Jenkins Shared Libraries Deep Dive

Julien Pivotto (@roidelapluie)

Day of Jenkins Oslo & Gothenburg

June 2017

whoami

- Julien "roidelapluie" Pivotto
- @roidelapluie
- Sysadmin at inuits
- Automation, monitoring, HA
- Jenkins user for 5+ years

inuits

Jenkins Pipeline

- Were you there this morning? :-)
- One of the best things that happened to Jenkins in 5 years

Pipeline limits

- What when we build 10 projects who look the same?
- How to keep Jenkinsfiles in sync?
- Split deployment from application code
- What if we want to interact with the heart of Jenkins (advanced use cases)
- **Declarative Pipelines are limited by design**

Shared Libraries ?

Why do we need that?

Why do we need that?

- Put logic in Declarative Pipeline
- DRY (e.g. microservices)
- Scalability
- Escape the Groovy Sandbox

What is a "Shared Library"?

What is a "Shared Library"?

- SCM repository (git, ...)
- Groovy code
- Pipeline steps
- Resources (plain scripts, flat files, etc)
- Easily reusable in your Pipeline jobs

The 4 features

- Resources
- Steps
- Untrusted library
- Trusted library

Now let's get prepared!!

Interrupt me at any time if you need help or have questions!!

We will move forward when most of the people are at the same point as me

Our goals today

- We will not build real things (to have low requirements)
- We will (maybe) not do the complete slidedeck... let's see!

Requirements

- Jenkins with the Git Plugin or Filesystem SCM plugin
- 2 repos (gitlab, github, filesystem...) + git client on your laptop (names: "trusted" and "untrusted")
- Python 2 ("python" command)

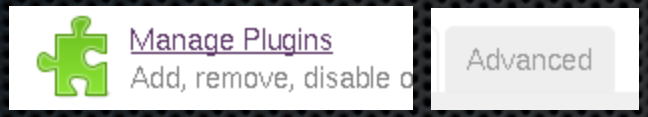
Open your Jenkins instance

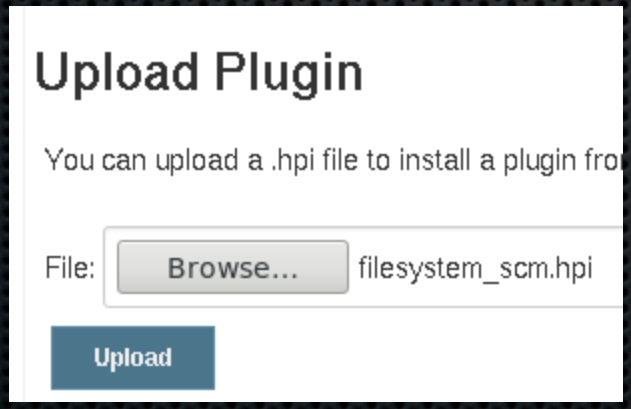
Make sure you have SCM repos your Jenkins has access to

You can use the Git plugin or Filesystem SCM plugin

Install from:

