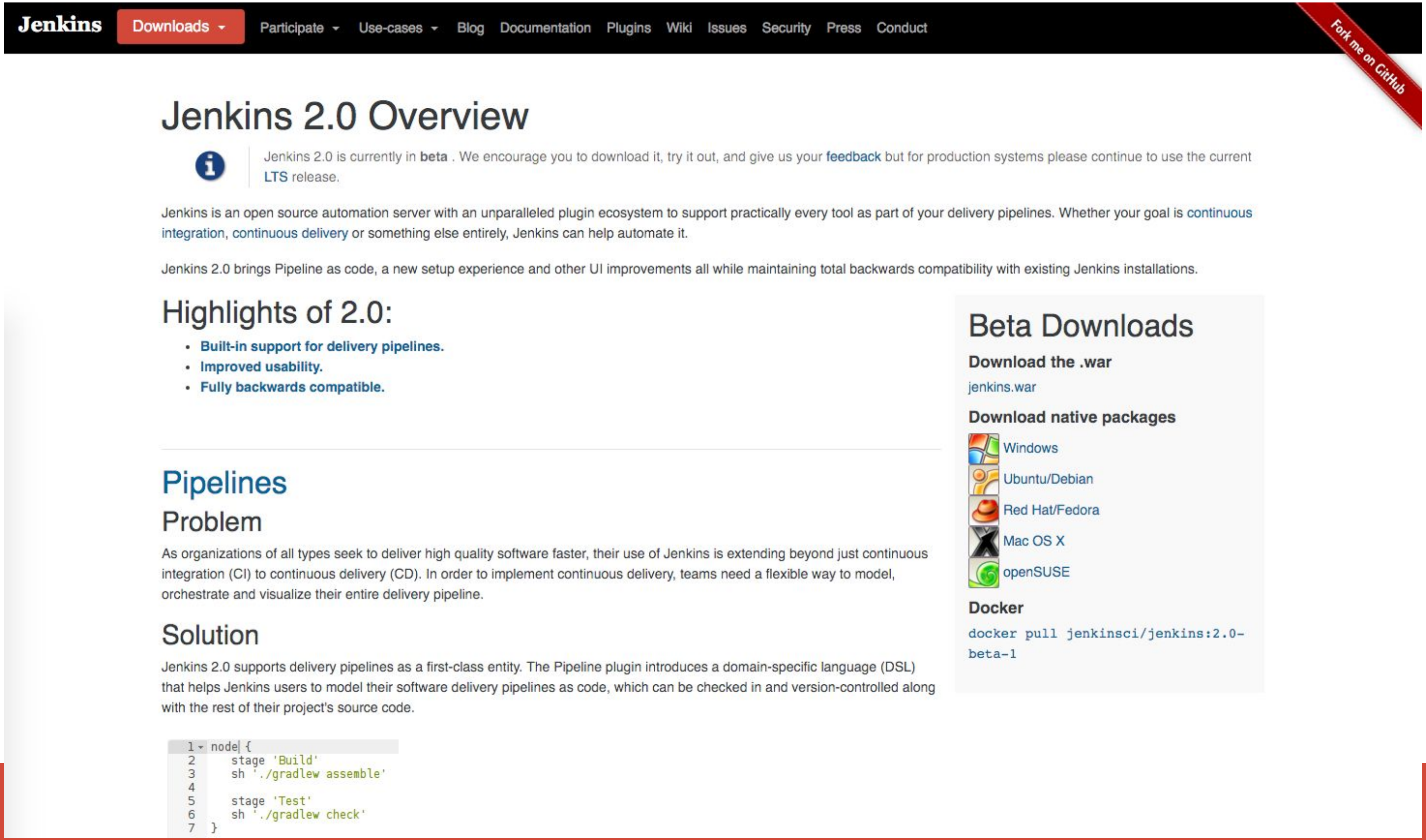
Jenkins2.0
Pipeline-as-code

Virtual Conference - May, 2016

Jenkins 2.0 - https://jenkins.io/2.0/



Jenkins 2.0 Overview

Jenkins 2.0 is currently in **beta** . We encourage you to download it, try it out, and give us your feedback but for production systems please continue to use the current LTS release.

Jenkins is an open source automation server with an unparalleled plugin ecosystem to support practically every tool as part of your delivery pipelines. Whether your goal is continuous integration, continuous delivery or something else entirely, Jenkins can help automate it.

