

DECLARATIVE

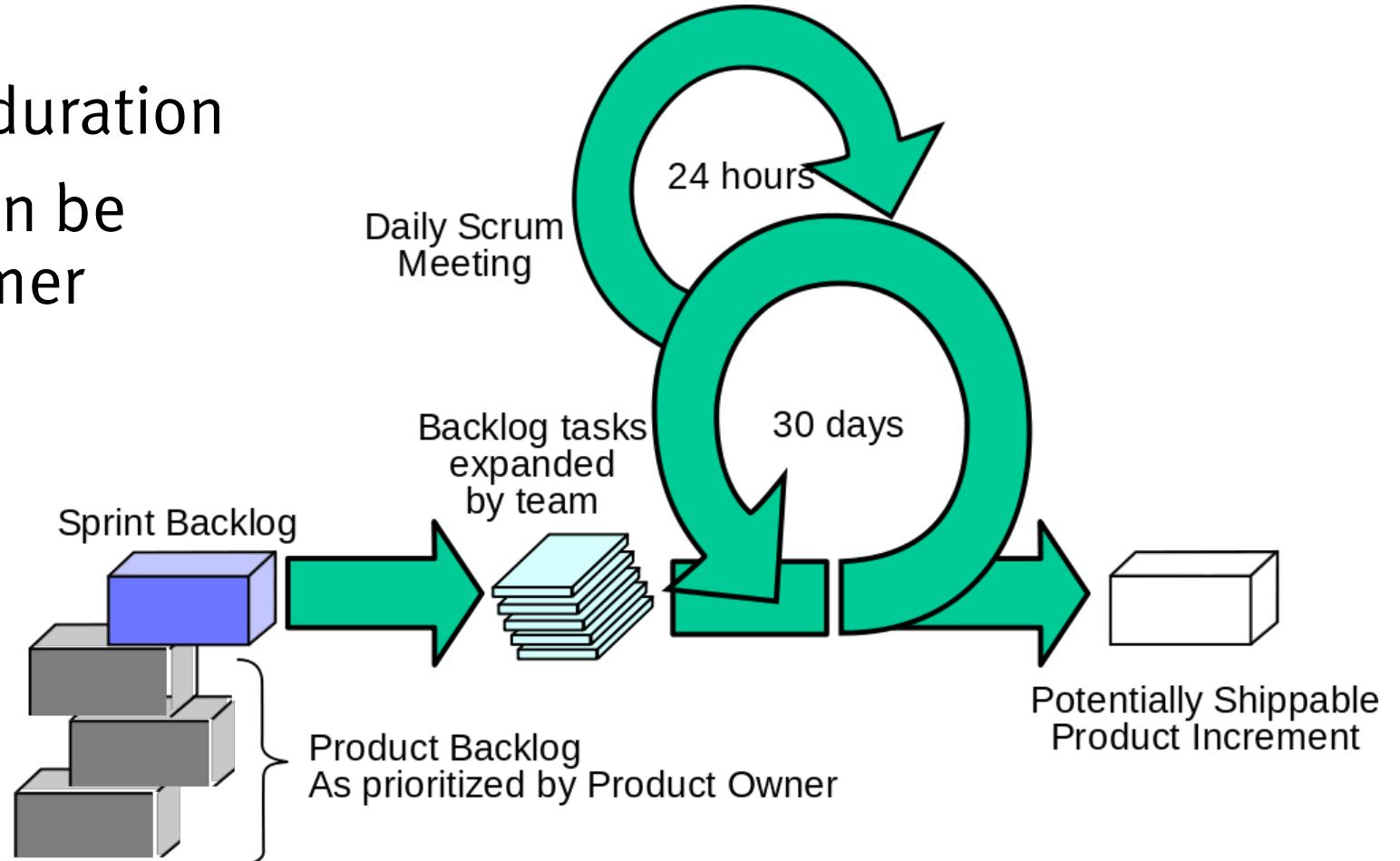
Jenkins Pipelines



Continuous Delivery

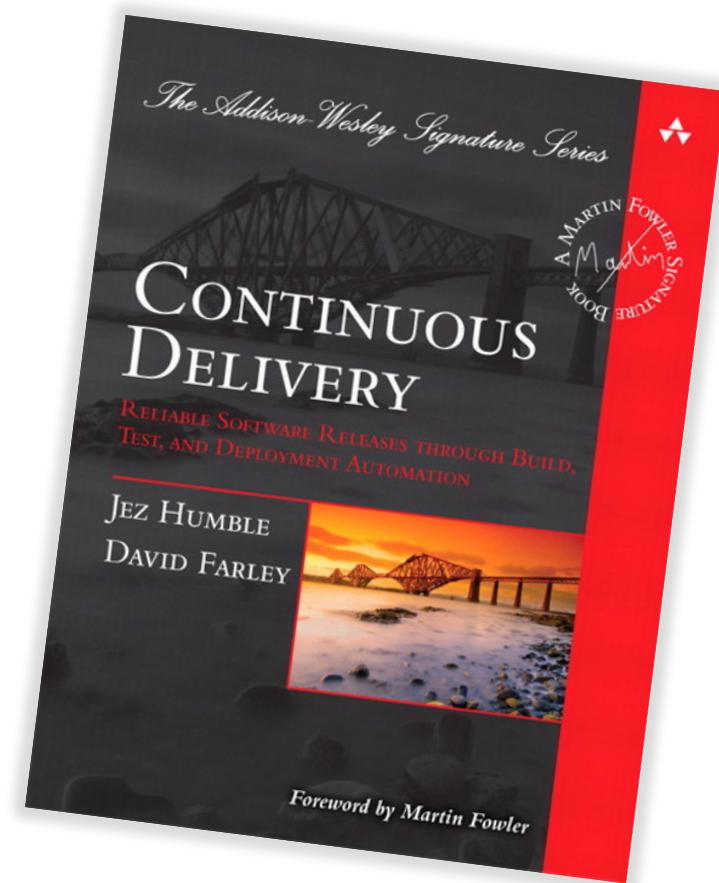
Agile Development

- Sprints of ~1-4 weeks duration
- Result: Product that can be shown/given to customer
- Return: Feedback



