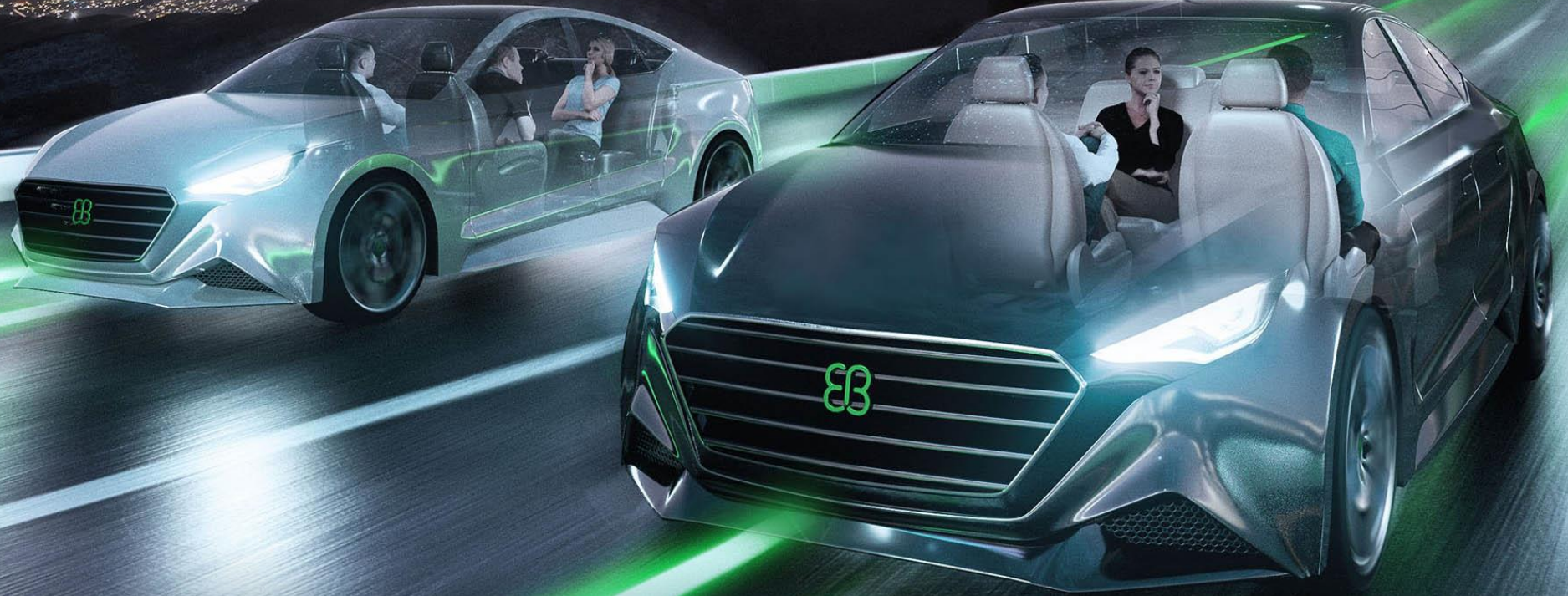


Continuous Delivery with Jenkins Pipelines (incl. Advanced Topics)



Elektrobit

Roman Pickl
23.05.2018



join at [Slido.com](https://www.slido.com) with #K100

Demo

1. `docker run -p 8080:8080 jenkinsci/blueocean` (add `-p 44444:44444` or any other port to try the ssh linter)
2. Go to `http://localhost:8080/`
3. Unlock jenkins with initialpw from log file
4. Install suggested plugins (you may have to continue the process in case any plugins are broken and update plugins later on)
5. Create admin user / or continue with admin and initialpw
6. Open `http://localhost:8080/blue/pipelines`
7. Generate new Pipeline with Github repository
8. Create Jenkinsfile (e.g. <https://github.com/rompic/jenkinspipeline>)

How did I end up here?

Roman Pickl (@rompic)

- Technical Project Manager @ Elektrotbit
- Uses Jenkins since 2012
- Loves CI/CD/DevOps
- Here to learn



Continuous Delivery (CD)

Automated implementation of your system's build, deploy, test, release process

- Every change results in a build
 - Every build is a release candidate
 - Delivery can be done at any time, on any environment
- Make releases a non-event

Deployment Pipeline provides:

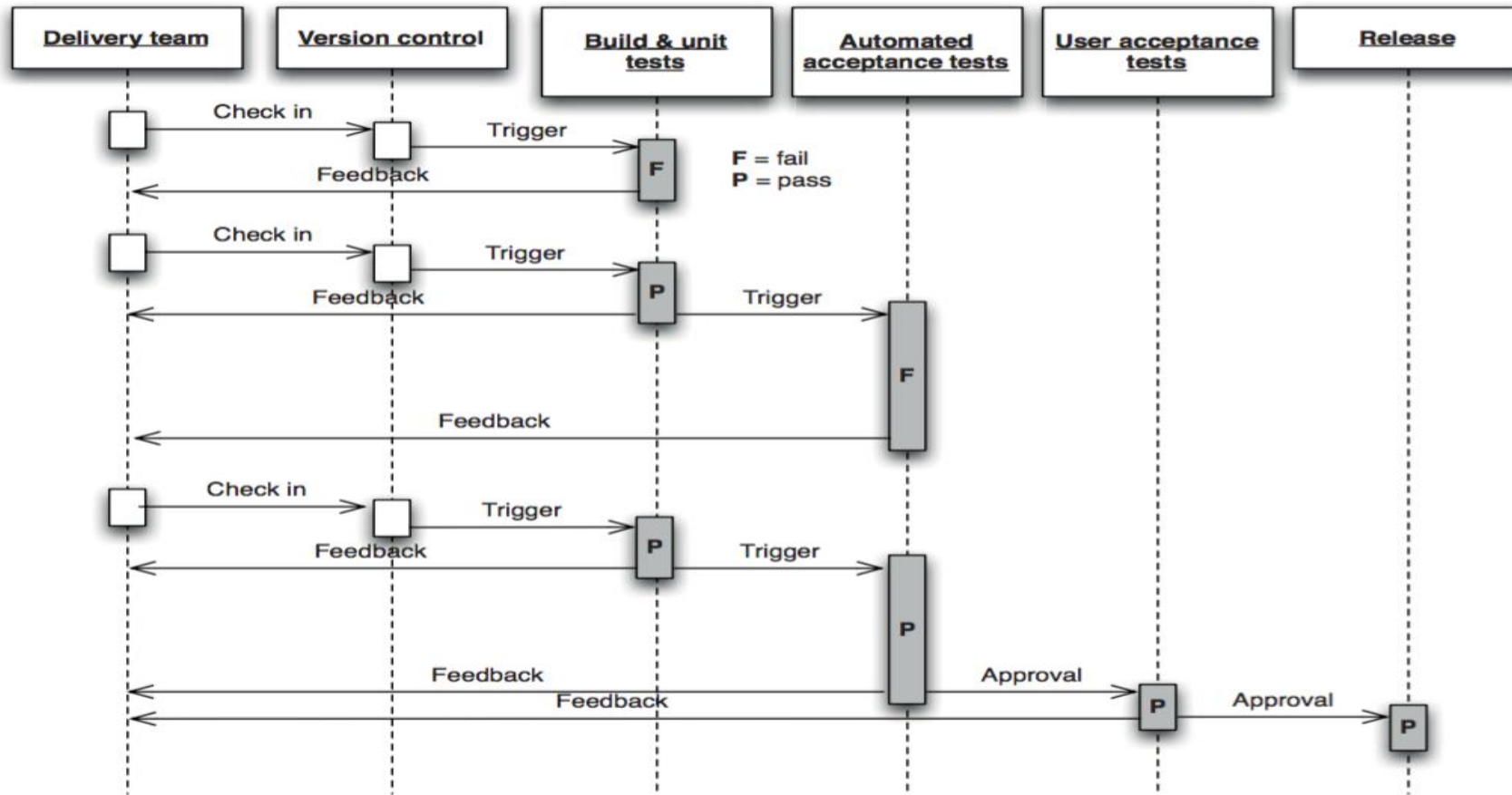
- Visibility
- Feedback
- Control



Read these books if you want to know more!