https://roidelapluie.be/uploads/filesystem_scm-1.21.hpi

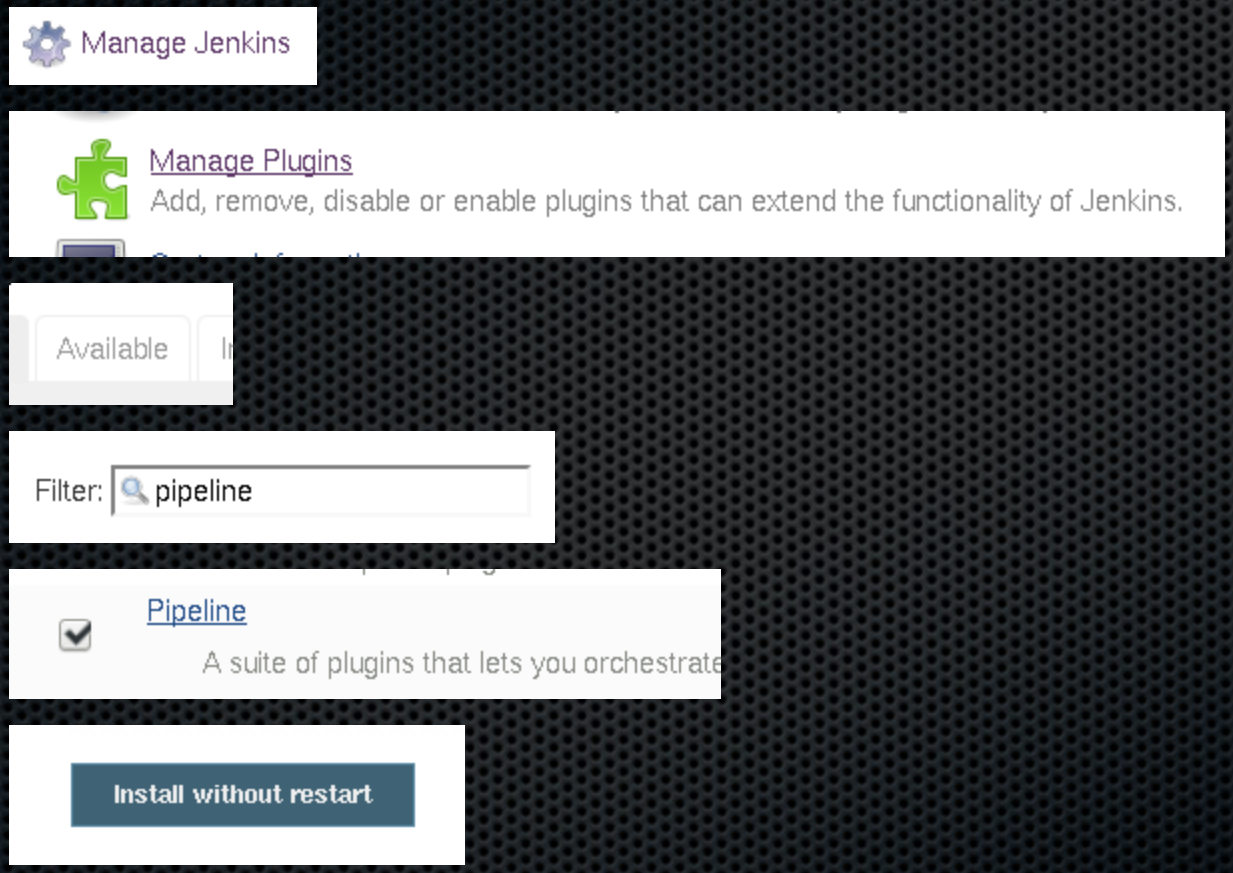




Login to Jenkins (as admin)

We login as admin because it is easier. The workshop will assume that.

Install the Pipeline Plugin



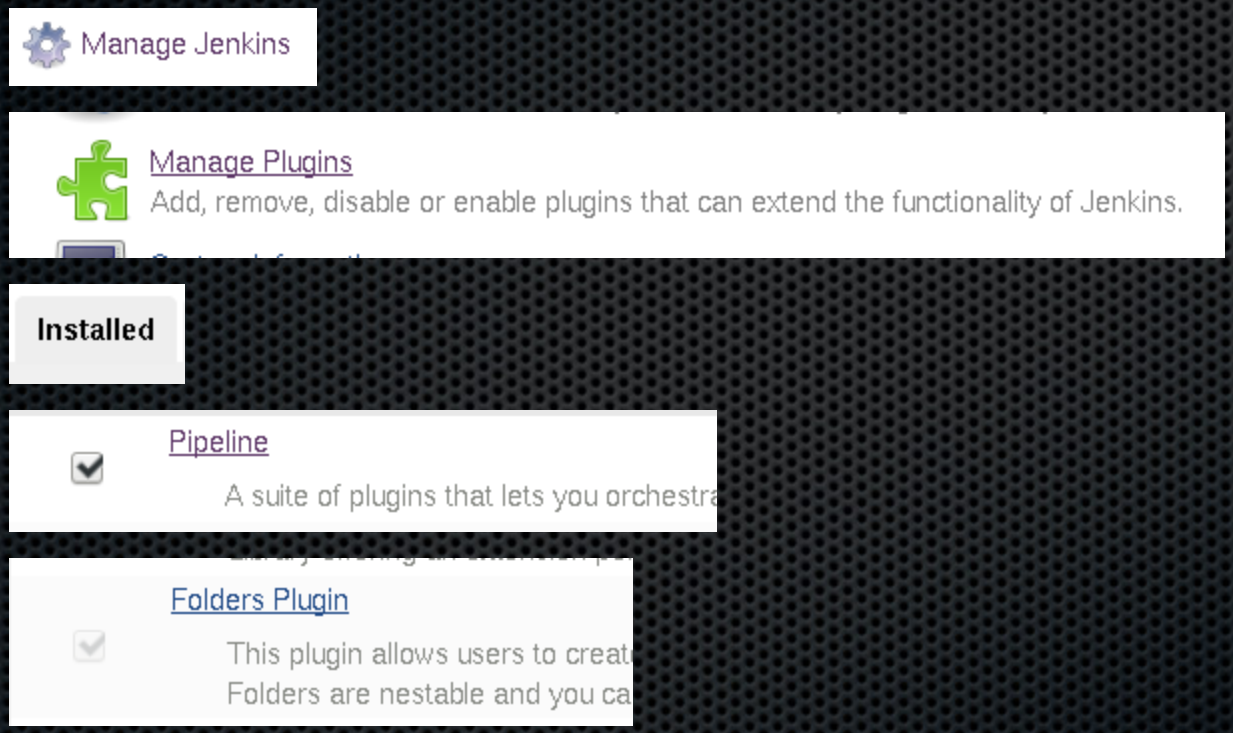
Download in Progress

Mailer Plugin	Pending
Pipeline: Basic Steps	Pending
Pipeline: Model Definition	Pending
Pipeline	Pending

Done!

Pipeline: Stage Tags Metadata	Success
Pipeline: Declarative Agent API	Success
Display URL API	Success
Mailer Plugin	Success
Pipeline: Basic Steps	Success
Pipeline: Model Definition	Success
Pipeline	Success

Check plugins installation



Raise your hand if you have problems getting the plugins installed

Create a folder

Enter an item name

python-projects

» *Required field*

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system,

Pipeline

Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is jus
long as they are in different folders.

Multibranch Pipeline

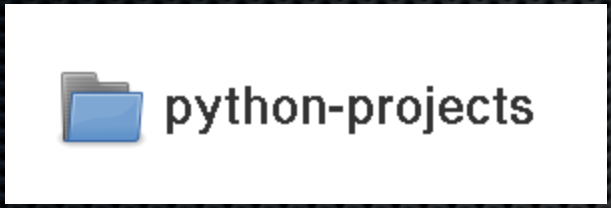
Creates a set of Pipeline projects according to detected branches in one SCM repository.