Jenkins 2.0 brings Pipeline as code, a new setup experience and other UI improvements all while maintaining total backwards compatibility with existing Jenkins installations.

Highlights of 2.0:

- **Built-in support for delivery pipelines.**
- **Improved usability.**
- **Fully backwards compatible.**

Pipelines

Problem

As organizations of all types seek to deliver high quality software faster, their use of Jenkins is extending beyond just continuous integration (CI) to continuous delivery (CD). In order to implement continuous delivery, teams need a flexible way to model, orchestrate and visualize their entire delivery pipeline.

Solution

Jenkins 2.0 supports delivery pipelines as a first-class entity. The Pipeline plugin introduces a domain-specific language (DSL) that helps Jenkins users to model their software delivery pipelines as code, which can be checked in and version-controlled along with the rest of their project's source code.

```
node {
    stage 'Build'
    sh './gradlew assemble'

    stage 'Test'
    sh './gradlew check'
}
```

Beta Downloads

Download the .war

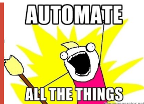
jenkins.war

Download native packages

- Windows
- Ubuntu/Debian
- Red Hat/Fedora
- Mac OS X
- openSUSE

Docker

```
docker pull jenkinsci/jenkins:2.0-beta-1
```

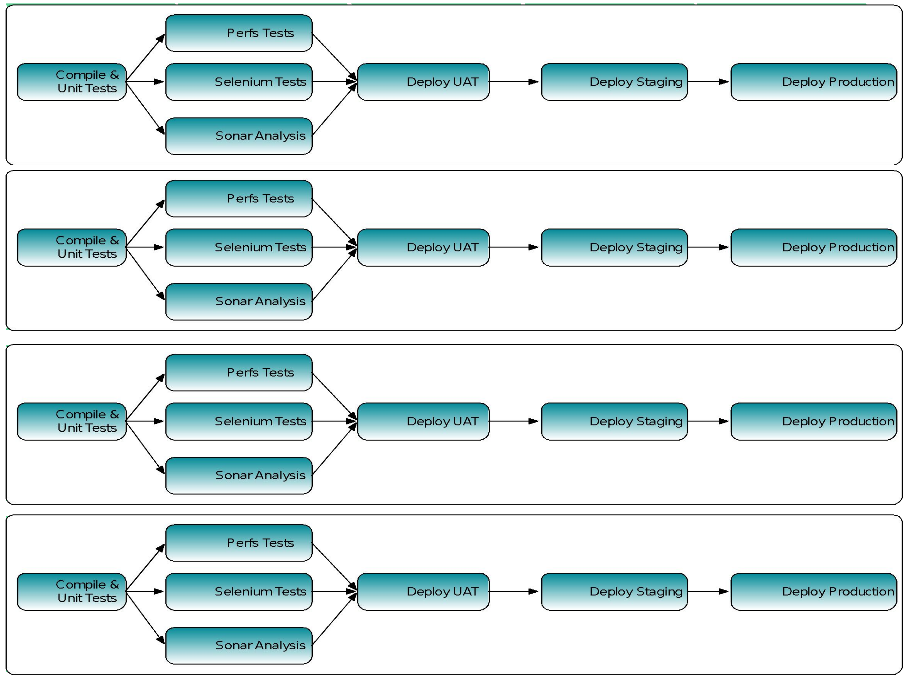


Pipeline Automates & Scales with Steps and Tools

2.0

Dev

Continuous Delivery



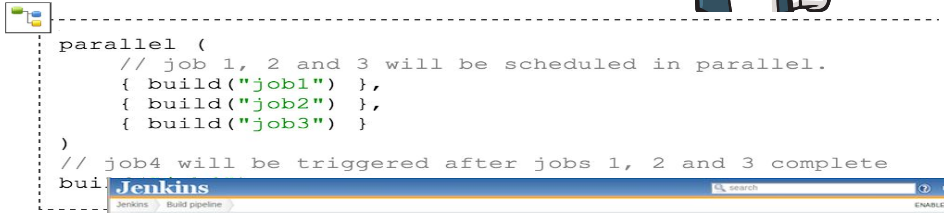
Prod

Needs

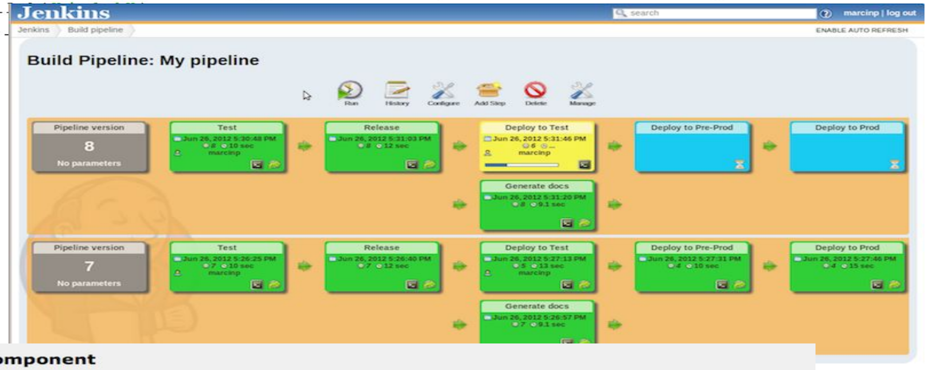
- Parallelism
- Branching
- Looping
- Restarts