Jez Humble on Continuous Delivery(2012): <https://www.youtube.com/watch?v=skLJucsRTw>

Deployment Pipelines (Let's build it with jenkins)



Jenkins

#1 Continuous Integration and Delivery Server

- Created by Kohsuke Kawaguchi
- Initial Release 2005 (Hudson)
- Open Source (MIT License)
- Active and independent community (<https://jenkins.io>)
- 164,000 active installations
- 1,500+ plugins (!)

- Since 2.0 Pipelines (April 2016) are first class citizens
- Pipeline as Code (Jenkinsfile).
- New User Experience “Blue Ocean” with Blue Ocean Pipeline Editor
- Blue Ocean 1.5 released in April 2018

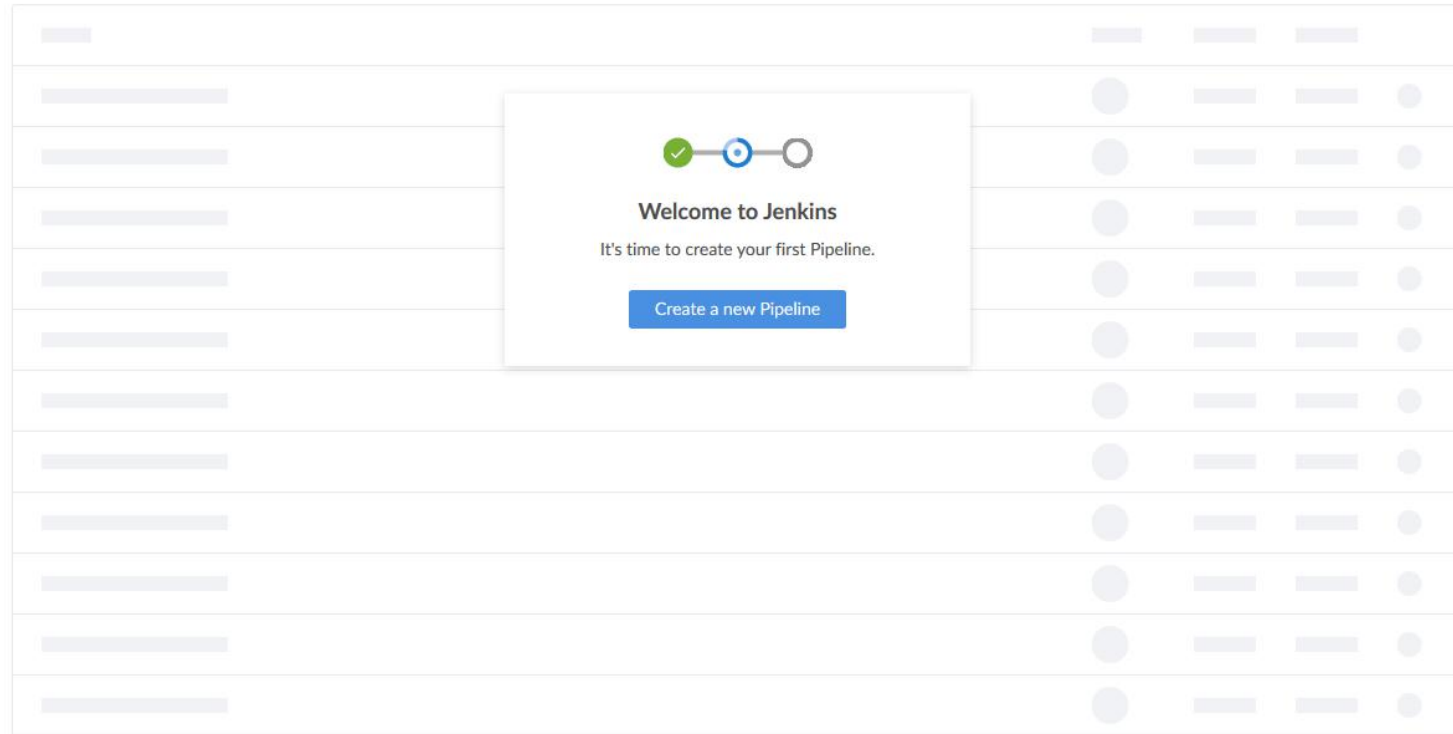
http://dduportal.github.io/presentations/mixit-2017-jenkins-pipeline/slides.html#/meet_jenkins
<https://jenkins.io/blog/2017/04/05/say-hello-blueocean-1-0/>
<http://stats.jenkins.io/>



Welcome Blue Ocean!

Jenkins Pipelines Administration [Logout](#)

Pipelines [New Pipeline](#)