OK

Save

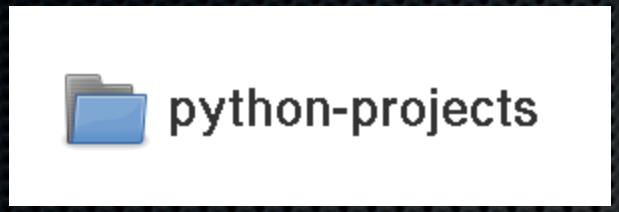
Don't do anything with Shared Libraries at that point, just save.

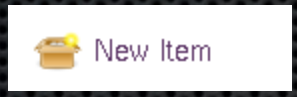
You should have the folder

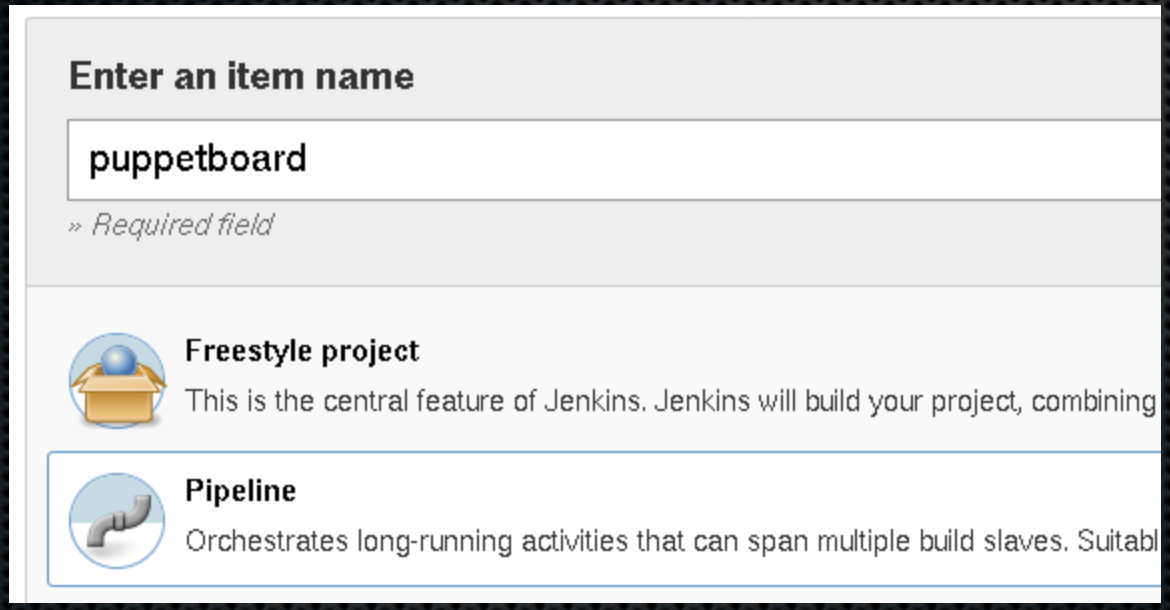


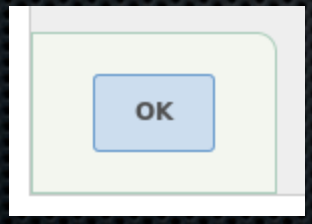
Raise your hand if you don't have the folder / have questions

Create a Pipeline Project in the folder









General | Build Triggers | Advanced Project Options | Pipeline

Pipeline name: puppetboard

Pipeline

Definition: Pipeline script

Script

1

☑ Use Groovy Sandbox

Pipeline Syntax

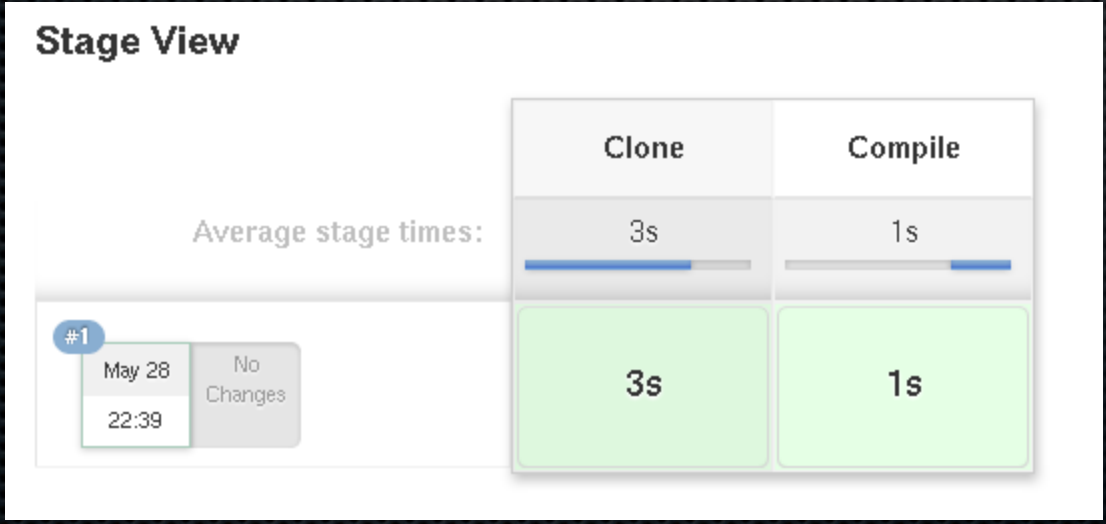
Pipeline script

```
pipeline {
 agent any
 stages {
  stage('Clone') {
   steps {
       git url: \
       'https://github.com/voxpupuli/puppetboard'
   }
  }
  stage('Compile') {
   steps {
    sh """find . -name '*.py' -print0|
    xargs -0 -L 1 python2 -m py_compile"""
   }
  }
 }
}
```

Run it!