- Checkpoints
- Manual Input

Build Pipelines before

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



```
parallel (
     // job 1, 2 and 3 will be scheduled in parallel.
     { build("job1") },
     { build("job2") },
     { build("job3") }
)
// job4 will be triggered after jobs 1, 2 and 3 complete
bui
```



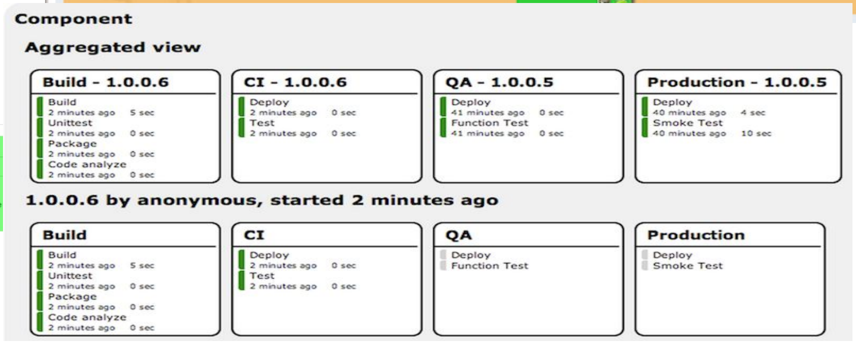
Jenkins   search   marcinp | log out

Jenkins › Build pipeline   ENABLE AUTO REFRESH

Build Pipeline: My pipeline

Pipeline version 8 — No parameters
Test — Jun 26, 2012 5:30:48 PM — #8 — 10 sec — marcinp
Release — Jun 26, 2012 5:31:03 PM — #8 — 12 sec
Deploy to Test — Jun 26, 2012 5:31:46 PM — #6 — ... — marcinp
Generate docs — Jun 26, 2012 5:31:20 PM — #8 — 9.1 sec
Deploy to Pre-Prod
Deploy to Prod

Pipeline version 7 — No parameters
Test — Jun 26, 2012 5:26:25 PM — #7 — 10 sec — marcinp
Release — Jun 26, 2012 5:26:40 PM — #7 — 12 sec
Deploy to Test — Jun 26, 2012 5:27:13 PM — #5 — 13 sec — marcinp
Generate docs — Jun 26, 2012 5:26:57 PM — #7 — 9.1 sec
Deploy to Pre-Prod — Jun 26, 2012 5:27:31 PM — #4 — 10 sec
Deploy to Prod — Jun 26, 2012 5:27:46 PM — #4 — 15 sec



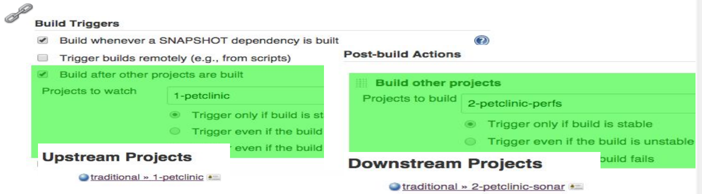
Component

Aggregated view

Build - 1.0.0.6	CI - 1.0.0.6	QA - 1.0.0.5	Production - 1.0.0.5
Build 2 minutes ago 5 sec Unittest 2 minutes ago 0 sec Package 2 minutes ago 0 sec Code analyze 2 minutes ago 0 sec	Deploy 2 minutes ago 0 sec Test 2 minutes ago 0 sec	Deploy 41 minutes ago 0 sec Function Test 41 minutes ago 0 sec	Deploy 40 minutes ago 4 sec Smoke Test 40 minutes ago 10 sec