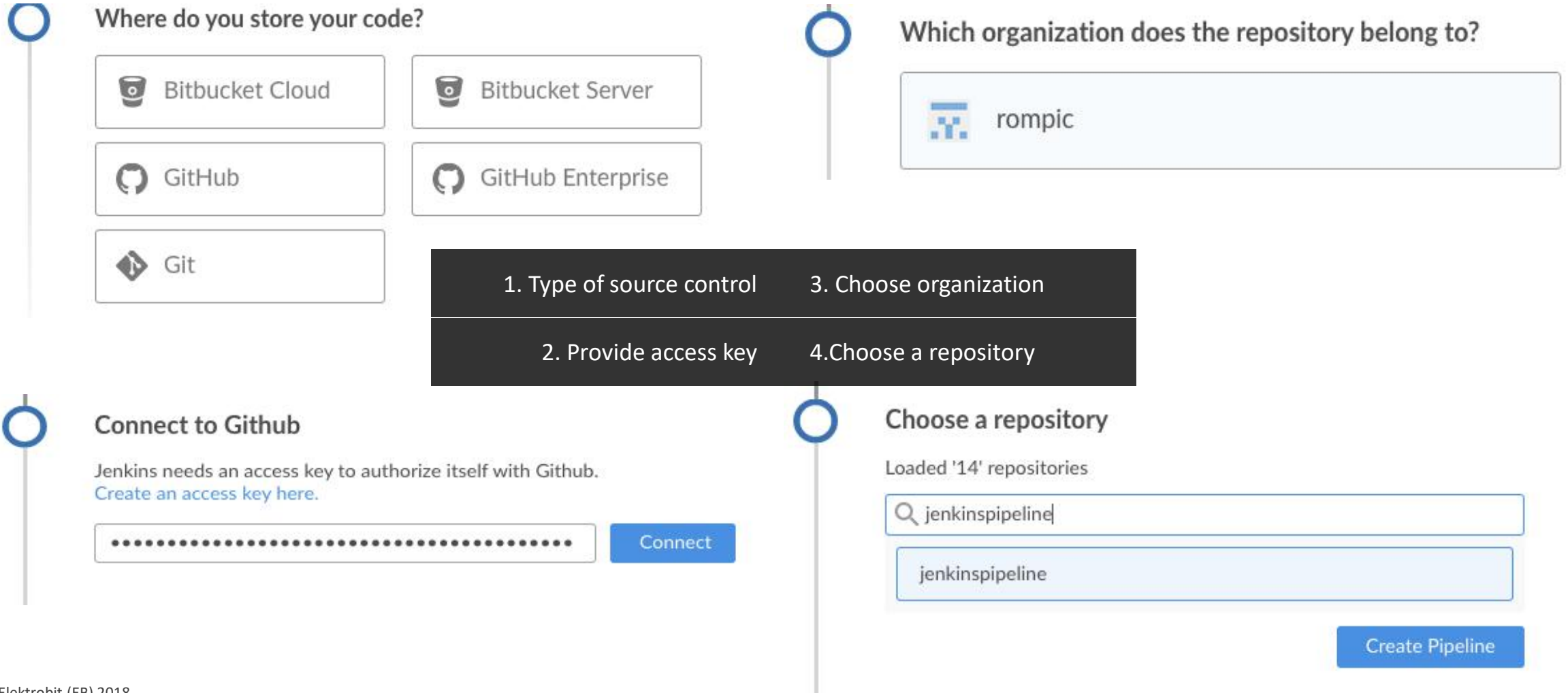


The screenshot shows the Jenkins Blue Ocean interface. A central modal dialog box is displayed with the following content:

- A progress indicator with three circles: the first is green with a checkmark, the second is blue with a plus sign, and the third is grey.
- Welcome to Jenkins**
- It's time to create your first Pipeline.
- [Create a new Pipeline](#)

The background of the interface is dimmed, showing a list of pipeline items with columns for name, status, and actions.

Create a Pipeline Wizard



Jenkinsfile

Written in a Groovy DSL

“Jenkinsfile” in top level folder (different path possible since June 2017 <https://issues.jenkins-ci.org/browse/JENKINS-34561>)

Store in SCM (e.g. GIT) for additional benefits

- Code review/iteration
- Audit trail
- Single source of truth

Supports two syntaxes (can be mixed)

- Declarative pipelines (easier; “new”; 1.0 Feb 2017)
- Scripted pipelines (more powerful)

<https://jenkins.io/doc/book/pipeline/jenkinsfile/>

Jenkinsfile (Declarative Pipeline)

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        echo 'Building..'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing..'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying....'
      }
    }
  }
}
```

Scripted vs. Declarative

Scripted Pipelines

Jenkinsfile (Scripted Pipeline)

```
node {
    stage('Example') {
        if (env.BRANCH_NAME == 'master') {
            echo 'I only execute on the master branch'
        } else {
            echo 'I execute elsewhere'
        }
    }
}
```

Jenkinsfile (Scripted Pipeline)

```
node {
    stage('Example') {
        try {
            sh 'exit 1'
        }
        catch (exc) {
            echo 'Something failed, I should sound the klaxons!'
            throw
        }
    }
}
```

<https://jenkins.io/doc/book/pipeline/syntax/>

Declarative Pipeline

Jenkinsfile (Declarative Pipeline)

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building..'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing..'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying....'
            }
        }
    }
}
```

Scripted vs. Declarative

Scripted Pipelines

- imperative programming model
 - fully featured programming environment,
 - higher flexibility and extensibility
 - very few limits
- ➔ for power-users and more complex requirements

Both

- use Groovy
 - same Pipeline sub-system underneath
 - mostly use same steps
 - able to utilize Shared Libraries
- ➔ can be mixed using the script step

Declarative Pipeline

- declarative programming model
 - simpler and more opinionated syntax for authoring Jenkins Pipeline.
 - Allows for validation and a visual editor
 - limits what is available to the user
- ➔ ideal choice for simpler continuous delivery pipelines

See <https://jenkins.io/blog/2017/01/19/converting-conditional-to-pipeline/> for a more complex example of migrating a freestyle job to a declarative/scripted pipeline.

<https://jenkins.io/doc/book/pipeline/syntax/>

script Step

- takes a block of Scripted Pipeline & executes that in the Declarative Pipeline
- can provide a useful "escape hatch".
- script blocks of non-trivial size and/or complexity should be moved into Shared Libraries

Jenkinsfile (Declarative Pipeline)

```
pipeline {
  agent any
  stages {
    stage('Example') {
      steps {
        echo 'Hello World'

        script {
          def browsers = ['chrome', 'firefox']
          for (int i = 0; i < browsers.size(); ++i) {
            echo "Testing the ${browsers[i]} browser"
          }
        }
      }
    }
  }
}
```


Settings

```
pipeline {
  agent none    //don't block an executor for approval
  //see http://bit.ly/2qrz2Ty
  // environment, options, tools, parameters
  //and triggers can also be defined here for the whole pipeline
  triggers { pollSCM('H/5 * * * *') } // poll every 5 mins
  options {
    timeout(time: 60, unit: 'DAYS')
    buildDiscarder(logRotator(numToKeepStr: '30'))
  }
}
```

Stages

```
stages {
  stage('Build & unit tests') {
    agent any
//    tools {
//      //this is ignored at top level if agent none is specified.
//      //jdk 'Oracle Java 8' (defined in jenkins setup)
//    }
    steps {
      echo 'running build and unit tests'
      deleteDir() //delete everything in this workspace
      checkout scm
      //sh './gradlew build'
      //archiveArtifacts
      //stash
    }
  }
}

//  post {
//    always{
//      //publish unit tests
//      //junit 'path/to/tests/*.xml'
//    }
//  }
```

Parallel Execution

```
stage('Automated Acceptance tests'){
  parallel{ //since declarative pipelines 1.2|
    stage('Automated Acceptance tests Firefox'){
      agent any
      steps{
        // unstash
        echo "testing Firefox"
      }
      //publish unit tests (omitted here)
    }
    stage('Automated Acceptance tests chrome'){
      agent any
      steps{
        // unstash
        echo "testing chrome"
      }
      //publish unit tests (omitted here)
    }
  }
}
```

Approval

```
stage('Deploy to Stage for User acceptance tests') {  
  when { branch 'master' } //only offer this option on master  
  steps {  
    milestone(1)  
    timeout(time: 30, unit: 'DAYS') {  
      input message: 'Deploy to Stage?', submitter: 'admin,tom.testster,pete.pm'  
    }  
    milestone(2)  
  }  
}
```

Deploy to Live / Release omitted here

Post Build Notifications

```
post {
  //feedback on failure (also always, success, unstable, changed, fixed, regression| available)
  failure {
    emailext (
      subject: "FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]'",
      body: """"<p>FAILED: Job '${env.JOB_NAME} [${env.BUILD_NUMBER}]':</p>
        <p>Check console output at &QUOT;<a href='${env.BUILD_URL}'>
          ${env.JOB_NAME} [${env.BUILD_NUMBER}]</a>&QUOT;</p>""",
      recipientProviders: [[class: 'CulpritsRecipientProvider']]
    )
  }
}
```

