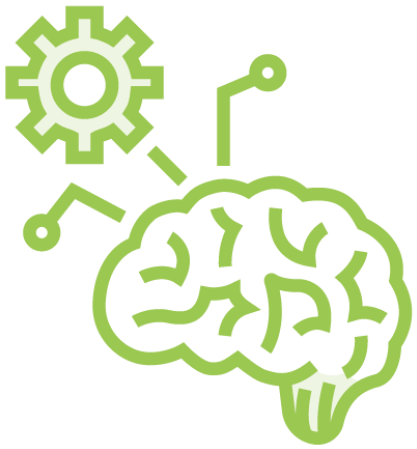
Working with Shared Libraries

Chris B. Behrens
SOFTWARE ARCHITECT

@chrisbbehrens

Moving Ahead



Reusing Groovy Code in Jenkins

Why bother with these advanced language features?

The more complex you get, the deeper you go in the stack

Global Pipeline Libraries

GPL repo must be secure

Our demo repo will be public

Our Demo Scenario

Next step after build is to push to a deployment server

QA can review the history to troubleshoot problems

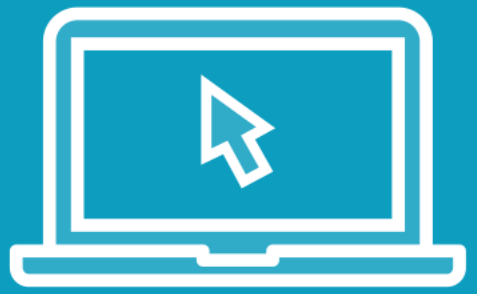
QA doesn't need to log into Jenkins

Create a releasenotes.txt

A shared library to generate it

- List of files
  - Name
  - Size

Demo



- Develop a Groovy script
- Lists directory contents
- Persists that information to a releasenotes.txt
- Commit that script to version control
- Configure the library in Jenkins
- Execute the library in our pipeline script
- Find a problem

Groovy CPS

Jenkins Pipeline uses a library called Groovy CPS to run Pipeline scripts. While Pipeline uses the Groovy parser and compiler, unlike a regular Groovy environment it runs most of the program inside a special interpreter.

Groovy CPS

This uses a Continuation-Passing Style (CPS) transform to turn your code into a version that can save its current state to disk and continue running even after Jenkins has restarted. While the CPS transform is usually transparent to users, there are limitations to what Groovy language constructs can be supported, and in some circumstances it can lead to counterintuitive behavior.

```
new File(dir.path + '/releasenotes.txt').withWriter('utf-8')
{
    writer ->
    dir.eachFileRecurse(FileType.ANY){ file ->
…
    }
}
```

Jenkins and CPS

Script objects are serialized

To program.dat

Not all objects can be serialized - yet

```
@NonCPS
new File(dir.path + '/releasenotes.txt').withWriter('utf-8')
{
    writer ->
    dir.eachFileRecurse(FileType.ANY){ file ->
…
    }
}
```

Getting Our Changes into Our Release Notes

Releasenotes.txt gets packaged up with our artifacts

QA needs the *revision data* for the commit

With this, they can contact the developer directly

Additionally:

- Date
- Time
- Build #
- Because that's what the dev's always ask for

Getting Our Changes into Our Release Notes

Just pipe changes.log into the file

Too easy

Let’s do things the hard way

Demo

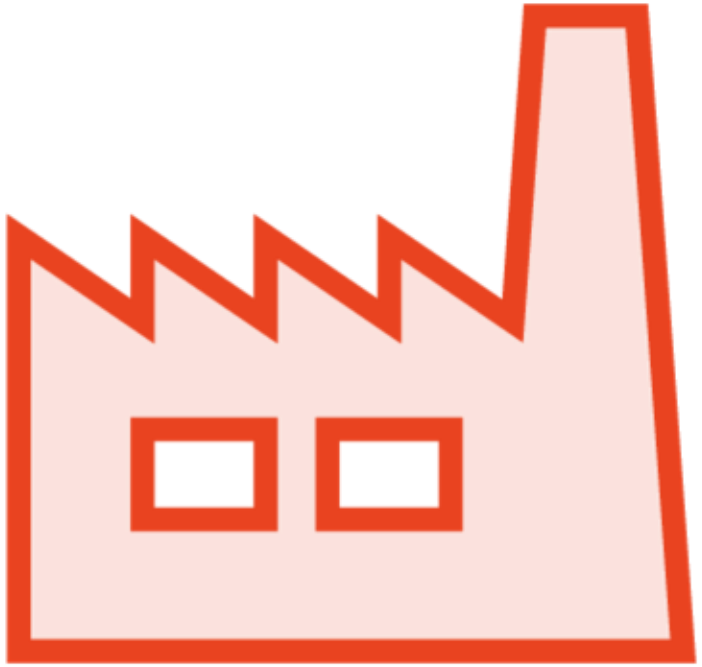
Three new features to our release notes

- Date and time
- Build number
- Our changeset details

The SCM libraries

- Work with source code management

Shared Pipelines



All your builds have the same phases

- SCM
- Build
- Test
- Deploy

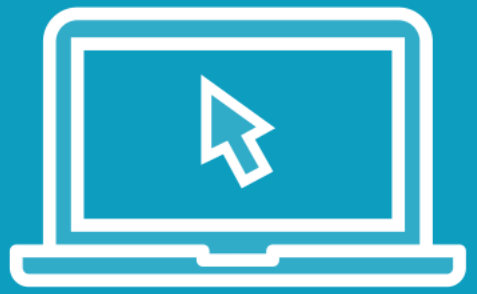
Implementing shared pipelines for two reasons

- To save time
- To enforce conventions

The only particular: ConsoleApp1

A shared pipeline is just another shared script

Demo



Our existing Jenkinsfile pipeline

Copy it to our SharedFunction repo

Modify it to reflect the conventions of shared libraries in Jenkins

Configure it as a shared library in Jenkins

Modify our original Jenkinsfile to call it

Using a single line

Wrap-up: Working with Shared Libraries

Reusing code is fundamental

Reusing pipelines enforces standards

Make the script as smart as it needs to be

Managing Plug-ins, Users and Credentials

Chris B. Behrens
SOFTWARE ARCHITECT

@chrisbbehrens

Working with Plug-ins

Startup scripts enforce configuration

Especially important with containers

Use scripts to enforce plug-in state

Demo

Modify our sample plug-in script

List out our plug-ins in a more readable way

Verify that a particular plug-in is installed

- By title
- By version

Installs the plug-in

A Plug-in Script Pattern

Jenkins has a lot of default plug-ins

- Each of those has an impact on your system

A plug-in whitelist

Disable those *not* on the whitelist

Install those not in the installed set

Demo

Load a simple plug-in whitelist

Load the set of installed plug-ins

Match them against the whitelist

The set of plug-ins not in the whitelist but installed

Disable those

The set of plug-ins in the whitelist but not installed

Install them

And Now, a Warning

Creating a whitelist is tricky business

Because plug-ins depend on each other

Disable a low-level one, and potentially disable a lot of others

My whitelist is NOT PERFECT

Take care in creating your whitelist

- Test by creating a complete list and removing one at a time

https://github.com/samrocketman/jenkins-script-console-scripts/blob/master/safe-restart.groovy

Working with Credentials and Users

A user has a logon and permissions into Jenkins

Credentials are used by Jenkins to access something *outside*

Demo

Lists out the users

Simple script to create a user

Delete a user

Talk about our whitelist pattern

Create a script to enforce it

Working with Credentials

Elements Jenkins uses to authenticate itself

- To something else

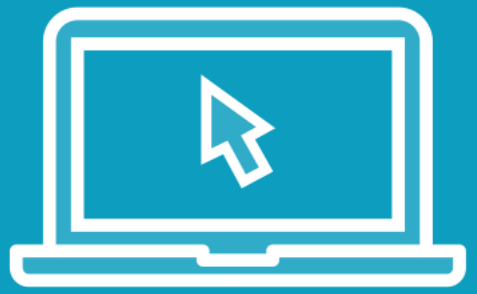
Credentials plug-in

- Hope we didn't disable it

Before we script

- How Jenkins works with Credentials in pipelines

Demo



Create a set of credentials

To login to GitHub

Access a protected repository

How to inject those credentials into a groovy pipeline

Verify that our pipeline was able to pull from our repo

Creating Credentials in Script

Two things we know

- What credentials are used for
- How they're used in the build

We need to change the password for our creds

In script

Demo

Create a CredentialManager

Lists all the credentials in the system

Change the password for a specified credential set

Course Summary

Whew!

New concepts folded in gradually

We could keep going, but you get the picture

If you've worked with C#, Java, or JavaScript...

- You know Groovy

If you're still working with classic builds

- Move to pipelines as quickly as you can

THANK YOU VERY MUCH
FOR WATCHING!!!