1.0.0.6 by anonymous, started 2 minutes ago

Build	CI	QA	Production
Build 2 minutes ago 5 sec Unittest 2 minutes ago 0 sec Package 2 minutes ago 0 sec Code analyze 2 minutes ago 0 sec	Deploy 2 minutes ago 0 sec Test 2 minutes ago 0 sec	Deploy Function Test	Deploy Smoke Test



Build Triggers

- [✓] Build whenever a SNAPSHOT dependency is built
- [ ] Trigger builds remotely (e.g., from scripts)
- [✓] Build after other projects are built

Projects to watch: 1-petclinic

● Trigger only if build is st
○ Trigger even if the build
even if the build

Upstream Projects

traditional » 1-petclinic

Post-build Actions

Build other projects

Projects to build: 2-petclinic-perfs

● Trigger only if build is stable
○ Trigger even if the build is unstable
build fails

Downstream Projects

traditional » 2-petclinic-sonar

Pipeline Today...

Is defined in ONE concise script

Is Resilient - survives Master restarts

Uses *Stages* to add control and context

Is Visualized – StageView provides status at a glance dashboard and trending

Supports slave elasticity

- As many as you want, when you want

Is Pausable - Supports live interaction

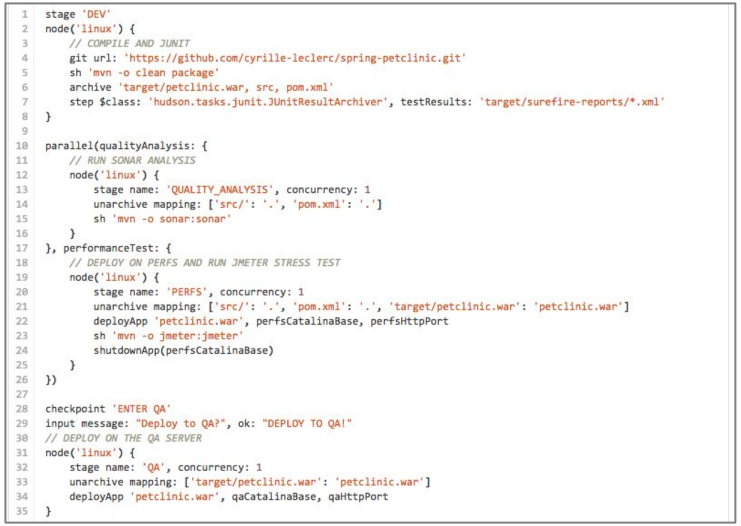
- pause and wait for human input/approval

Is Efficient- Restartable from checkpoints

Extensibility – the Jenkins way

- SCM, artifacts, plugins

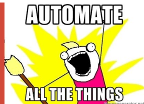
Delivers on “process as code”