Blue Ocean 1.5 released 11.04.2018

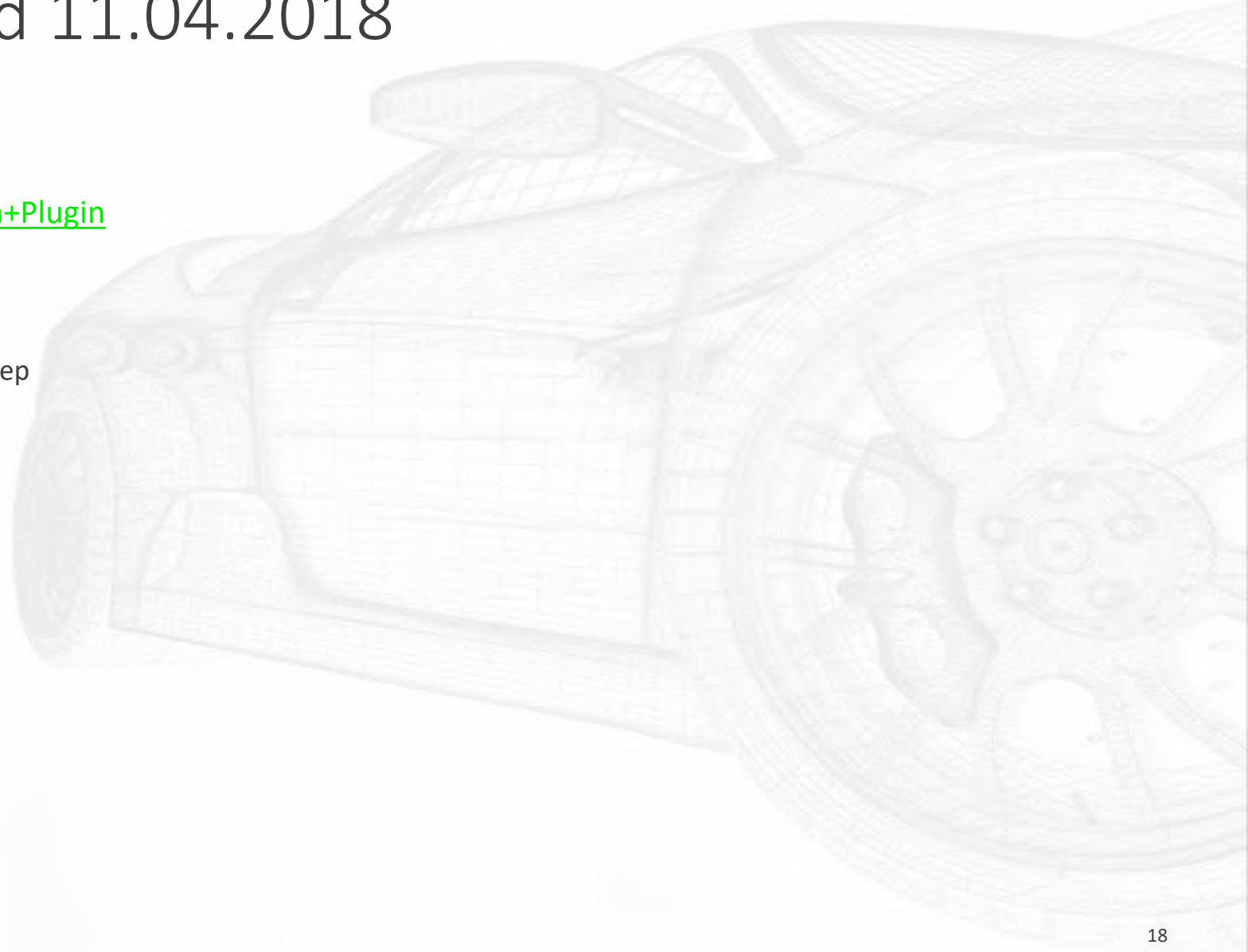
<https://wiki.jenkins.io/display/JENKINS/Blue+Ocean+Plugin>

- Latest additions
 - Show the downstream jobs launched with the *build* step
 - Reorder steps In pipeline editor by drag and drop
 - Pagination of artifacts page

Public roadmap

- <https://jenkins.io/projects/blueocean/roadmap/>

<https://jenkins.io/blog/2018/04/18/blueocean-1-5-0/>



Overview

Jenkins Pipelines Administration Logout

jenkinspipeline ☆ ⚙️ Activity Branches Pull Requests

STATUS	RUN	COMMIT	BRANCH	MESSAGE	DURATION	COMPLETED
⏸	2	ec0e8f5	master	updated comment on failure modes	33m 7s	-
⏸	1	f30f169	master	Branch indexing	6d 13h 36m 21s	-
✅	1	f30f169	dev	Branch indexing	54s	7 days ago

Approval

jenkinspipeline 2

Branch: master [🔗](#) 37m 42s Changes by noreply
Commit: ec0e8f5 - Started by an SCM change

Deploy LIVE? - 20s

✓	> 3	- The milestone step forces all builds to go through in order	<1s
	▼	Wait for interactive input	33s

Deploy to Live?

Detail

✓ jenkinspipeline 2 Pipeline Changes Tests Artifacts ↺ ✎ ⚙ 📄 Logout ✕

Branch: master [🔗](#) 🕒 38m 46s 👤 Changes by noreply
Commit: ec0e8f5 🕒 a few seconds ago 👤 Started by an SCM change

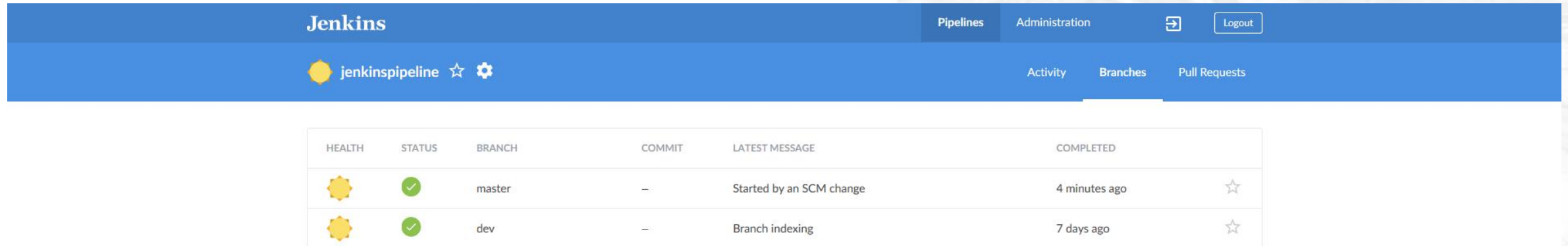
```
graph LR; Start((Start)) --> Build[Build & unit tests]; Build --> AT1[Automated Acceptance tests]; AT1 --> AT2[Automated Acceptance tes...]; AT2 --> AT3[Automated Acceptance tes...]; AT3 --> DS1[Deploy to Stage for User accepta...]; DS1 --> DS2[Deploy to Stage]; DS2 --> DL[Deploy LIVE?]; DL --> Release[Release]; Release --> End((End));
```

Release - 2s 🔗 📄







- ✓ > General SCM 2s
- ✓ v deploying to live and running smoke tests — Print Message <1s

```
1  deploying to live and running smoke tests
```

Multi branch support



The screenshot shows the Jenkins web interface for a pipeline named 'jenkinspipeline'. The interface includes a top navigation bar with 'Pipelines', 'Administration', and 'Logout' buttons. Below the navigation bar, there are tabs for 'Activity', 'Branches', and 'Pull Requests'. The main content area displays a table of pipeline runs for the 'jenkinspipeline'.