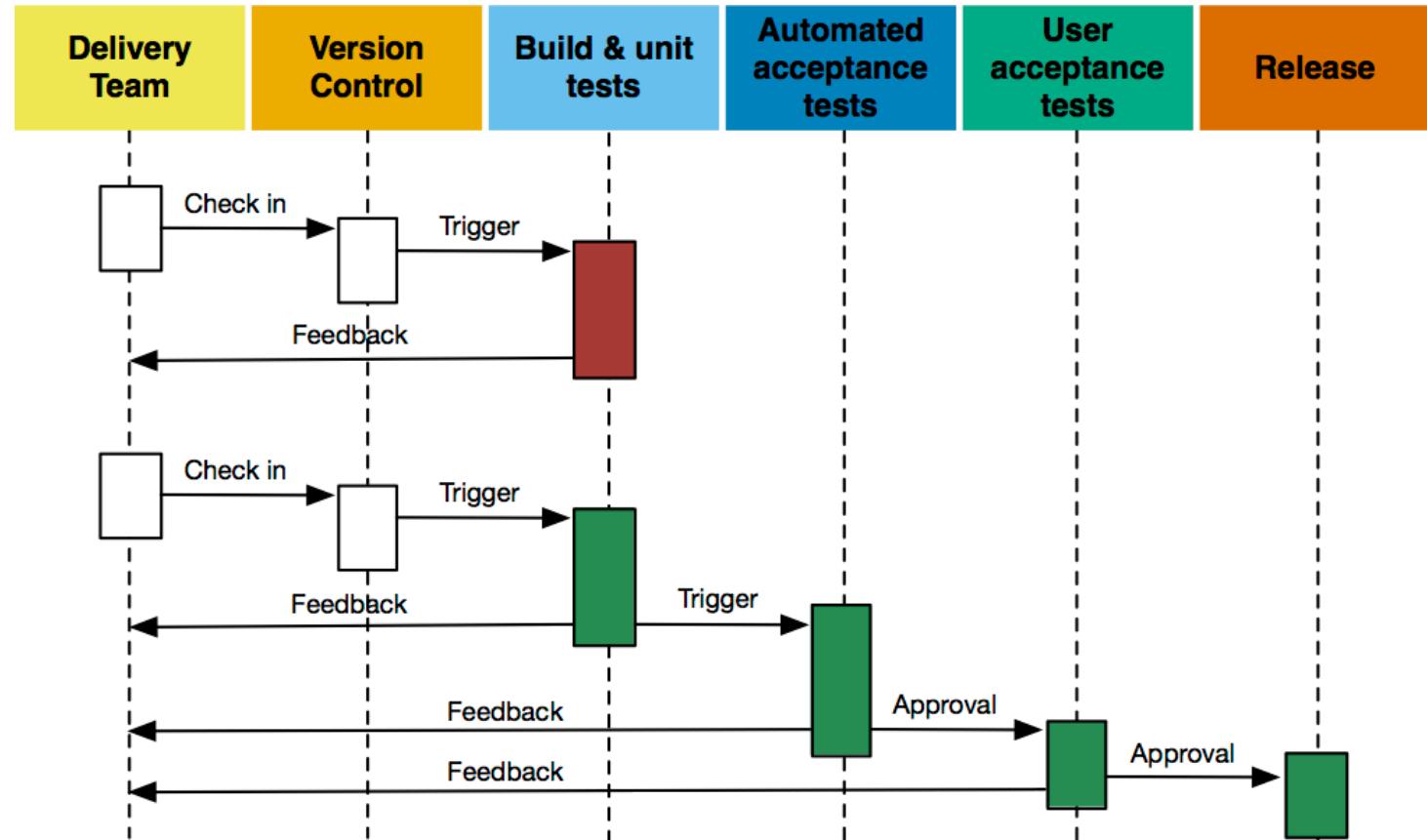
Continuous Delivery

- Reduce cycle times (even more)
 - “*How long does it take to release a one-line change?*”
- Potentially deliverable code with every commit
 - Automated tests decide about acceptance
 - You don’t *have to* release/deploy it, but you *could*
 - Continuous Deployment: Deploy every successfully tested commit automatically



Pipelines

- Every check-in triggers pipeline execution
- Feedback to the team in every stage
 - “*Bring the pain forward*”
 - “*Fail fast, fail often*”
- Minimize execution time
- Always aware of latest *stable* release



CI & CD Tools

CI/CD Tools

- On-premise
 - *Jenkins*
 - Thoughtworks Go
 - Gitlab CI
 - Pivotal Concourse
 - Jetbrains TeamCity
- SaaS
 - TravisCI
 - CircleCI
 - AppVeyor
 - Codeship
 - Visual Studio Team Services



Why (I like) Jenkins



- Established open source project
- On premise installation
- Thousands of plugins
- Integrates with many tools/services
- Tailored to CI/CD

Jenkins

search log in

ENABLE AUTO REFRESH

People Build History Job Priorities

Build Queue

No builds in the queue.

All

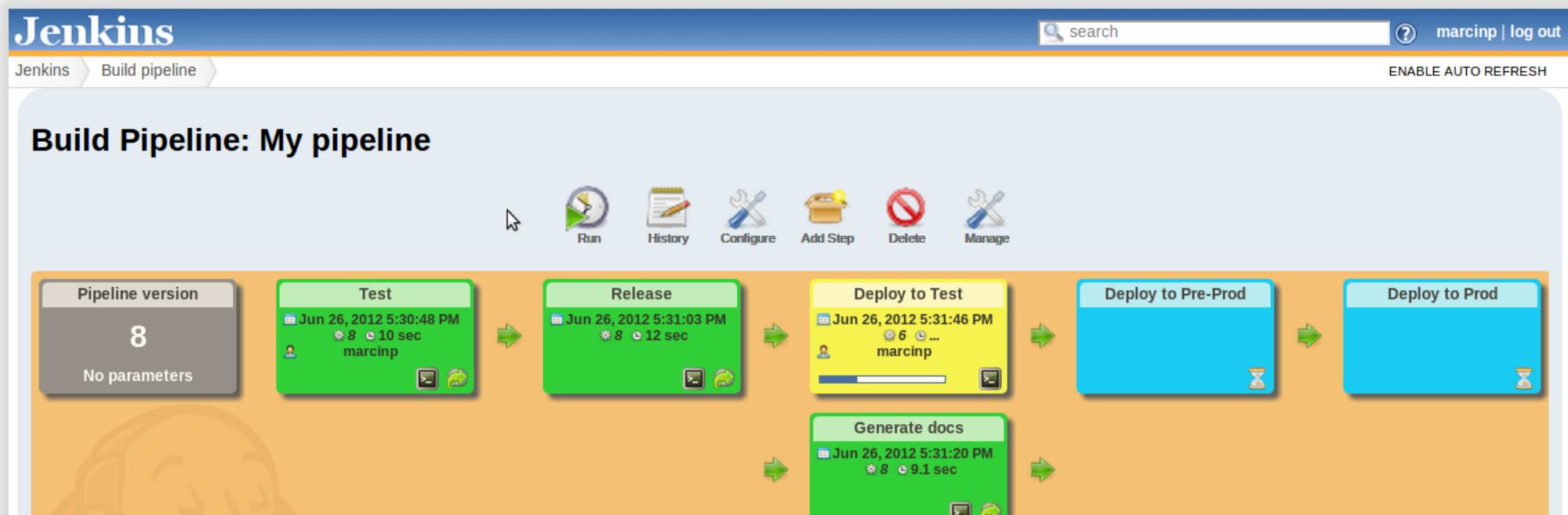
S	W	Name ↓	Last Success	Last Failure	Last Duration
Green	Sunny	Main Chef repo	3 days 23 hr - #42	3 mo 18 days - #14	1 sec
Green	Sunny	Seed: Chef Main Repo	1 mo 28 days - #2	N/A	1.3 sec
Orange	Cloudy	TYPO3's Chef Cookbooks	N/A	N/A	N/A