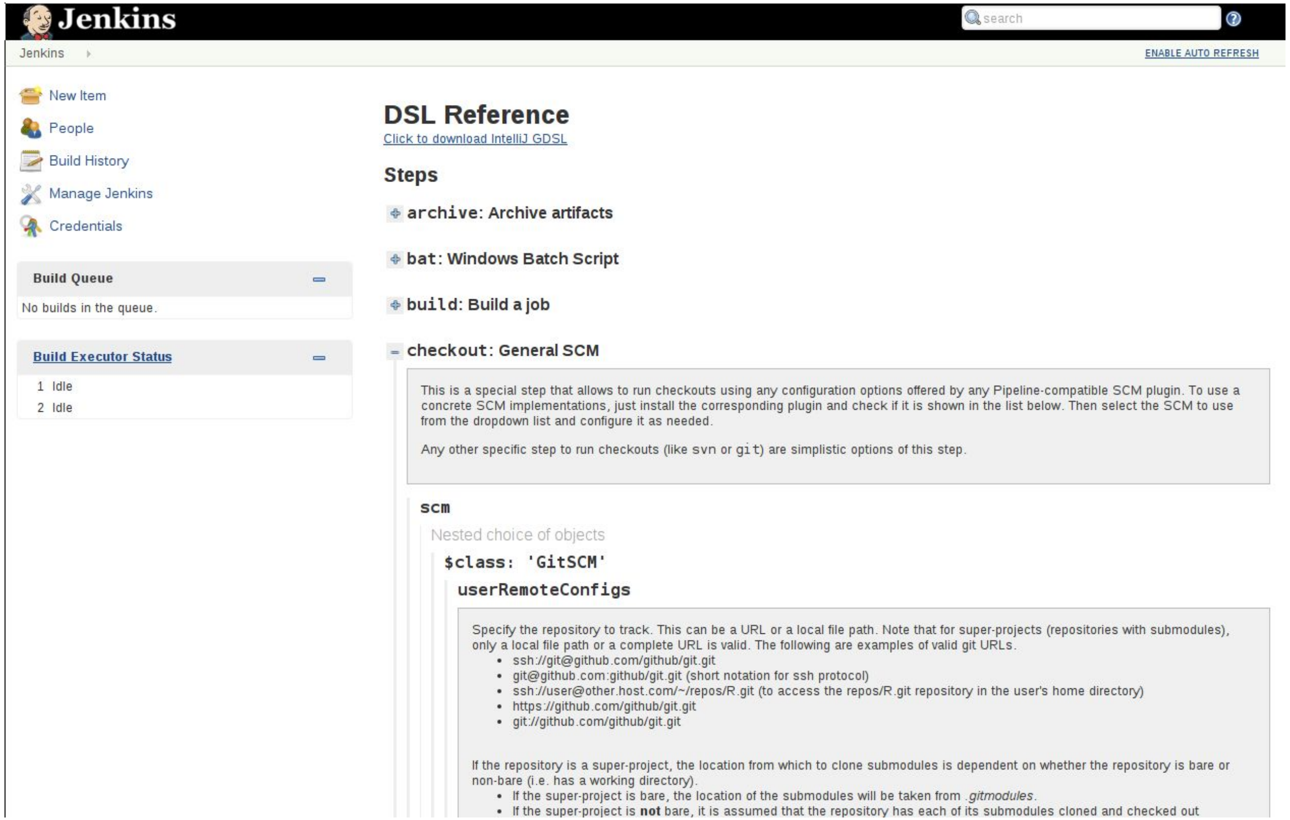
```
stage 'DEV'
node('linux') {
    // COMPILE AND JUNIT
    git url: 'https://github.com/cyrille-leclerc/spring-petclinic.git'
    sh 'mvn -o clean package'
    archive 'target/petclinic.war, src, pom.xml'
    step $class: 'hudson.tasks.junit.JUnitResultArchiver', testResults: 'target/surefire-reports/*.xml'
}

parallel(qualityAnalysis: {
    // RUN SONAR ANALYSIS
    node('linux') {
        stage name: 'QUALITY_ANALYSIS', concurrency: 1
        unarchive mapping: ['src/': '.', 'pom.xml': '.']
        sh 'mvn -o sonar:sonar'
    }
}, performanceTest: {
    // DEPLOY ON PERFS AND RUN JMETER STRESS TEST
    node('linux') {
        stage name: 'PERFS', concurrency: 1
        unarchive mapping: ['src/': '.', 'pom.xml': '.', 'target/petclinic.war': 'petclinic.war']
        deployApp 'petclinic.war', perfsCatalinaBase, perfsHttpPort
        sh 'mvn -o jmeter:jmeter'
        shutdownApp(perfsCatalinaBase)
    }
})

checkpoint 'ENTER QA'
input message: "Deploy to QA?", ok: "DEPLOY TO QA!"
// DEPLOY ON THE QA SERVER
node('linux') {
    stage name: 'QA', concurrency: 1
    unarchive mapping: ['target/petclinic.war': 'petclinic.war']
    deployApp 'petclinic.war', qaCatalinaBase, qaHttpPort
}
```

Pipeline DSL

```
node('docker') {
    checkout scm

    /* Grab the abbreviated SHA1 of our pipeline’s commit.*/
    sh 'git rev-parse HEAD > GIT_COMMIT'
    def shortCommit = readFile('GIT_COMMIT').take(6)

    stage 'Build'
    def image = docker.build("jenkinsciinfra/bind:build-${shortCommit}")

    stage 'Deploy'
    image.push()
}
```



Pipeline DSL Reference



Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

DSL Reference

Click to download IntelliJ GDSL

Steps

archive: Archive artifacts

bat: Windows Batch Script

build: Build a job

checkout: General SCM

This is a special step that allows to run checkouts using any configuration options offered by any Pipeline-compatible SCM plugin. To use a concrete SCM implementations, just install the corresponding plugin and check if it is shown in the list below. Then select the SCM to use from the dropdown list and configure it as needed.