HEALTH	STATUS	BRANCH	COMMIT	LATEST MESSAGE	COMPLETED
		master	-	Started by an SCM change	4 minutes ago 
		dev	-	Branch indexing	7 days ago 

Multi branch (When)

jenkinspipeline 1

Branch: dev 54s No changes
Commit: f30f169 7 days ago Branch indexing

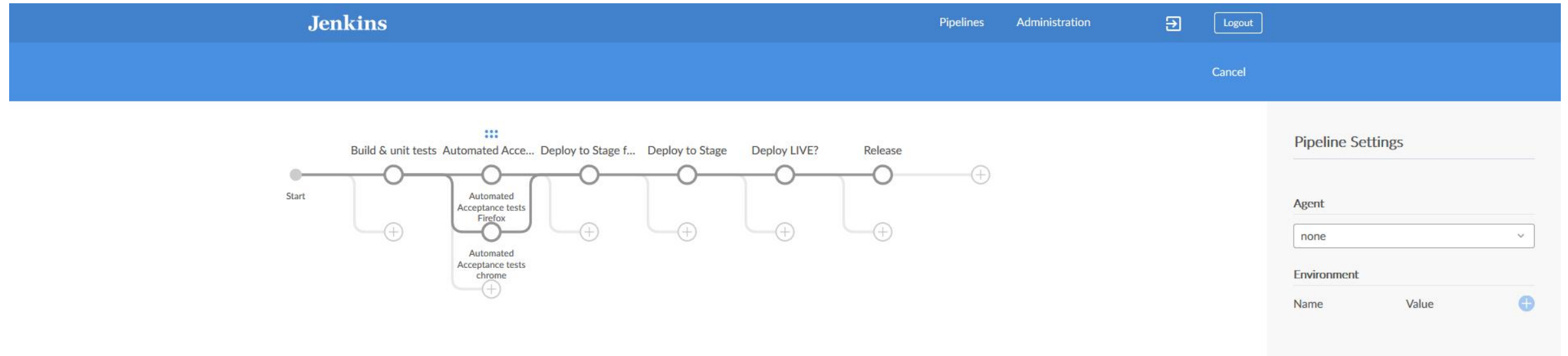
Automated Acceptance tests / Automated Acceptance tests Firefox - <1s

General SCM	2s
testing Firefox - Print Message	<1s

Pipeline Development and Advanced Tools

- Blue Ocean Pipeline Editor
- Snippet Generator
- Directive Generator (*NEW*)
- Auto-Convert Freestyle Jobs to Jenkins Pipeline
- Replay Feature
- IntelliJ IDEA GDSL – Autocomplete
- Command-line Pipeline Linter
- Jenkins File Runner (*NEW*)
- Unit Testing Jenkins Pipelines
- Shared Libraries

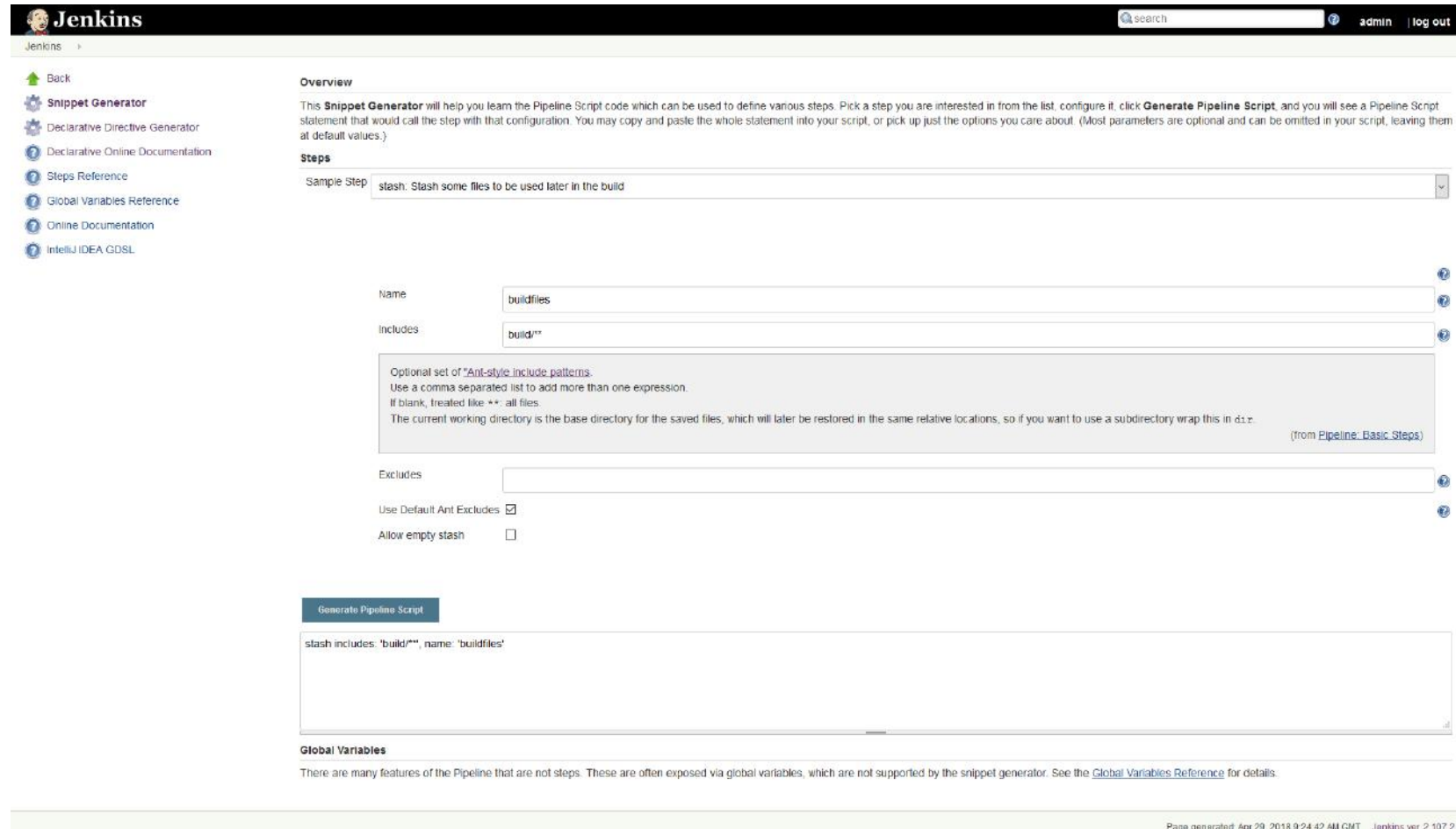
Pipeline Editor



The screenshot shows the Jenkins Pipeline Editor interface. The top navigation bar includes the Jenkins logo, 'Pipelines', 'Administration', a home icon, and a 'Logout' button. Below the navigation bar is a 'Cancel' button. The main area displays a pipeline graph with stages: 'Start', 'Build & unit tests', 'Automated Acce...', 'Deploy to Stage f...', 'Deploy to Stage', 'Deploy LIVE?', and 'Release'. The 'Automated Acce...' stage is expanded to show two parallel tasks: 'Automated Acceptance tests Firefox' and 'Automated Acceptance tests chrome'. Each task has a plus sign icon below it. To the right of the pipeline graph is a 'Pipeline Settings' panel with a dropdown for 'Agent' (set to 'none') and a table for 'Environment' with columns 'Name' and 'Value'.