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

History of CI/CD with Jenkins

- Downstream Jobs
 - Job A triggers job B triggers job C triggers job D trig...
 - If one job fails, fail all
- *Build Pipeline View*



... and what I want instead

But that's awkward

Configuration as Code

- Define jobs/pipelines as code
 - Avoid point & click
 - **In version control**
 - Can live with the application
 - **Scales better**
- Example: `.travis.yml` (from TravisCI)
- Similar to GitlabCI

```
language: php

services:
- redis-server

before_script:
- composer install

script:
- ./bin/phpunit -c ...

notifications:
slack:
...
```

Code-driven Approaches in Jenkins

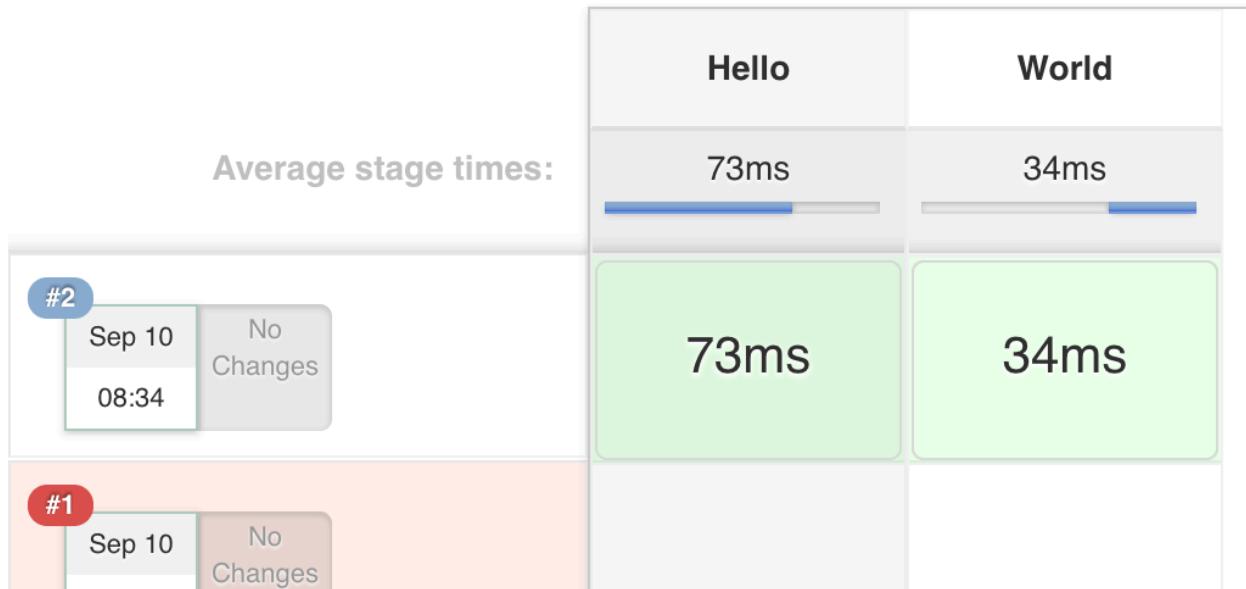
- Jenkins Job Builder
 - Python / YAML based, from the OpenStack project
- Job DSL plugin
 - Groovy DSL!
(e.g. query Github API and create jobs for all branches)
 - Great thing!
 - But creates single jobs

```
job('my-project-main') {  
    scm {  
        git('https://github.com/...')  
    }  
    triggers {  
        scm('H/15 * * * *')  
    }  
    publishers {  
        downstream('my-project-unit')  
    }  
}
```

Jenkins Pipeline Plugins

Jenkins Pipeline Plugins

- Whole suite of plugins (20+), open-sourced a year ago
- Shipped with Jenkins 2.0
- Formerly commercially available by CloudBees, called *Workflow*
- Define pipeline as code (again Groovy DSL)



```
stage("Hello") {  
    ...  
}  
stage("World") {  
    ...  
}
```

Pipeline Example



Console Output

```
Started by user anonymous
[Pipeline] node
Running on master in /var/lib/jenkins/jobs/test/workspace
[Pipeline] {
  [Pipeline] stage
  [Pipeline] { (Build)
  [Pipeline] echo
Starting engines
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Unit)
[Pipeline] sh
[workspace] Running shell script
+ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/dm-0        38G  2.3G   34G   7% /
udev            202M  202M    0B  100% /run/user/110
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

```
pipeline {
  stages {
    stage("Build") {
      steps {
        echo "Starting engines"
      }
    }
    stage("Unit") {
      steps {
        sh "df -h"
      }
    }
  }
  agent any
}
```

Pipeline DSL Steps

- Shellout

- *nix systems: sh

```
sh('make'), sh('rm -rf /')
```

- Windows systems: bat

```
bat('C:\Program Files\..')
```

- SCM

```
git('https://github.com/..')
```

- File handling

```
readFile('my.config')
```

```
writeFile(file: 'README.md', text: 'Hello World')
```

```
fileExists('filename.txt')
```

Pipeline DSL Steps (2)

- Copy workspace to next agent (aka executor/node)

```
stage('build') {  
    agent 'build-tool'  
    steps {  
        step {  
            sh('make')  
            stash('my-workspace')  
        }  
    }  
}  
// continue - - >
```