Any other specific step to run checkouts (like svn or git) are simplistic options of this step.

scm

Nested choice of objects

$class: 'GitSCM'

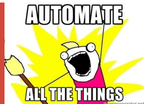
userRemoteConfigs

Specify the repository to track. This can be a URL or a local file path. Note that for super-projects (repositories with submodules), only a local file path or a complete URL is valid. The following are examples of valid git URLs.

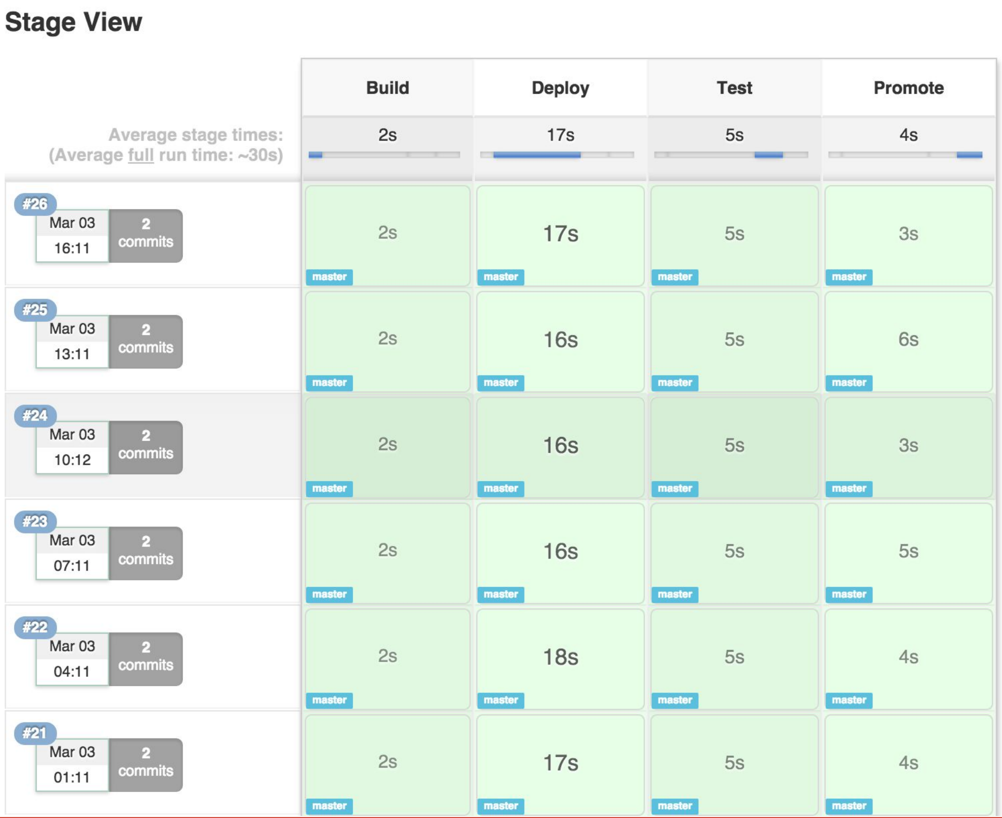
- ssh://git@github.com/github/git.git
- git@github.com:github/git.git (short notation for ssh protocol)
- ssh://user@other.host.com/~/repos/R.git (to access the repos/R.git repository in the user's home directory)
- https://github.com/github/git.git
- git://github.com/github/git.git

If the repository is a super-project, the location from which to clone submodules is dependent on whether the repository is bare or non-bare (i.e. has a working directory).