- GitLab currently not supported (planned! <https://issues.jenkins-ci.org/browse/JENKINS-43976>)
- Workaround: <http://localhost:8080/blue/organizations/jenkins/pipeline-editor/>
- Ctrl-S / Cmd-S to open the load save dialog

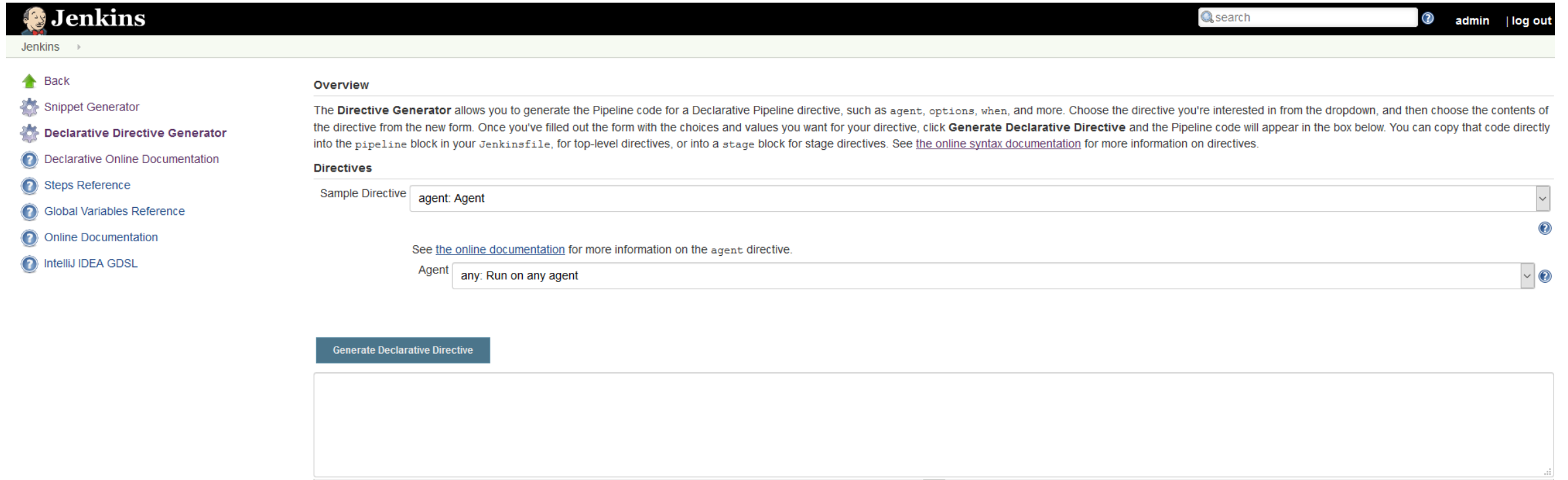
Snippet generator



The screenshot shows the Jenkins Snippet Generator web interface. At the top, the Jenkins logo and navigation links (admin, log out) are visible. A sidebar on the left contains links for Back, Snippet Generator, Declarative Directive Generator, Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, and IntelliJ IDEA GDSDL. The main content area is titled "Overview" and explains the tool's purpose. Below this, a "Steps" section features a dropdown menu with "stash: Stash some files to be used later in the build" selected. The configuration fields include "Name" (buildfiles), "Includes" (build/**), "Excludes" (empty), "Use Default Ant Excludes" (checked), and "Allow empty stash" (unchecked). A "Generate Pipeline Script" button is present. The generated script is shown in a text area: `stash includes: 'build/**', name: 'buildfiles'`. A "Global Variables" section at the bottom provides additional context.

- <http://localhost:8080/pipeline-syntax>

Declarative generator



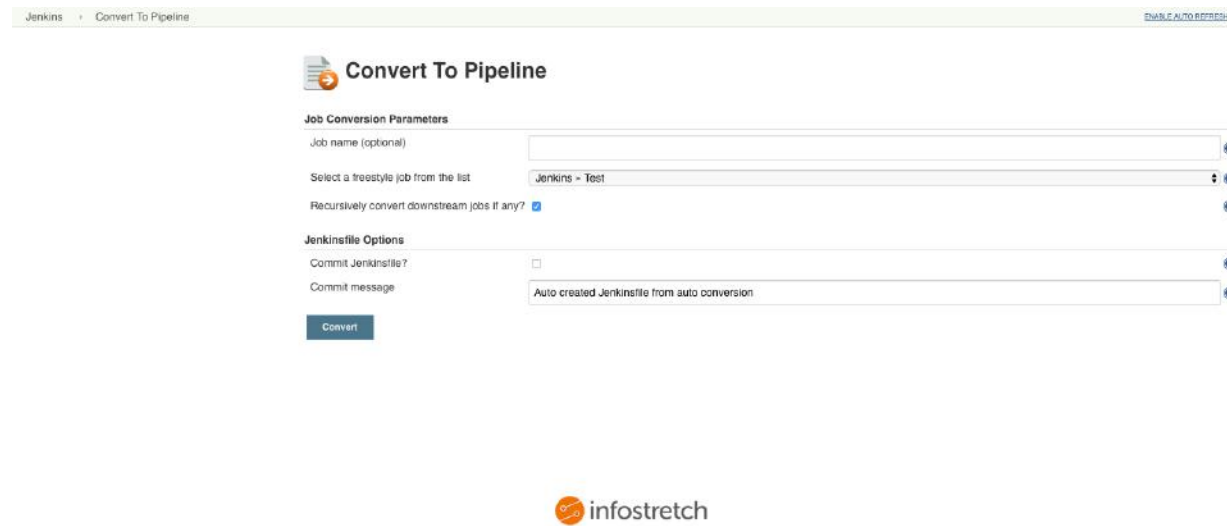
The screenshot shows the Jenkins Declarative Directive Generator interface. At the top, there is a navigation bar with the Jenkins logo, a search bar, and user information (admin | log out). Below the navigation bar, there is a sidebar with a list of links: Back, Snippet Generator, Declarative Directive Generator (highlighted), Declarative Online Documentation, Steps Reference, Global Variables Reference, Online Documentation, and IntelliJ IDEA GDSDL. The main content area is titled "Overview" and contains the following text: "The **Directive Generator** allows you to generate the Pipeline code for a Declarative Pipeline directive, such as `agent`, `options`, `when`, and more. Choose the directive you're interested in from the dropdown, and then choose the contents of the directive from the new form. Once you've filled out the form with the choices and values you want for your directive, click **Generate Declarative Directive** and the Pipeline code will appear in the box below. You can copy that code directly into the `pipeline` block in your `Jenkinsfile`, for top-level directives, or into a `stage` block for stage directives. See [the online syntax documentation](#) for more information on directives."

Below the overview text, there is a section titled "Directives" with a dropdown menu for "Sample Directive" set to "agent: Agent". Below this, there is a link to "the online documentation" for more information on the `agent` directive. Below that, there is another dropdown menu for "Agent" set to "any: Run on any agent". At the bottom of the interface, there is a button labeled "Generate Declarative Directive" and a large empty text area for the generated code.