```
stage('release') {  
    agent 'release-machine'  
    steps {  
        step {  
            unstash('my-workspace')  
            sh('release')  
        }  
    }  
}
```

Pipeline DSL Steps (3)

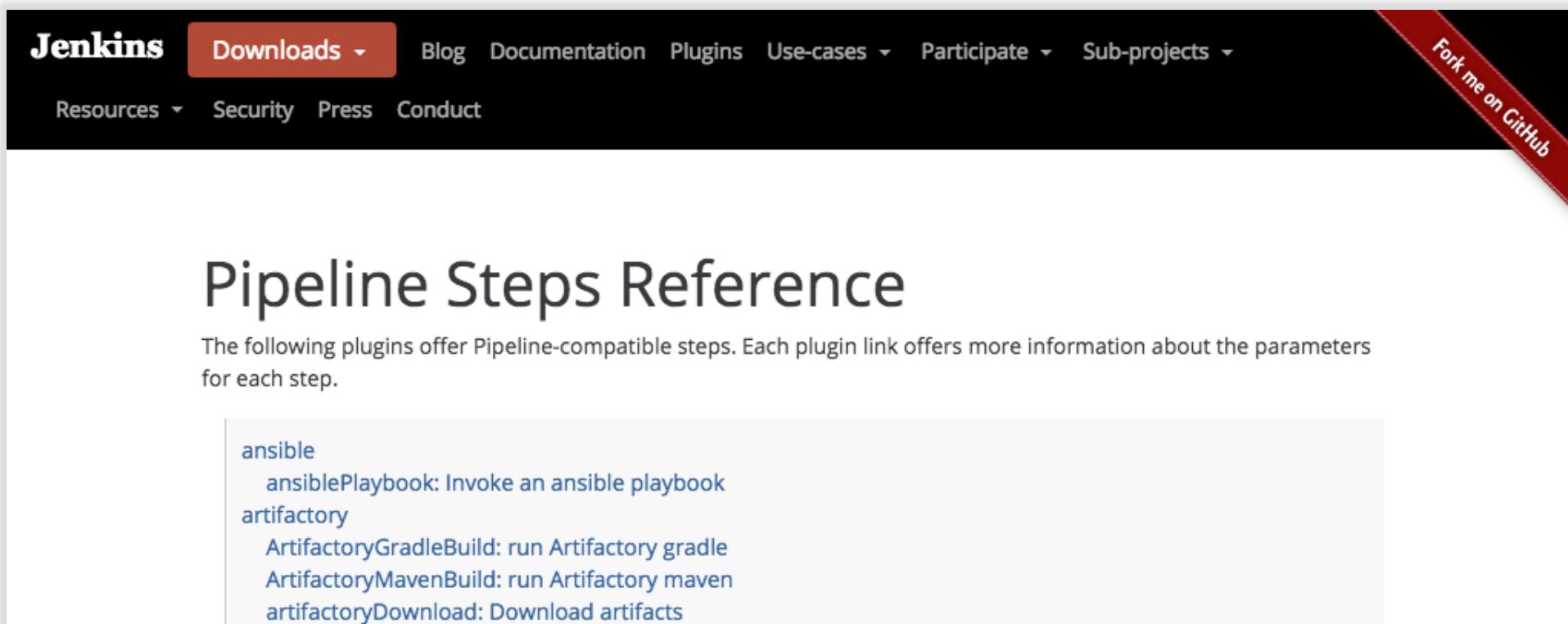
- Build another job:

```
build("jobname")
build("jobname", wait: false, propagate: false)
build(job: 'jobname', parameters: [
    [$class: 'StringParameterValue', name: 'target', value: target]
    [$class: 'ListSubversionTagsParameterValue', name: 'release', tag: release],
    [$class: 'BooleanParameterValue', name: 'update_composer', value:
        update_composer.to_boolean()]])
```

- Will not go further into detail ☺

Even More Pipeline Steps

- Plugins contribute additional steps
 - Steps available in this Jenkins instance via  Pipeline Syntax
 - Online reference: <https://jenkins.io/doc/pipeline/steps/>



The screenshot shows the Jenkins Pipeline Steps Reference page. The top navigation bar includes links for Jenkins, Downloads, Blog, Documentation, Plugins, Use-cases, Participate, Sub-projects, Resources, Security, Press, Conduct, and a GitHub fork button. The main content area features a large title "Pipeline Steps Reference". Below it, a paragraph states: "The following plugins offer Pipeline-compatible steps. Each plugin link offers more information about the parameters for each step." A list of plugins is provided, each with a link to its documentation:

- ansible
 - [ansiblePlaybook](#): Invoke an ansible playbook
- artifactory
 - [ArtifactoryGradleBuild](#): run Artifactory gradle
 - [ArtifactoryMavenBuild](#): run Artifactory maven
 - [artifactoryDownload](#): Download artifacts

User Input?

The screenshot shows the Jenkins Blue Ocean interface. At the top, there are tabs for Pipelines, Pipeline Editor, Administration, and a user icon. Below these are sub-tabs: Activity, Branches, and Pull Requests. A modal window titled "Input required" is open on the left, prompting for a target environment ("TESTING") and a number (radio buttons for 1, 2, or 3, with 1 selected). It also contains "Run" and "Cancel" buttons. To the right, the "Stage View" is displayed, showing a grid of stages across five columns: Git Checkout, Lint, Build, Acceptance, and Publish. The "Acceptance" column has a tooltip: "Bump major, minor or patch version? versionIncrement". A dropdown menu is open, showing options: patch, minor, and major. The "Patch" option is checked. The "Acceptance" stage for run #36 is currently "almost complete". The "Build" stage for run #36 is taking 7 seconds. The "Acceptance" stage for run #36 is taking 7 minutes and 34 seconds. The "Publish" stage for run #36 is taking 9 seconds. The "Acceptance" stage for run #35 is taking 2 seconds. The "Acceptance" stage for run #34 - 0.0.13 (patch) is taking 11 seconds. The "Acceptance" stage for run #33 is taking 6 seconds. The "Acceptance" stage for run #32 is taking 0ms.

Blue Ocean

Pipelines Pipeline Editor Administration

Parameetrized Activity Branches Pull Requests

Run Run

Status 2
✓
Run 1
✗

The target environment

TESTING

Pick a number

1 (selected)
2
3

Run Cancel

Input required

Average stage times:

Git Checkout	Lint	Build	Acceptance	Publish
2s	2s	7s	7min 34s	9s

Stage View

#36 Feb 05 2 commits 15:19 almost complete

#35 Feb 05 2 commits 15:02

#34 - 0.0.13 (patch) Feb 03 No Changes 13:58

#33 Feb 03 1 commits 13:37

#32

Bump major, minor or patch version?
versionIncrement

✓ patch
minor
major

Proceed Abort

2s 2s 2s 2s 2s

4s 11s 12min 40s 23s

(paused for 1min 42s)

(paused for 1min 39s)

0ms (paused for 0ms)

Parameters and Input

- Parametrized build (before build starts) ¹

```
pipeline {  
    // agent, steps. etc.  
    parameters {  
        string(name: 'DEPLOY_ENV', defaultValue: 'TESTING',  
              description: 'The target environment')  
    }  
}
```

- Input step (during pipeline execution) ²

```
input(message: 'Heyho', parameters: [string(..), ..])
```

bonus points for lock, milestone, timeout..

1) see <https://st-g.de/2016/12/parametrized-jenkins-pipelines>

2) see <http://stackoverflow.com/questions/42501553/jenkins-declarative-pipeline-how-to-read-choice-from-input-step>

Pipeline DSL

- Specify custom environment variables (and access credentials)

```
environment {  
    AWS_ACCESS_KEY_ID = credentials('my-aws-key')  
}  
  
sh "echo $AWS_ACCESS_KEY_ID"
```

- Variables provided by Jenkins

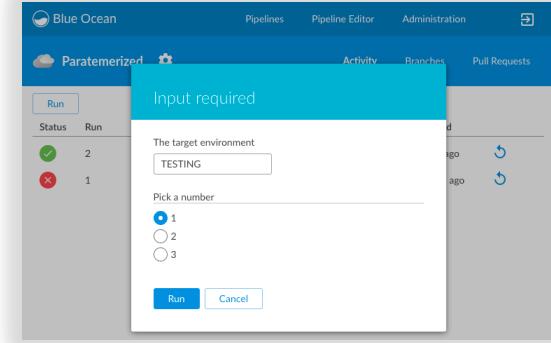
```
sh 'echo $BUILD_NUMBER'  
echo env.BUILD_NUMBER
```

- Control flow (a tiny bit..)