- If the super-project is bare, the location of the submodules will be taken from *.gitmodules*.
- If the super-project is **not** bare, it is assumed that the repository has each of its submodules cloned and checked out



Pipeline Stage View



Stage View

	Build	Deploy	Test	Promote
Average stage times: (Average full run time: ~30s)	2s	17s	5s	4s
#26 Mar 03 16:11 2 commits	2s master	17s master	5s master	3s master
#25 Mar 03 13:11 2 commits	2s master	16s master	5s master	6s master
#24 Mar 03 10:12 2 commits	2s master	16s master	5s master	3s master
#23 Mar 03 07:11 2 commits	2s master	16s master	5s master	5s master
#22 Mar 03 04:11 2 commits	2s master	18s master	5s master	4s master
#21 Mar 03 01:11 2 commits	2s master	17s master	5s master	4s master

2.0

AUTOMATE ALL THE THINGS

CD Pipeline-as-code?

Overall job definition is a script

- calls your build tools and scripts for details

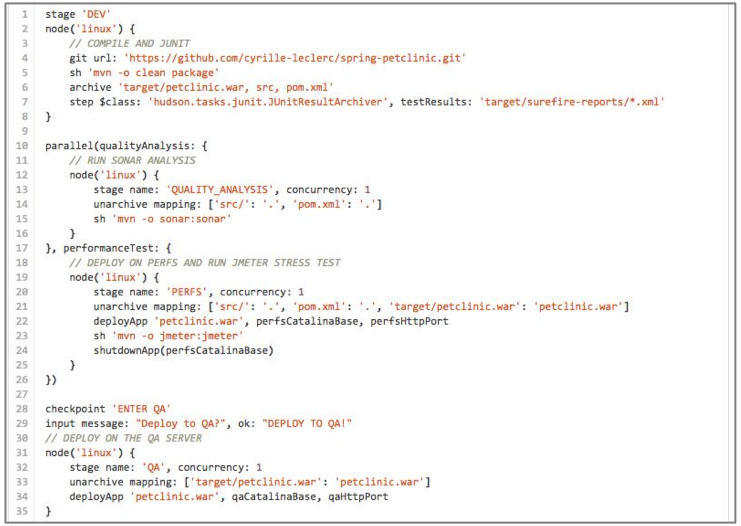
Script can be versioned alongside project sources

- experimental branches

- code review!

Keep less configuration in $JENKINS_HOME

Pipeline Global libs (DRY)

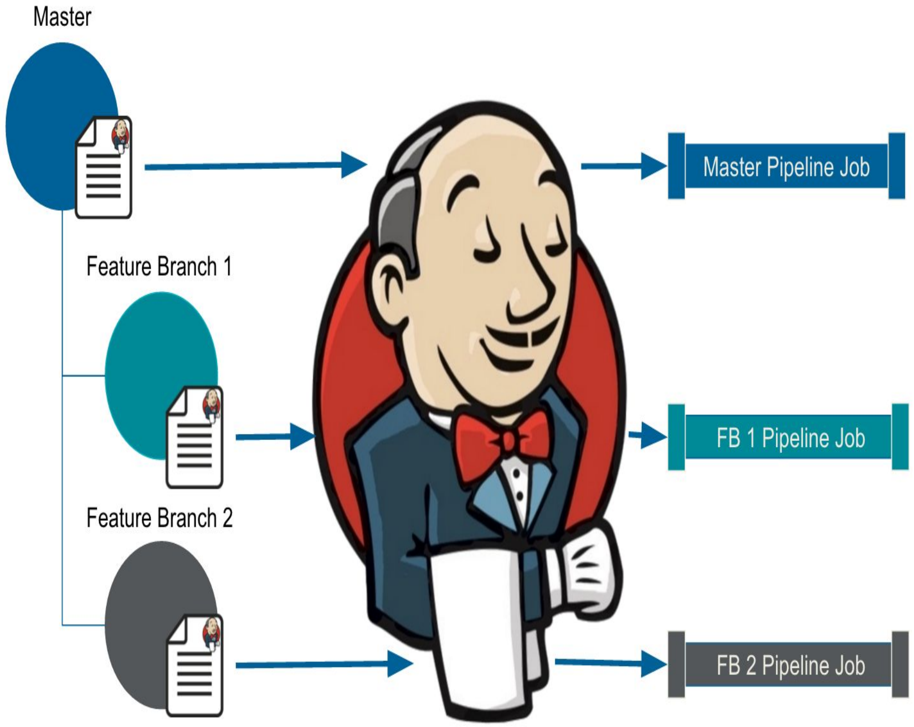


```
stage 'DEV'
node('linux') {
    // COMPILE AND JUNIT
    git url: 'https://github.com/cyrille-leclerc/spring-petclinic.git'
    sh 'mvn -o clean package'
    archive 'target/petclinic.war, src, pom.xml'
    step $class: 'hudson.tasks.junit.JUnitResultArchiver', testResults: 'target/surefire-reports/*.xml'
}

parallel(qualityAnalysis: {
    // RUN SONAR ANALYSIS
    node('linux') {
        stage name: 'QUALITY_ANALYSIS', concurrency: 1
        unarchive mapping: ['src/': '.', 'pom.xml': '.']
        sh 'mvn -o sonar:sonar'
    }
}, performanceTest: {
    // DEPLOY ON PERFS AND RUN JMETER STRESS TEST
    node('linux') {
        stage name: 'PERFS', concurrency: 1
        unarchive mapping: ['src/': '.', 'pom.xml': '.', 'target/petclinic.war': 'petclinic.war']
        deployApp 'petclinic.war', perfsCatalinaBase, perfsHttpPort
        sh 'mvn -o jmeter:jmeter'
        shutdownApp(perfsCatalinaBase)
    }
})

checkpoint 'ENTER QA'
input message: "Deploy to QA?", ok: "DEPLOY TO QA!"
// DEPLOY ON THE QA SERVER
node('linux') {
    stage name: 'QA', concurrency: 1
    unarchive mapping: ['target/petclinic.war': 'petclinic.war']
    deployApp 'petclinic.war', qaCatalinaBase, qaHttpPort
}
```