- <http://localhost:8080/directive-generator/>
- <https://jenkins.io/blog/2018/04/09/whats-in-declarative/>

Auto-Convert Freestyle Jobs to Jenkins Pipeline

Plugin to automatically convert Freestyle Jobs to Jenkins Pipeline



The screenshot shows the Jenkins 'Convert To Pipeline' plugin interface. At the top, there is a breadcrumb 'Jenkins > Convert To Pipeline' and a 'DISABLE AUTO-CONVERSION' link. The main heading is 'Convert To Pipeline' with a document icon. Below this, there are two sections: 'Job Conversion Parameters' and 'Jenkinsfile Options'. In the 'Job Conversion Parameters' section, there is a text input for 'Job name (optional)', a dropdown menu for 'Select a freestyle job from the list' (currently showing 'Jenkins - Test'), and a checkbox for 'Recursively convert downstream jobs if any?' which is checked. In the 'Jenkinsfile Options' section, there is a checkbox for 'Commit Jenkinsfile?' which is unchecked, and a text input for 'Commit message' containing 'Auto created Jenkinsfile from auto conversion'. A 'Convert' button is located at the bottom left of the form. The footer of the page features the 'infostretch' logo.

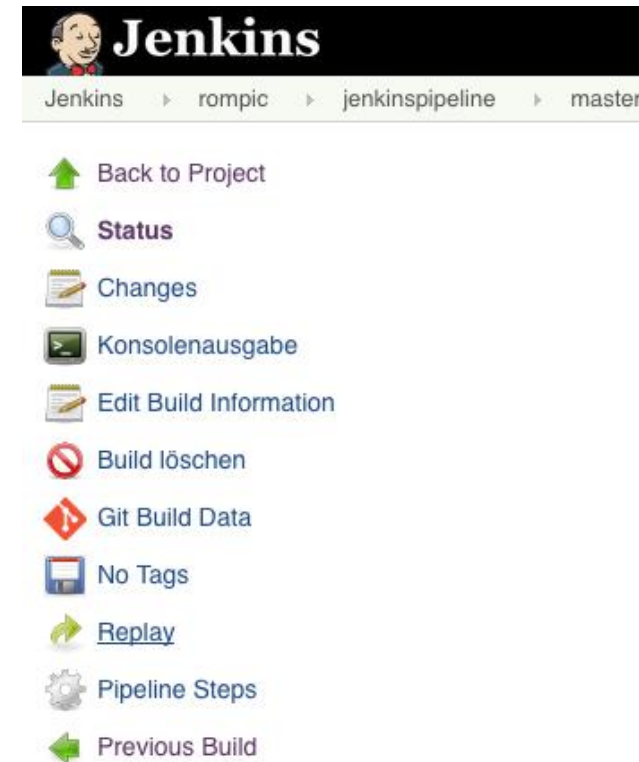
<https://jenkins.io/blog/2017/12/15/auto-convert-freestyle-jenkins-jobs-to-coded-pipeline/>

<https://wiki.jenkins.io/display/JENKINS/Convert+To+Pipeline+Plugin>

Replay Feature

Green sub-title

- Allows for quick modifications and execution of an existing (valid!) Pipeline without changing the Pipeline configuration or creating a new commit.
- Once you are satisfied with the changes, you can use Replay to view them again, copy them back to your Pipeline job or Jenkinsfile, and then commit them using your usual engineering processes

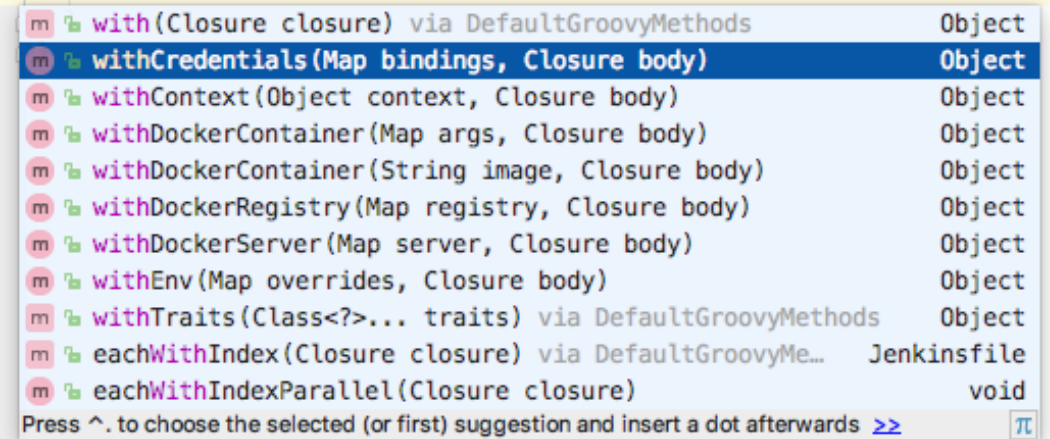


IntelliJ IDEA GDSL - Autocomplete


Green sub-title

- Auto completion of steps for **scripted** pipelines
- Install Groovy Plugin
- Download <http://localhost:8080/pipeline-syntax/gdsl>
- Add it as e.g. pipeline.gdsl to your projects src path

```
node( label: ''){
  stage( name: 'asdf'){
    echo( message: "asdf")
    with
```



m	with(Closure closure) via DefaultGroovyMethods	Object
m	withCredentials(Map bindings, Closure body)	Object
m	withContext(Object context, Closure body)	Object
m	withDockerContainer(Map args, Closure body)	Object
m	withDockerContainer(String image, Closure body)	Object
m	withDockerRegistry(Map registry, Closure body)	Object
m	withDockerServer(Map server, Closure body)	Object
m	withEnv(Map overrides, Closure body)	Object
m	withTraits(Class<?>... traits) via DefaultGroovyMethods	Object
m	eachWithIndex(Closure closure) via DefaultGroovyMe...	Jenkinsfile
m	eachWithIndexParallel(Closure closure)	void

Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> 

See

<https://st-g.de/2016/08/jenkins-pipeline-autocompletion-in-intellij>

<https://stackoverflow.com/questions/41062514/use-gdsl-file-in-a-java-project-in-intellij>

<https://stackoverflow.com/a/41149255/3165782>

for setting it up.

Command-line Pipeline Linter

Validate **Declarative Pipelines** from the cli before actually running it/checking it in.

Linting via the CLI with SSH

```
# ssh (Jenkins CLI)
# JENKINS_SSHD_PORT=[sshd port on master]
# JENKINS_HOSTNAME=[Jenkins master hostname]
ssh -p $JENKINS_SSHD_PORT $JENKINS_HOSTNAME declarative-linter < Jenkinsfile
```

Errors encountered validating Jenkinsfile:

```
WorkflowScript: 30: Unknown stage section "step". Starting with version 0.5, steps in a stage must be in a steps block. @ line 30,
  stage ('Test') {
  ^
```

```
WorkflowScript: 30: Nothing to execute within stage "Test" @ line 34, column 5.
  stage ('Test') {
  ^
```

See <https://jenkins.io/doc/book/pipeline/development/#linter> for details. Remember to enable SSH access, expose a port on your docker container and add ssh key to try this!

jenkinsfile-runner

So i guess we can run now run a job in jenkins which downloads jenkins to run a jenkins job ...

Experiment to package Jenkins pipeline execution as a command line tool.