```
when { branch 'production' { sh 'rm -rf /' } }
```

SKIP

Full Example Parameterized Build



```
pipeline {  
    agent any  
  
    parameters {  
        string(name: 'DEPLOY_ENV', defaultValue: 'TESTING', description: 'Target environment')  
        choice(name: 'FRUIT', choices: 'apple\nbanana\npizza', description: 'Pick a fruit')  
    }  
  
    stages {  
        stage('Something') {  
            steps {  
                echo "Will deploy to ${params.DEPLOY_ENV}"  
                writeFile(file: 'fruit.txt', text: params.FRUIT)  
                echo readFile('fruit.txt')  
            }  
        }  
    }  
}
```

Docker

- Run build jobs within Docker containers
 - No need to install software on Jenkins master/slave
 - Use multiple versions of the same tool
- Containers can be existing ones or based on Dockerfile
- .. and Kubernetes

```
agent {  
    docker {  
        image 'maven:3-alpine'  
    }  
    stages { .. }  
}
```

Demo Time!



SKIP

```
pipeline {
    agent any
    tools {
        maven 'maven-3'
    }
    // or alternatively:
    // agent { docker 'maven:3-alpine' }
    stages {
        stage('Checkout') {
            steps {
                git "https://github.com/StephenKing/MAJUG17-jenkins-mvn"
                sh 'mvn clean'
            }
        }
        stage('Build') {
            steps {
                sh 'mvn package -DskipTests'
            }
        }
        stage('Tests') {
            steps {
                sh 'mvn test'
            }
            post {
                always {
                    junit 'target/surefire-reports/**/*xml'
                }
            }
        }
    }
    post {
        always {
            archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
        }
        success {
            emailext (
                subject: "SUCCESSFUL: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
                body: """SUCCESSFUL: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]': Check console output at ${env.BUILD_URL}""",
                to: 'mail@example.org'
            )
        }
        failure {
            emailext (
                subject: "FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
                body: """FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]': Check console output at ${env.BUILD_URL}""",
                to: 'mail@example.org'
            )
        }
    }
}
```

Declarative What?

- What you've seen now is the brand new stuff (pipeline-model-* v1.1)
- Formerly, there were "scripted pipelines"
 - Groovy-based DSL
 - Real code → real power
 - Sometimes: real pain*
- Both approaches officially supported

Scripted Pipeline Example

```
node {  
    stage("Foo") {  
        def data = new groovy.json.JsonSlurper().parseText(readFile('somefile.txt'))  
        sh "make ${data.options}"  
    }  
    stage("Bar") {  
        try {  
            sh "make"  
        } catch (err) {  
            slackSend message: "Oh dude, didn't workout. ${err}"  
            error "Things went wrong"  
        }  
    }  
    if (env.BRANCH_NAME == 'master') {  
        stage("Bar") {  
            echo "Deploy!!"  
        }  
    }  
}
```

Scripted Pipeline Execution

- node step allocates executor slot (“heavyweight executor”)
 - As other Jenkins jobs
 - Filter node by labels (i.e. `node('php7')`, `node('master')`)
 - Required for heavy lifting
 - Avoid many sequential allocations (slows down pipeline progress)
- Code outside of node
 - Flyweight executor
 - Running on master, “for free”
- Pipeline execution survives Jenkins restarts (CPS)

Pipeline Syntax: Declarative vs. Scripted

- Beginning

```
pipeline { .. }
```

```
// everything else
```

- Executor allocation

```
agent(docker:'image')  
steps { .. }
```

```
node('label') { sh .. } // or  
docker.image('..').inside{ .. }
```

- Post-build steps, failure handling

```
post { failure {} always {} }
```

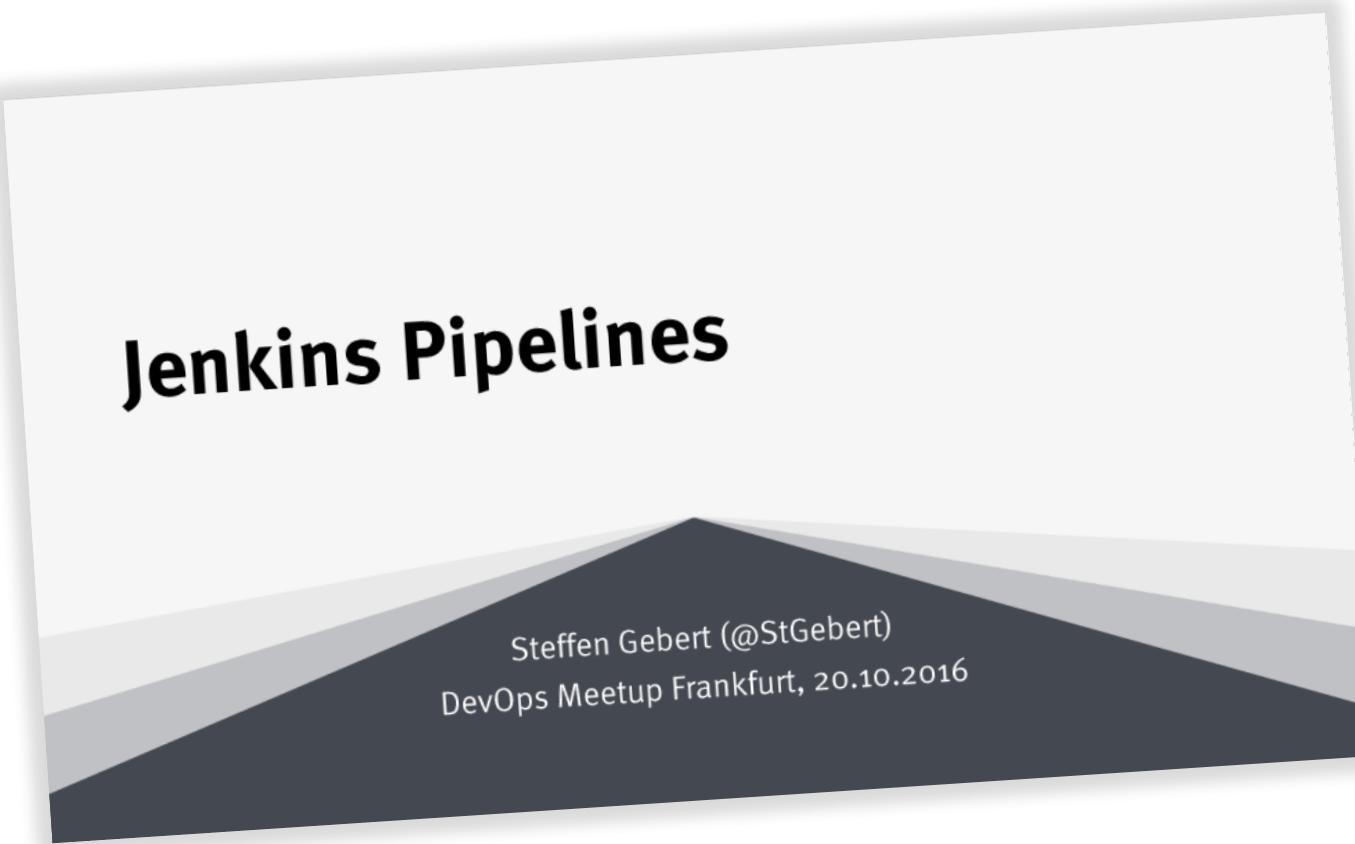
```
try { sh .. } catch { cry }
```

- Flow control