Pipeline-as-code – MultiBranch Pipeline

2.0

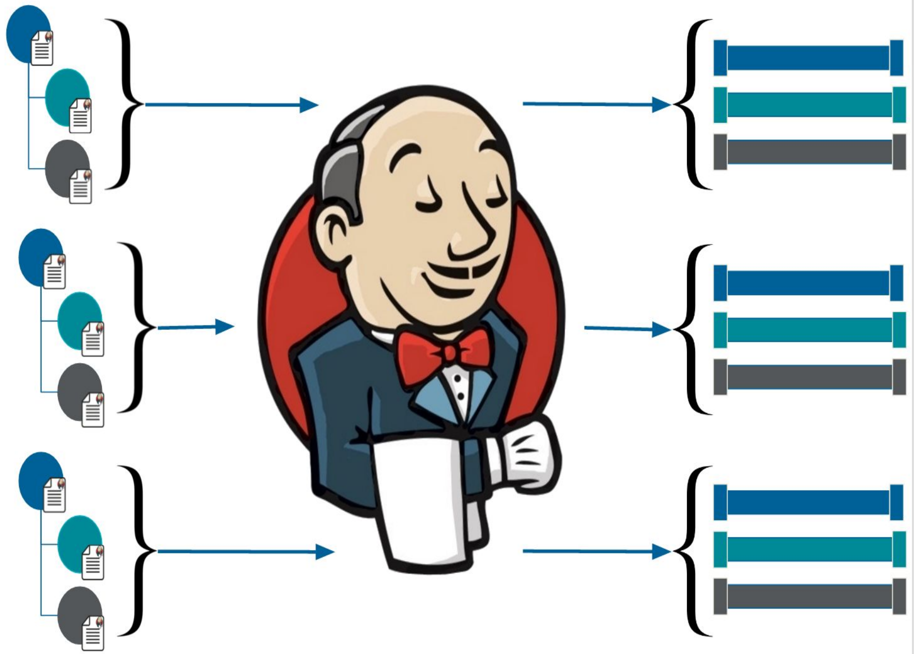
- Branch with a Jenkinsfile → one per subproject
  - that is your Pipeline script
  - just checkout scm to get full source tree
- Can edit Jenkinsfile (Pipeline) in your branch
  - revision matches sources
- Git, SVN, Mercurial
- Dedicated GitHub support
  - GitHub API
  - Webhooks
  - PullRequest



Pipeline-as-code – Organization Folders

2.0

- Before: custom scripting just to add all 100 repos
- New folder type: "organization"
  - each item is a multibranch Pipeline project
  - adds/removes projects automatically
- Only configuration is org name + credentials
  - one step closer to "code as config"



Pipeline-as-code: Demo

Resources

- Jenkins Pipeline reference
  - https://jenkins.io/doc/pipeline/
- Official Docker image
  - `$ docker pull jenkinsci/pipeline-as-code-github-demo`
  - https://hub.docker.com/r/jenkinsci/pipeline-as-code-github-demo/