It should run successfully

If not, raise your hand



Congratz!! We have a Python project (vaguely) tested within our Jenkins with a Declarative Pipeline

Now let's see how resources work in shared libraries

Clone your (empty) shared library git repo on your laptop

```
$ git clone git@github.com:jenkinsws/untrusted01.git
$ cd untrusted01
```

If you use Filesystem SCM, just create a empty directory

```
$ mkdir untrusted01
$ cd untrusted01
```

Create a directory "resources"

This must be the exact name

```
$ mkdir resources
```

Then subdirecties

```
$ mkdir -p resources/eu/inuits
```

What did we do?

What did we do?

In the directories resources, we created a directory for the "inuits.eu" namespace: "eu/inuits".

Namespace for "example.com" would be "com/example".

You can nest that as you like: "com/example/web/design".

That's kind of a Java thing.

Let's create a file in `resources/eu/inuits` called `python.sh` with the following content:

```
find . -name '*.py' -print0|
xargs -0 -L 1 python2 -m py_compile
```

Let's commit and push the file to the repo

```
$ git add resources/eu/inuits/python.sh
$ git commit -m 'add shared script python'
$ git push
```

you should see

```
 * [new branch]      master -> master
```

Raise your hand if you have problems

Wait, Shared Libraries need a vars or src directory

I let you decide if it's a bug or a feature

Create an empty file `vars/noop.groovy`

```
$ mkdir vars
$ touch vars/noop.groovy
```

(git only)

```
$ git add vars/noop.groovy
$ git commit -m 'add vars directory'
$ git push
```

What did we do?

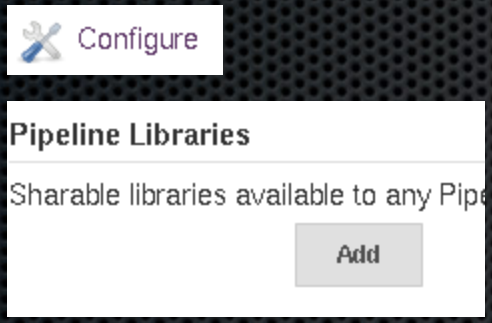
What did we do?

We created our first shared pipeline!!!

Congratulations!!!!

Let's use it now

In the folder



(git)

Library

Name: pythonhelpers

Default version: master

Cannot validate default versio

Load implicitly ☐

Allow default version to be overridden ☑

Retrieval method

◉ Modern SCM

Source Code Management

◉ Git

Project Repository: https://github.com/jenkinsws/untrusted01

Credentials: - none - | Add

Save

(filesystem)

Library

Name: pythonhelpers

Default version: master

Cannot validate default versio

Load implicitly ☐

Allow default version to be overridden ☑

Retrieval method

○ Modern SCM

◉ Legacy SCM

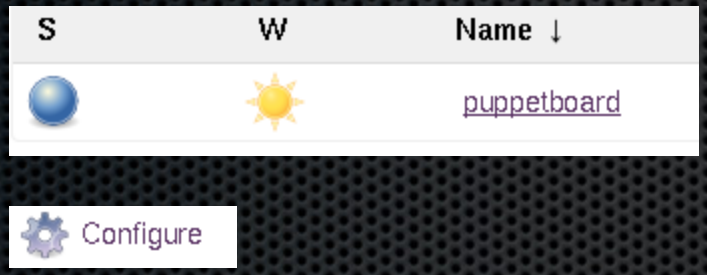
Source Code Management

◉ File System

Path: /home/roidelapluie/dev/untrusted01

Save

Open your Pipeline config



Import the library

At the beginning of the pipeline script

```
@Library('pythonhelpers') _
```

Replace

```
sh """find . -name '*.py' -print0|
xargs -0 -L 1 python2 -m py_compile"""
```

with

```
sh(libraryResource('eu/inuits/python.sh'))
```

Now, run the job

We should get the same result as before.

Job output

```
Started by user admin
Loading library pythonhelpers@master
```

```
New file: vars/noop.groovy
New file: resources/eu/inuits/python.sh
```

(filesystem SCM)

```
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/jenkinsws/untrusted01 # timeout=10
```

(git)

```
 > git config remote.origin.url https://github.com/voxpupuli/puppetboard # timeout=10
```

```
[Pipeline] sh
[workspace] Running shell script
+ find . -name '*.py' -print0
+ xargs -0 -L 1 python2 -m py_compile
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Raise your hand if you don't have that

We have used our first shared resource!! Congratz!!

Let's update our library

Change `resources/eu/inuits/python.sh`

```
find . -name '*.py' -print0|
xargs -0 -L 1 python2 -m py_compile
```

by

```
find . -name '*.py' -print0|
xargs -0 -t -L 1 python2 -m py_compile
```

(add -t in the second line)

Push that change

```
$ git add resources/eu/inuits/python.sh
$ git commit -m 'make xargs more verbose'
$ git push
```

Run the script again

Before:

```
[Pipeline] sh
[workspace] Running shell script
+ find . -name '*.py' -print0
+ xargs -0 -L 1 python2 -m py_compile
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Now you should see

Changes

1. make xargs more verbose (detail / githubweb)

```
[workspace] Running shell script
+ find . -name '*.py' -print0
+ xargs -0 -t -L 1 python2 -m py_compile
python2 -m py_compile ./dev.py
python2 -m py_compile ./wsgi.py
python2 -m py_compile ./setup.py
python2 -m py_compile ./test/test_form.py
python2 -m py_compile ./test/test_app_error.py
python2 -m py_compile ./test/test_utils.py
python2 -m py_compile ./test/test_docker_settings.py
python2 -m py_compile ./test/test_app.py
```

Change in our script has been integrated in the build!! Imagine changing the script once applies to all the pipelines!