```
when { .. }  
script { // groovy script }
```

```
// any groovy statement
```

More on Scripted Pipelines

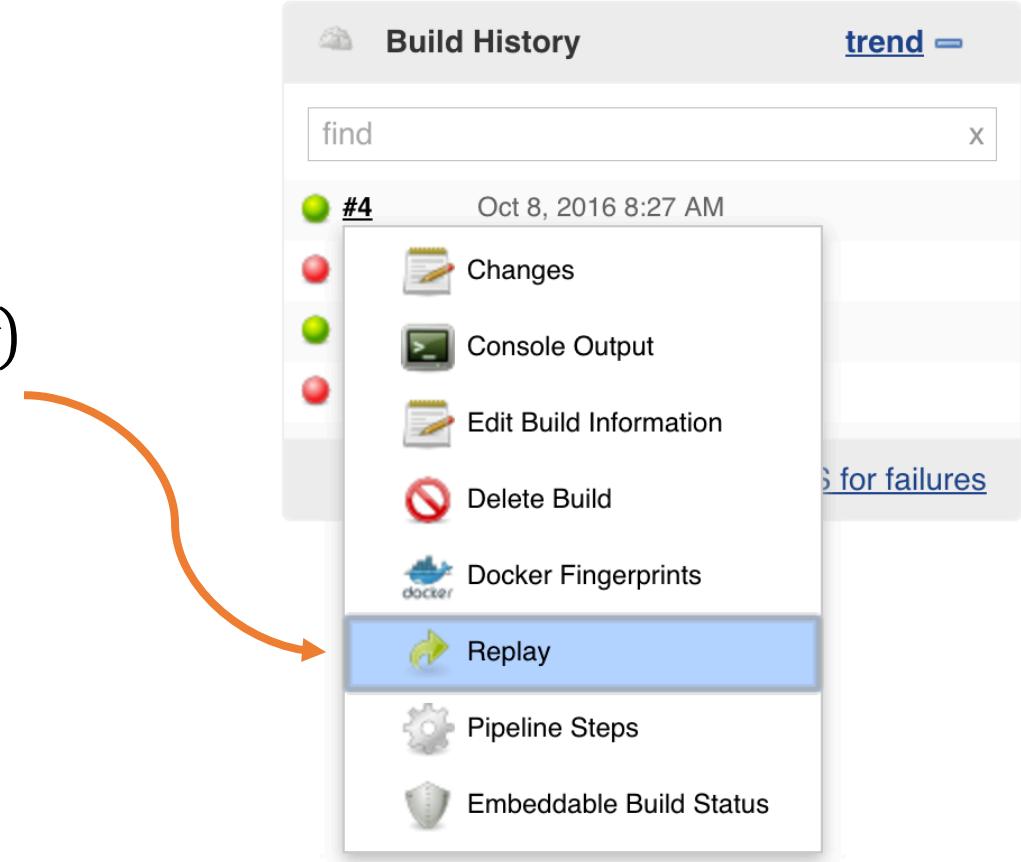


..in older versions of this talk
<https://st-g.de/speaking>

Where to put that config?

Pipeline Configuration

- Paste code into job config (fine for testing)
(job type: Pipeline)
- Create pipelines via JobDSL
- Commit it to your repo (job type: Multibranch)
 - File called `Jenkinsfile`
 - It evolves with the application
 - It is versioned
 - Everybody can read (and modify) it
 - You can throw away your Jenkins master at any time



Multibranch & Organization Folder Plugins

- Scans a complete GitHub/Bitbucket organization for Jenkinsfile
- Triggered by Webhook and/or runs periodically
- Automatically adds pipeline jobs per repo/branch/PR

The screenshot shows the Jenkins dashboard for the 'TYPO3's Chef Cookbooks' organization. The left sidebar includes links for Up, Status, Re-scan Organization, People, Build History, and GitHub. The main content area displays a list of repositories under the heading 'TYPO3's Chef Cookbooks'. The table lists five repositories:

S	W	Repository ↓	Description
■	⚡	backuppc	Chef cookbook for BackupPC
■	⚡	backuppc » develop	
■	⚡	gerrit	Chef cookbook for the Gerrit review system.
■	⚡	gerrit » develop	
■	⚡	gerrit » master	

DRY: Jenkins Global Library

- Provides shared functionality available for all jobs
- Loaded from remote Git repos

- Specified in global configuration or on folder-level

```
node { deployToProd() }
```

```
@Library("mylib@v1.2")
node { deployToProd() }
```

- Specified in Jenkinsfile