Use cases include:

- Assist editing and testing Jenkinsfile locally
- Use Jenkins in Function-as-a-Service context
- Integration test shared libraries

- downloads latest Jenkins LTS
- installs plugins as defined by a plugins.txt file
- setup .jenkinsfile-runner directory
- runs Jenkins master headless
- run a single job based on a local Jenkinsfile, then shutdown on completion.

<https://github.com/ndeloof/jenkinsfile-runner>



Shared Libraries

Green sub-title

Share parts of Pipelines between various projects to reduce redundancies and keep code "DRY".

Functions can then be called from Jenkinsfiles.

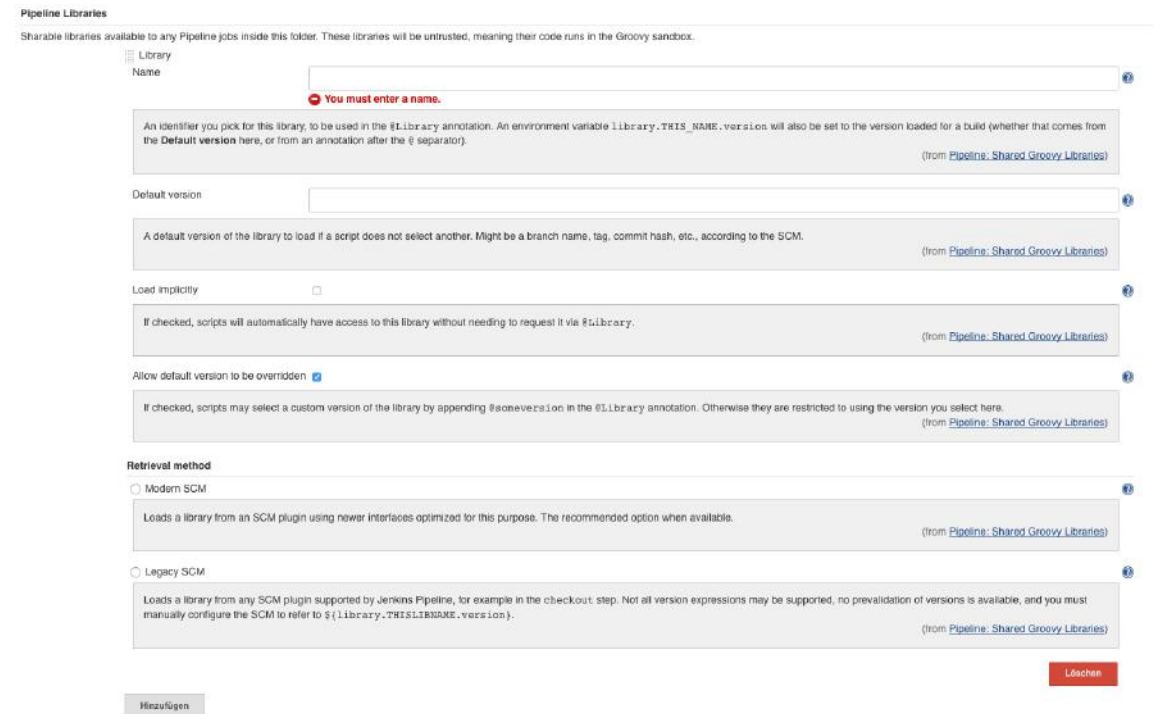
See

<https://jenkins.io/doc/book/pipeline/shared-libraries/>

and

<https://jenkins.io/blog/2017/10/02/pipeline-templates-with-shared-libraries/>

for more details.



Pipeline Libraries

Sharable libraries available to any Pipeline jobs inside this folder. These libraries will be untrusted, meaning their code runs in the Groovy sandbox.

Library Name

You must enter a name.

An identifier you pick for this library, to be used in the `@Library` annotation. An environment variable `!Library.TRIS_NAME.version` will also be set to the version loaded for a build (whether that comes from the **Default version** here, or from an annotation after the `@` separator).

(from Pipeline: Shared Groovy Libraries)

Default version

A default version of the library to load if a script does not select another. Might be a branch name, tag, commit hash, etc., according to the SCM.

(from Pipeline: Shared Groovy Libraries)

Load implicitly

If checked, scripts will automatically have access to this library without needing to request it via `@Library`.

(from Pipeline: Shared Groovy Libraries)

Allow default version to be overridden

If checked, scripts may select a custom version of the library by appending `#someversion` in the `@Library` annotation. Otherwise they are restricted to using the version you select here.

(from Pipeline: Shared Groovy Libraries)

Retrieval method

Modern SCM

Loads a library from an SCM plugin using newer interfaces optimized for this purpose. The recommended option when available.

(from Pipeline: Shared Groovy Libraries)

Legacy SCM

Loads a library from any SCM plugin supported by Jenkins Pipeline, for example in the `checkout` step. Not all version expressions may be supported, no prevalidation of versions is available, and you must manually configure the SCM to refer to `${!Library.TRISLIBRARY.version}`.

(from Pipeline: Shared Groovy Libraries)

Unit Testing Jenkins Pipelines

- Allows to unit test Pipelines and Shared Libraries before running them in full
- Provides a mock execution environment that can be used to check for expected behavior
- Still quite rough around the edges. (e.g. no support for declarative pipeline yet <https://github.com/lesfurets/JenkinsPipelineUnit/pull/13>)
- See:
 - <https://github.com/lesfurets/JenkinsPipelineUnit>
 - <https://github.com/lesfurets/JenkinsPipelineUnit/blob/master/README.md>
 - <https://issues.jenkins-ci.org/browse/JENKINS-33925>

Things missing / Things to come

- Missing:
 - Support for definition of variables in declarative pipelines (see workaround in <https://issues.jenkins-ci.org/browse/JENKINS-41335>)
 - Keep build forever (<https://issues.jenkins-ci.org/browse/JENKINS-39028>; workaround via shared lib, change to be released)
 - Restart stages for pipelines (checkpoint as commercial feature, <https://issues.jenkins-ci.org/browse/JENKINS-33846> ; feature for declarative pipelines planned: <https://issues.jenkins-ci.org/browse/JENKINS-45455>)
- To Come:
 - More editor coverage of declaration syntax
 - GitLab read/write support
 - Jenkins Essentials
 - Project Cheetah <https://jenkins.io/blog/2018/02/22/cheetah/>
 - For more see <https://jenkins.io/projects/blueocean/roadmap/>

Further references & information I

- Website: <https://jenkins.io>
- Blog: <https://jenkins.io/node/>
- <https://www.slideshare.net/legrimpeur/belgium-jenkins-area-meetup-jenkins-blueocean-and-declarative-pipelines>

- Getting Started
- <https://jenkins.io/doc/book/getting-started/>
- <https://jenkins.io/doc/book/pipeline/syntax/>
- <https://jenkins.io/doc/tutorials/>
- <https://jenkins.io/doc/pipeline/steps/>
- <https://jenkins.io/doc/book/blueocean/getting-started/>
- <https://github.com/jenkinsci/pipeline-model-definition-plugin/wiki/getting%20started>
- <https://jenkins.io/blog/2017/05/18/pipeline-dev-tools/>

Further references & information II

Docker Files:

- <https://github.com/jenkinsci/docker/blob/master/README.md>
- <https://hub.docker.com/r/jenkinsci/blueocean/>

Get in touch!



Elektrobit

Roman Pickl (@rompic)
roman.pickl@elektrobit.com
www.elektrobit.com