Raise your hand if you have problems

let's go for something more powerful

In the shared library git repo, create a file `vars/compilePython.groovy` with

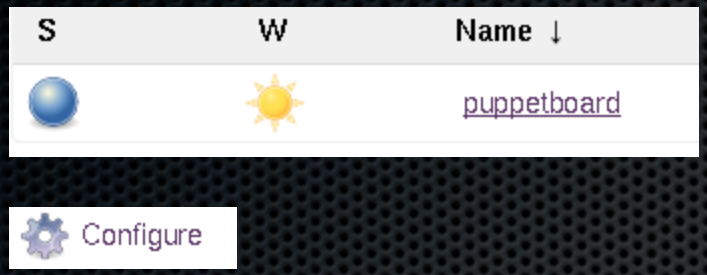
```
def call() {
    sh """find . -name '*.py' -print0|
    xargs -0 -t -L 1 python2 -m py_compile"""
}
```

Note: you could reuse the libraryResource here.

Push the file

```
$ git add vars/compilePython.groovy
$ git commit -m 'add a compilePython step'
$ git push
```

In the Pipeline script



Replace

```
sh(libraryResource('eu/inuits/python.sh'))
```

with

```
compilePython()
```

Run, it should work!

Raise your hand if you need help

Congratz!! Our first shared step!

Let's add LOGIC

In the shared library git repo, change

`vars/compilePython.groovy`

```
def call(){
 if (fileExists("requirements.txt")){
 sh "virtualenv venv"
 sh "venv/bin/pip install -r requirements.txt"
 }
 sh """find . -name '*.py' -print0|
 xargs -0 -t -L 1 python2 -m py_compile"""
}
```

Push the file

```
$ git add vars/compilePython.groovy
$ git commit -m 'add vitualenv support'
$ git push
```

Run the build ; should fail

Let's add a parameter

In the shared library git repo, change

`vars/compilePython.groovy`

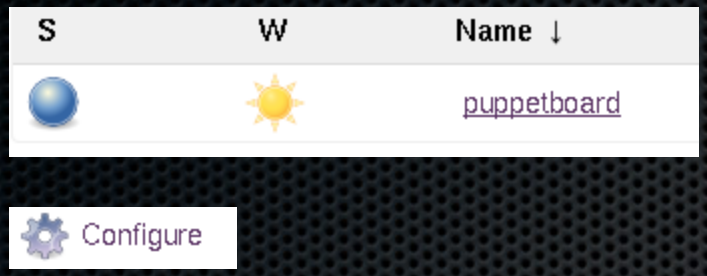
```
def call(String directory = '.') {
 sh """find ${directory} -name '*.py' -print0|
 xargs -0 -t -L 1 python2 -m py_compile"""
}
```

Push the file

```
$ git add vars/compilePython.groovy
$ git commit -m 'add compilePython parameter'
$ git push
```

Run the Pipeline again, it should fail as before

In the Pipeline script



Replace

```
compilePython()
```

with

```
compilePython("puppetboard")
compilePython("test")
```

Run, it should work!

See the two steps

```
+ find puppetboard -name '*.py' -print0
+ xargs -0 -t -L 1 python2 -m py_compile
python2 -m py_compile puppetboard/forms.py
python2 -m py_compile puppetboard/dailychart.py
python2 -m py_compile puppetboard/version.py
python2 -m py_compile puppetboard/app.py
python2 -m py_compile puppetboard/__init__.py
python2 -m py_compile puppetboard/docker_settings.py
python2 -m py_compile puppetboard/utils.py
python2 -m py_compile puppetboard/default_settings.py
[Pipeline] sh
[workspace] Running shell script
+ xargs -0 -t -L 1 python2 -m py_compile
+ find test -name '*.py' -print0
python2 -m py_compile test/test_form.py
python2 -m py_compile test/test_app_error.py
```

Documentation

In the pipeline

S	W	Name ↓
		puppetboard

Pipeline Syntax

Global Variables Reference

compilePython

In shared lib repo create

`vars/compilePython.txt`

```
This step compiles python files in the given
directory
```

Push the file

```
$ git add vars/compilePython.txt
$ git commit -m 'add compilePython doc'
$ git push
```

Build once and check the doc again

S	W	Name ↓
		puppetboard

Pipeline Syntax

Global Variables Reference

compilePython

This step compiles python files in the given directory

\o/ Raise your hands if you have questions

Note

You can remove our noop.groovy in the shared library

```
$ git rm vars/noop.groovy
$ git commit -m 'remove noop.groovy'
$ git push
```

Let's do something awesome now.

Remember that Declarative Pipeline?

We can do this as a "shared step".