```
@Library("https://github.com/..")
node { deployToProd() }
```

Snipped Editor & Docs

- Because first steps are hard..
- For scripted and declarative pipelines
- Auto-generated DSL documentation
(Pipeline Syntax → Step Reference)

Steps

Sample Step `writeFile: Write file to workspace`

File path in workspace `README.txt`

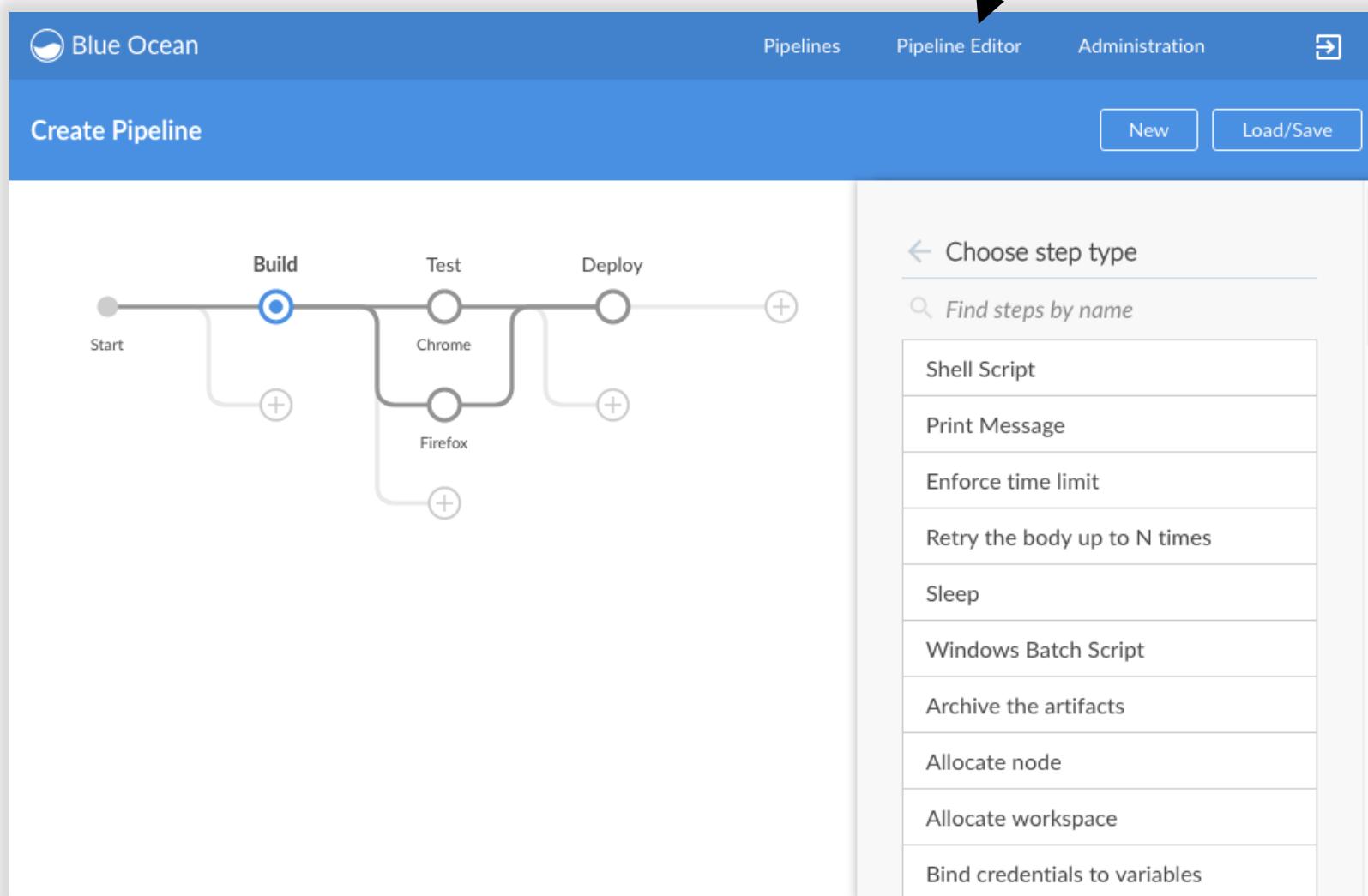
Text to write `Hello World`

Character encoding

Generate Groovy

```
writeFile file: 'README.txt', text: 'Hello World'
```

(Declarative) Pipeline Editor



Real-World Example

Jenkins Pipelines for Chef Cookbooks

Chef CI/CD at TYPO3.org

- Code that runs the *.typo3.org infrastructure, chef-ci.typo3.org
- Objective: Chef cookbooks
 - Server provisioning (installs packages, configures services)
 - Code: github.com/TYPO3-cookbooks



Many Cookbooks, Many Pipelines

- Scans our GitHub organization *TYPO3-cookbooks*
 - Automatically adds/removes pipelines for branches and pull requests*
 - Triggered via Webhooks
 - Contents of Jenkinsfile:

```
def pipe = new org.typo3.chefci.v1.Pipeline()  
pipe.execute()
```

* Currently suspicious to arbitrary code execution

Jenkins Global Library

- Pipelines implemented in Global Library
[TYPO3-infrastructure/jenkins-pipeline-global-library-chefci](#)

Branch: master ▾ [jenkins-pipeline-global-library-chefci](#) / [src](#) / [org](#) / [typo3](#) / [chefci](#) / [v1](#) /

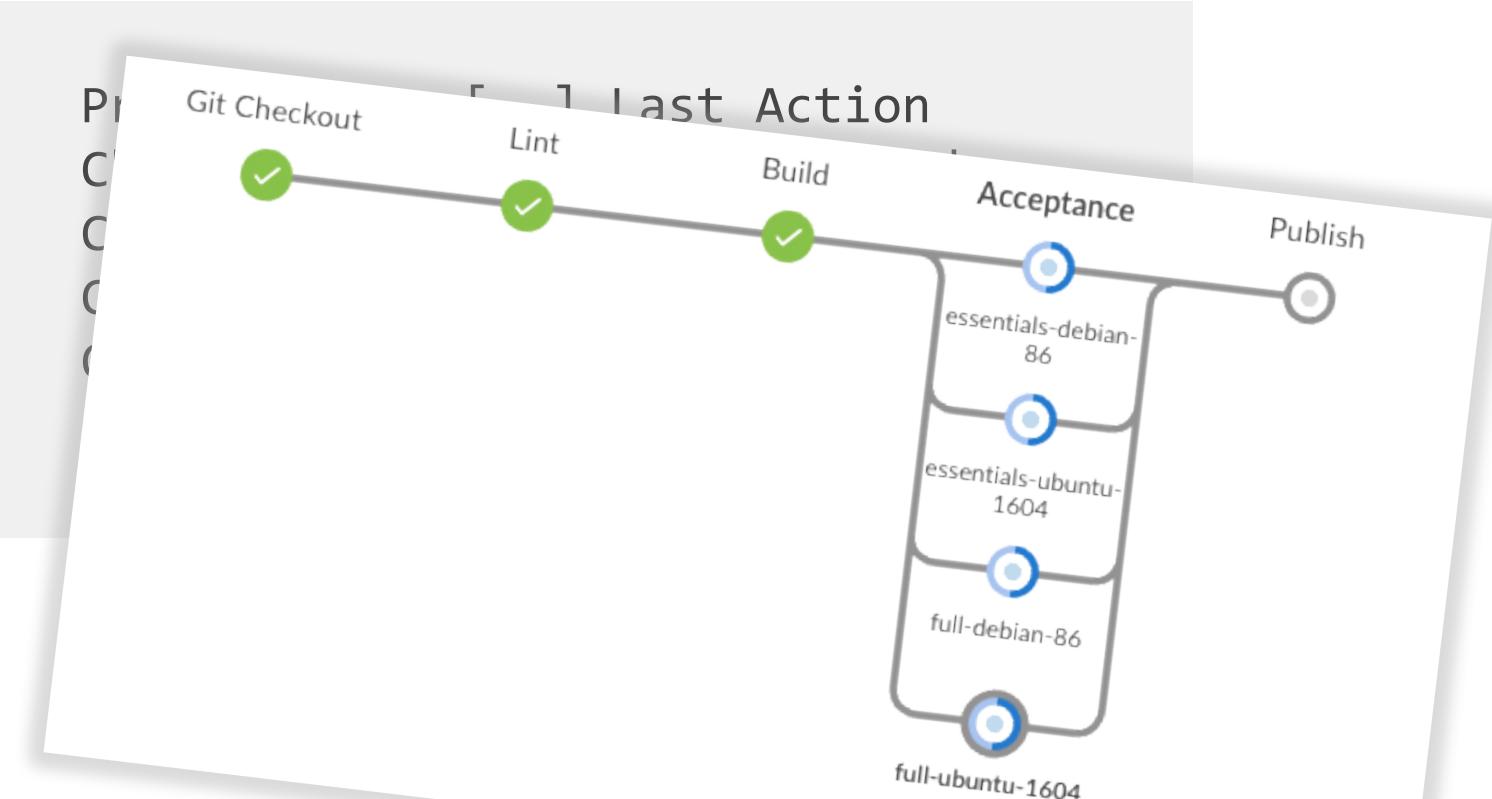
 StephenKing [BUGFIX] Forgotten statement to delete tempfile	..
ArchiveArtifacts.groovy	Feels like I'm learning Groovy..
BerkshelfInstall.groovy	Don't use try/catch
BerkshelfUpload.groovy	Really upload cookbook
Lint.groovy	Feels like I'm learning Groovy..
Pipeline.groovy	Not clear to me, why it clones to workspace@script/
SlackPostBuild.groovy	Add SlackPostBuildAction
TidyUp.groovy	[REMOVED]

Parallelize Integration Tests

- Run *Test-Kitchen* (integration test for Chef cookbooks)
- Run all instances in parallel (by Jenkins)