Create `vars/pythonPipeline.groovy`

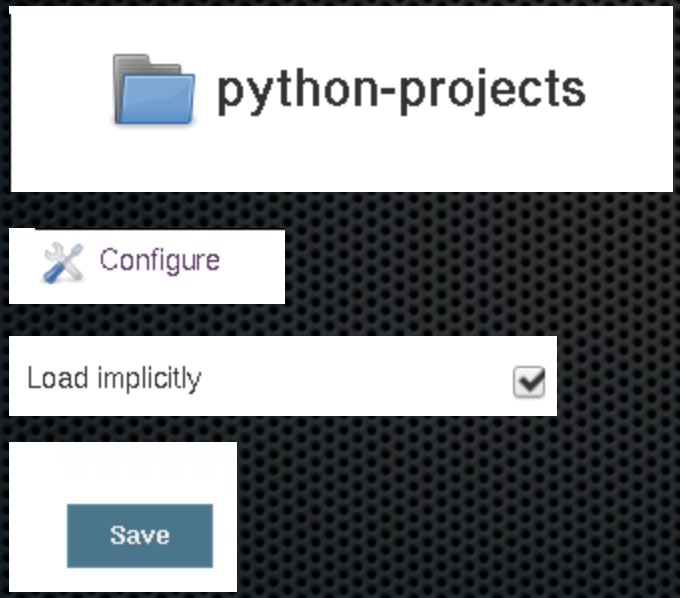
```
def call(String githubproject){
 pipeline {
  agent any
  stages {
   stage('Clone') {
    steps {
       git url: \
       "https://github.com/${githubproject}"
    }
   }
   stage('Compile') {
    steps{
     sh("ls")
    }
   }
  }
 }
}
```

Add, commit and push

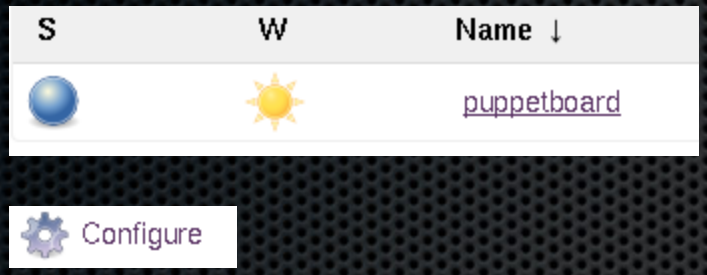
```
$ git add vars/pythonPipeline.groovy
$ git commit -m 'add a Python Pipeline step'
$ git push
```

Make the library import implicit

That way you do not need to import it in your Pipeline script.



Change the Pipeline




```
pythonPipeline("voxpupuli/puppetboard")
```

Yes, that's all

Save & run

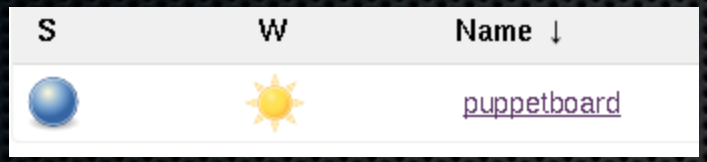
What happened?

- We have put the COMPLETE declarative pipeline in a shared lib
- The jenkins file is really small
- It is parametrizable
- Change to the shared lib can change a lot of pipelines!

Improvement: groovy closure `vars/pythonPipeline.groovy`

```
def call(body) {
 def config = [:]
 body.resolveStrategy = Closure.DELEGATE_FIRST
 body.delegate = config
 body()
 pipeline {
  agent any
  stages {
   stage('Clone') {
    steps {
      git url: \
        "https://github.com/${config.githubproject}"
    }
   }
   stage('Compile') {
    steps{
     sh("ls")
    }
   }
  }
 }
}
```

Change the Pipeline




```
pythonPipeline {
  githubproject = "voxpupuli/puppetboard"
}
```

Save & run

Let's get groovy

We will now recreate the same step but as "plain groovy".
The real power will come later with "trusted" library.

Reset the Pipeline script

Now we will edit the job's Pipeline and revert our latest change. Here is the new script (next slide).

```
pipeline {
 agent any
 stages {
  stage('Clone') {
   steps {
    git url: \
     "https://github.com/voxpupuli/puppetboard"
   }
  }
  stage('Compile') {
   steps{
    compilePython("puppetboard")
    compilePython("test")
   }
  }
 }
}
```

Save and run

In Pipeline dir, create

`src/eu/inuits`

```
$ mkdir -p src/eu/inuits
```

once again, "eu/inuits" means "inuits.eu"

Create

src/eu/inuits/PythonCompiler.groovy

```
package eu.inuits
class PythonCompiler {
 static def compileDirectory(script, directory) {
  script.sh """find ${directory} -name '*.py' \
  -print0|
 xargs -0 -t -L 1 python2 -m py_compile"""
 }
}
```

Change `vars/compilePython.groovy`

```
import static eu.inuits.PythonCompiler.*

def call(String directory = '.') {
    echo("Compiling ${directory}")
    compileDirectory(this, directory)
}
```

Push the files

```
$ git add src/eu/inuits/PythonCompiler.groovy
$ git add vars/compilePython.groovy
$ git commit -m 'add PythonCompiler class'
$ git push
```

Run, it should work :-)

```
[Pipeline] echo
Compiling puppetboard
[Pipeline] sh
[workspace] Running shel
+ find puppetboard -name
+ xargs -0 -t -L 1 pytho
python2 -m py_compile pu
python2 -m py_compile pu
python2 -m py_compile pu
python2 -m py_compile pu
python2 -m py_compile pu
python2 -m py_compile pu
python2 -m py_compile pu
python2 -m py_compile pu
[Pipeline] echo
Compiling test
[Pipeline] sh
```

Raise your hand if it does not

What happened?

We created a plain groovy library and we called it from within a (shared) step

If you do not use Declarative Pipeline you can do it in your Jenkinsfile directly

Keeping State

Create a `vars/compileFile.groovy` in your shared lib.

```
import eu.inuits.FileCompiler

def call(String project) {
    fc = new FileCompiler(this, project)
    fc.analyze('requirements.txt')
    fc.analyze('setup.py')
}
```

Implement it!

src/eu/inuits/FileCompiler.groovy

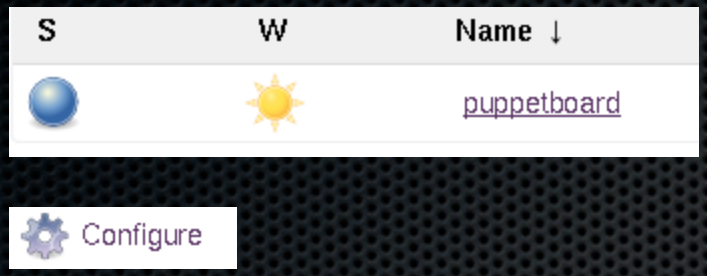
```
package eu.inuits
class FileCompiler implements Serializable {
  def script
  def project
  FileCompiler(script, String project){
    this.script = script
    this.project = project
    this.script.echo("Compiling ${project}")
  }
  def analyze(String file) {
    this.script.echo("${project}/${file}")
    this.script.sh("cat ${file}")
  }
}
```

Add and push

(you know the drill)

```
$ git add src/eu/inuits/FileCompiler.groovy
$ git add vars/compileFile.groovy
$ git commit -m "Add compileFile step"
$ git push
```

In the Pipeline script



Replace

```
compilePython("puppetboard")
compilePython("test")
```

with

```
compileFile("puppetboard")
```

Run it

ooops

```
[Pipeline] End of Pipeline
org.jenkinsci.plugins.scriptsecurity.sandbox.RejectedAccessException: unclassified new eu.inuits.FileCompiler java.lang.String
        at org.jenkinsci.plugins.scriptsecurity.sandbox.groovy.SandboxInterceptor.onNewInstance(SandboxInterceptor.java:126)
        at org.kohsuke.groovy.sandbox.impl.Checker$3.call(Checker.java:191)
        at org.kohsuke.groovy.sandbox.impl.Checker.checkedConstructor(Checker.java:188)
        at com.cloudbees.groovy.cps.sandbox.SandboxInvoker.constructorCall(SandboxInvoker.java:20)
```

We hit the security sandbox

In the Pipeline script

S	W	Name ↓
		puppetboard

Configure

```
16    }
17    stage('Compile') {
```

Use Groovy Sandbox

Save

& run again! Should work!

```
[Pipeline] echo
Compiling puppetboard
[Pipeline] echo
puppetboard/requirements.txt
[Pipeline] sh
[workspace] Running shell script
+ cat requirements.txt
Flask >=0.12
Flask-WTF >=0.14.2
Jinja2 >=2.9.5
```

What happened? We have a "state" (variable project). It can be changed on the fly in the class, etc...

Are you still following?

Global Trusted Library

Now let's look into another kind of global library: the Trusted Libraries.

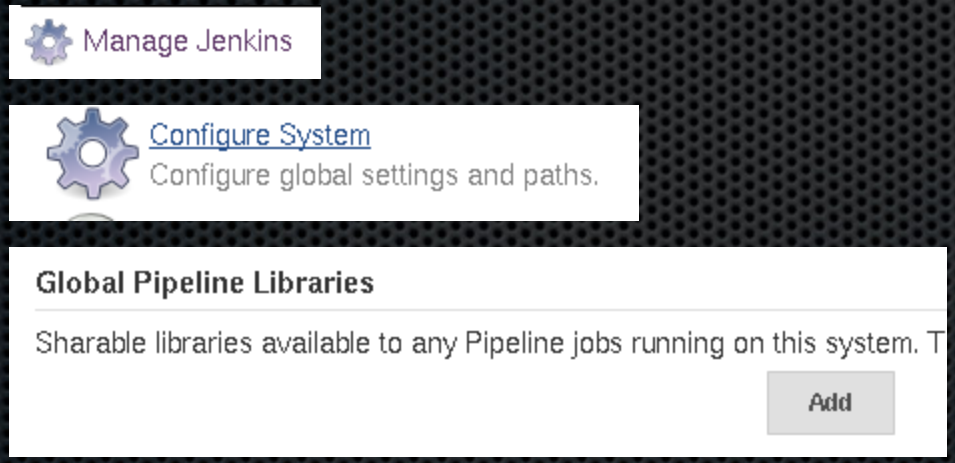
Those libraries are defined in the global config, and run unsandboxed.