```
$ kitchen list
```

Instance	Driver	Provider	Last Action
default-debian-78	Docker	Cookbook	Git Checkout
default-debian-82	Docker	Cookbook	Lint
physical-debian-78	Docker	Cookbook	Build
physical-debian-82	Docker	Cookbook	Acceptance
production-debian-78	Docker	Cookbook	Publish
production-debian-82	Docker	Cookbook	



Parallelize Integration Tests (2)

- Goal: Extract instance list, run kitchen commands in parallel
- Expected result:

```
parallel(  
  'default-debian-82': {  
    node {  
      unstash('cookbook-tk')  
      sh('kitchen test --destroy always default-debian-82')  
    }  
  },  
  'physical-debian-82': {  
    node {  
      unstash('cookbook-tk')  
      sh('kitchen test --destroy always physical-debian-82')  
    }...  
  }
```

Parallelize Integration Tests (3)

- Grab list of instance names

```
def ArrayList<String> getInstanceNames(){
    def instanceNames = []
    node {
        def lines = sh(script: 'kitchen list', returnStdout: true).split('\n')
        for (int i = 1; i < lines.size(); i++) {
            instanceNames << lines[i].tokenize(' ')[0]
        }
    }
    return instanceNames
}
```

Parallelize Integration Tests (4)

```
def Closure getNodeForInstance(String instanceName) {  
    return {  
        // this node (one per instance) is later executed in parallel  
        node {  
            // restore workspace  
            unstash('cookbook-tk')  
            sh('kitchen test --destroy always ' + instanceName)  
        }}}  
}
```

```
for (int i = 0; i < instanceNames.size(); i++) {  
    def instanceName = instanceNames.get(i)  
    plNodes[instanceName] = this.getNodeForInstance(instanceName)  
}  
parallel plNodes
```

Failure Notification

- Pipeline stops, when any step fails
- But.. I want that info in Slack!

```
def execute() {  
    this.prepare()  
    this.run(new Lint())  
    this.run(new BerkshelfInstall())  
    this.run(new TestKitchen())  
    this.run(new ArchiveArtifacts())  
  
    if (env.BRANCH_NAME == "master") {  
        this.run(new BerkshelfUpload())  
    }  
}
```

```
def run(Object step){  
    try {  
        step.execute()  
    } catch (err) {  
        this.postBuildNotify  
        failTheBuild("Build failed")  
    }  
}
```



#infrastructure-status

★ | 8 12 | 0 0 | Add a topic



github BOT 19:55

[jenkins-chefci:master] 1 new commit by Steffen Gebert:

| 9cc9c8a [TESTS] Can't test docker in docker - Steffen Gebert



Chef CI BOT 19:55

TYPO3-cookbooks/jenkins-chefci/master build #24 started:

[Branch indexing]

<https://chef-ci.typo3.org/job/TYPO3-cookbooks/job/jenkins-chefci/job/master/24/>



Chef CI BOT 20:05

TYPO3-cookbooks/jenkins-chefci/master build #24 waiting for input:

<https://chef-ci.typo3.org/job/TYPO3-cookbooks/job/jenkins-chefci/job/master/24/>



github BOT 20:15

[jenkins-chefci] New tag 0.0.10 was pushed by chefcityo3org



Chef CI BOT 20:15

TYPO3-cookbooks/jenkins-chefci/master build #24 finished:

<https://chef-ci.typo3.org/job/TYPO3-cookbooks/job/jenkins-chefci/job/master/24/>

Summary

- Finally nice pipelines!
 - The way to go with Jenkins
 - Many Jenkins plugins already compatible
- Pipeline as code!
 - Versioned
 - Doesn't mess up Jenkins jobs
 - Code sharing
- New UI stuff still freaking broken
- Endless possibilities - can be complex and painful*
- Chained pipelines? Yes, but..

*IMHO still better than YAML

Further Reading

- Pipeline Documentation:
<https://jenkins.io/doc/book/pipeline/>
- Step documentation:
<https://jenkins.io/doc/pipeline/steps/>
- Pipeline shared libraries:
<https://jenkins.io/doc/book/pipeline/shared-libraries/>
- Parallel execution:
<https://jenkins.io/blog/2016/06/16/parallel-test-executor-plugin/>
- Extending pipeline DSL:
<https://jenkins.io/blog/2016/04/21/dsl-plugins/>
- Controlling the Flow with Stage, Lock, and Milestone:
<https://jenkins.io/blog/2016/10/16/stage-lock-milestone/>
- TYPO3's Chef CI:
<https://chef-ci.typo3.org>

*Slides will be available at
st-g.de/speaking*