Clone a new git repo for your trusted library

```
$ git clone git@github.com:jenkinsws/trusted01.git
$ cd trusted01
```

Add the Library as a Global Shared Library

Same as we did before but in the global config



Library

Name: globalhelp

Default version: master

Cannot validate de

Load implicitly ☐

Allow default version to be overridden ☐

Retrieval method

◉ Modern SCM

Source Code Management

◉ Git

Project Repository: https://github.com/jenkinsws/trusted01

Credentials: - none - | Add

Save

Global Libraries

- Used with caution: can do everything
- Unsandboxed

Let's go!

create `src/eu/inuits/Admin.groovy`

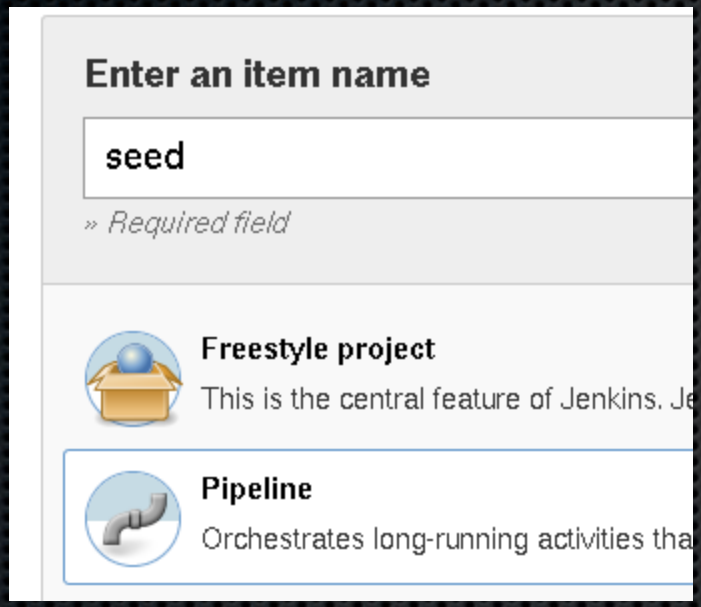
```
package eu.inuits
import com.cloudbees.groovy.cps.NonCPS
class Admin implements Serializable {
 def seedFullName = "seed"
 def script

 Admin(script) {
  this.currentJobValidation(script)
  this.script = script
 }

 @NonCPS
 void currentJobValidation(script){
  def name = \
    script.currentBuild.rawBuild.project.fullName
  assert name == this.seedFullName : "DENIED"
 }
}
```

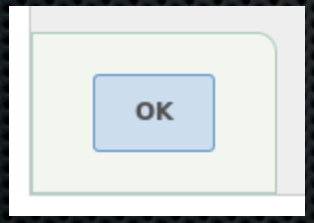
Add and commit file

```
$ git add src/eu/inuits/Admin.groovy
$ git commit -m 'add admin lib'
$ git push
```

Raise your hand if you need help!

Create a new job in the top





Set the following Pipeline Script

```
@Library('globalhelp') _
import eu.inuits.Admin

node {
    adm = new Admin(this)
}
```

And save.

Run the Pipeline

```
Started by user admin
Loading library globalhelp@master
 > git rev-parse --is-inside-work-tree
```

```
  [Pipeline] }
  [Pipeline] // node
  [Pipeline] End of Pipeline
  Finished: SUCCESS
```

Rename the job

Jenkins ▸ seed ▸

Back to Project

Configure

Pipeline name seed2

Save

Are you sure about renaming seed to seed2? Yes

Pipeline seed2

Run the renamed job

Build Now

```
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
java.lang.AssertionError: DENIED. Expression: (name == this.seedFullName). Values: name = seed2
        at org.codehaus.groovy.runtime.InvokerHelper.assertFailed(InvokerHelper.java:404)
        at org.codehaus.groovy.runtime.ScriptBytecodeAdapter.assertFailed(ScriptBytecodeAdapter.
```

Our class can only be called from within a job named seed.

Rename the job back

Jenkins ▸ seed2 ▸

Back to Project

Configure

Pipeline name seed

Save

Are you sure about renaming seed2 to seed? Yes

Pipeline seed

Who needs help at this point?

Add a function to the trusted lib

src/eu/inuits/Admin.groovy

after Package, before Class

```
import jenkins.model.Jenkins
```

then, in body

```
/**
  * Set the description of a folder
  *
  * @param folder A jenkins project name
  * @param description New description
  */
@NonCPS
void setFolderDescription(folder, description) {
    def f = Jenkins.instance.getItemByFullName(folder)
    f.setDescription(description)
}
```

Add, Commit and Push

```
$ git add src/eu/inuits/Admin.groovy
$ git commit -m 'add setFolderDescription'
$ git push
```

Change seed job Pipeline Script

```
@Library('globalhelp') _
import eu.inuits.Admin

node {
  adm = new Admin(this)
  adm.setFolderDescription("python-projects",
  "Description set withing Global Pipeline")
}
```

Save and Run

Check the result

python-projects

Description set withing Global Pipeline

All +

S	W	Name ↓
		puppetboard

Raise your hand if you have a different result

@Grab

Just an example from

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

```
@Grab('org.apache.commons:commons-math3:3.4.1')
import org.apache.commons.math3.primes.Primes
void parallelize(int count) {
  if (!Primes.isPrime(count)) {
    error "${count} was not prime"
  }
  // Ã¢â‚¬Â¦
}
```

Global Libs

- Do not require sandbox escape at the job level
- No sandbox exceptions needed
- @Grab
- Freedom!

Advice for safe global lib

- Check what you return: void, string.. do not return objects like Jenkins Job
- Check the name of the job
- No "implicit loading"
- No "version overwrite"

Conclusions

- Shared Libraries keep your Jenkinsfiles small
- They allow you to scale your Pipelines
- Change the build process without comitting to all the repos
- No need for tools like modulesync etc...

For this talk, thanks Jenkins, Praqma, Inuits.

Used for this talk: DigitalOcean, github, packer, consul, terraform, traefik, let's encrypt, gibby/letsencrypt-dns-digitalocean.

Contact

Julien Pivotto
roidelapluie
roidelapluie@inuits.eu

Inuits
https://inuits.eu
info@inuits.eu