

WASHINGTON DC JAM

CODIFY THE BUILD AND RELEASE PROCESS WITH PIPELINE
SHARED LIBRARIES AND DOCKER

Presented by Alvin Huang, DevOps Engineer

COPYRIGHT © 2017, FIREEYE, INC. ALL RIGHTS RESERVED.



Who Am I

- DevOps Engineer, GSI @ FireEye
- Twitter: @RealAlvinHuang, #jenkins IRC: ahuang
- alvin.huang@fireeye.com



Tonight's Agenda

- GSI Overview
- What is Pipeline?
- The Great Migration; Why Pipeline? Why Docker?
- How we did this
- Lessons Learned
- Benefits for the Future

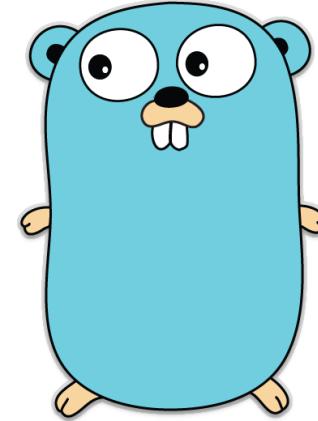
GSI Overview

- Unified global organization combining Mandiant, FireEye iSIGHT Intelligence, and FireEye as a Service operations
- **DevOps Goal:** Make the feedback loop between intelligence on the front lines to capabilities in software, smaller

What We Build



django



Java™

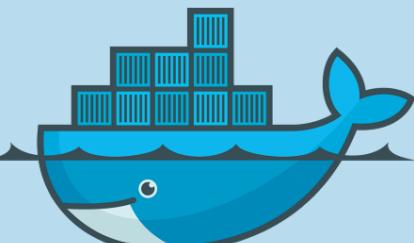


How We Build

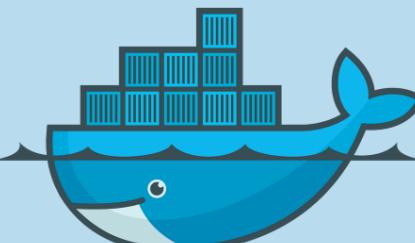
Code



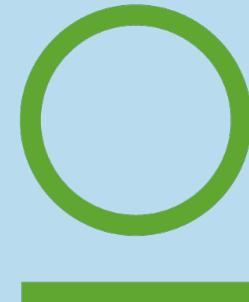
Build



Test

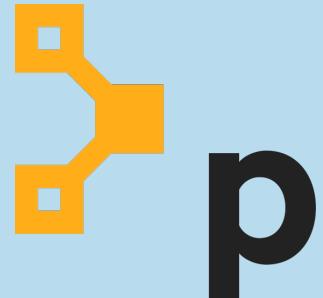


Deliver



yum
yellowdog updater modified

Deploy



What is Jenkins Pipeline?

- Pipeline as Code - Jenkins DSL to help users model, orchestrate, and visualize entire delivery pipelines
- Pipeline Stage View
- Durability – can survive planned and unplanned Jenkins master outages
- Built in support for Git/Github branches

Jenkinsfile (Declarative Pipeline)

```
pipeline {  
    agent { docker 'python:3.5.1' }  
    stages {  
        stage('build') {  
            steps {  
                sh 'python --version'  
            }  
        }  
    }  
}
```

Jenkinsfile (Scripted Pipeline)

```
/* Requires the Docker Pipeline plugin */  
node('docker') {  
    checkout scm  
    stage('Build') {  
        docker.image('python:3.5.1').inside {  
            sh 'python --version'  
        }  
    }  
}
```

THE GREAT MIGRATION

Scenario

- Consolidate datacenters
- Migrate Jenkins off a shared instance
- ~60 days
- ~150 Freestyle jobs
- ~20 different build agents
- Upgrade from 1.x to 2.x



Legacy Jenkins Jobs

- Jenkins primarily used to run tests and build RPMs
- SCP to publish RPMs to yum server
- Ad-hoc Github triggers
- No owner/SME for Jenkins

Goals

- Standardize and make consistent CI pipelines
- Jenkins SME = DevOps team
- Make new app onboarding easy
 - Build abstraction that covers ~90% of jobs
- Jenkins jobs tracked in git
- Decouple tools from Jenkins/shell => Docker

Migration Strategies

Migration Strategy	LOE	Advantages	Disadvantages
Copy XMLs from old server to new server	Minimal	- Easy, no extra work	- Tech debt - Global changes are hard - Job config not in SCM
Create Freestyle Jenkins job templates	Moderate	- Global changes are easier - Hide much of the configuration from end users - Make Freestyle jobs DRY	- Tech debt - Debugging a job is hard - Job config not in SCM - Global changes can break all builds
Convert Freestyle jobs to Pipeline jobs	Significant	- Job configuration in SCM	- Learn pipeline syntax - Large job footprint - Global changes are hard - Too complex for many users - Subject to repetition and tech debt
Create a Jenkins Shared Library for Pipeline jobs	Moderate	- Jobs in SCM - Standardized and consistent - Easy on-boarding, just needs parameters - Job configuration hidden from user	- Learn pipeline syntax - Need owner for library - Global changes can break all builds

Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	
Jobs were not DRY	

Solution: Build Wrapper

```
standardBuild {  
    machine          = 'docker'  
    dev_branch      = 'develop'  
    release_branch  = 'master'  
    artifact_apttern = '*.rpm'  
    html_pattern    = [keepAll: true, reportDir: '.', reportFiles: 'output.html', reportName: 'OutputReport']  
    dev_repo         = 'pipeline-examples-dev'  
    prod_repo        = 'pipeline-examples-prod'  
    pr_script        = 'make prs'  
    dev_script       = 'make dev'  
    release_script   = 'make release'  
}
```

Solution: Use Github Organizations

- Index every branch of every repo that has a Jenkinsfile
- Configure one Github hook on the organization level

The screenshot shows the Jenkins web interface. In the top left, there's a Jenkins logo and the word "Jenkins". To its right is a search bar with a magnifying glass icon. On the far right of the header is a "Logout" link. Below the header, the URL "Jenkins > alvin-huang-jenkinsworld-org" is visible. On the left side, there's a sidebar with various links: "Up", "Status", "Configure", "Scan Organization Now", "Scan Organization Log", "Organization Events", "Delete Organization", "People", "Build History", "GitHub", "Pipeline Syntax", and "Credentials". The main content area is titled "alvin-huang-jenkinsworld-org". Below that, it says "Repositories (4)". A table lists four repositories: "testrepo1", "testrepo2", "testrepo3", and "testrepo4". Each repository entry includes a small icon, a status indicator (yellow sun), and a link to the repository details. At the bottom of the table, it says "Icon: S M L". To the right of the table, there are "Legend" and "RSS" links.

S	W	Name ↓	Description
		testrepo1	
		testrepo2	
		testrepo3	
		testrepo4	

Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper

Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	

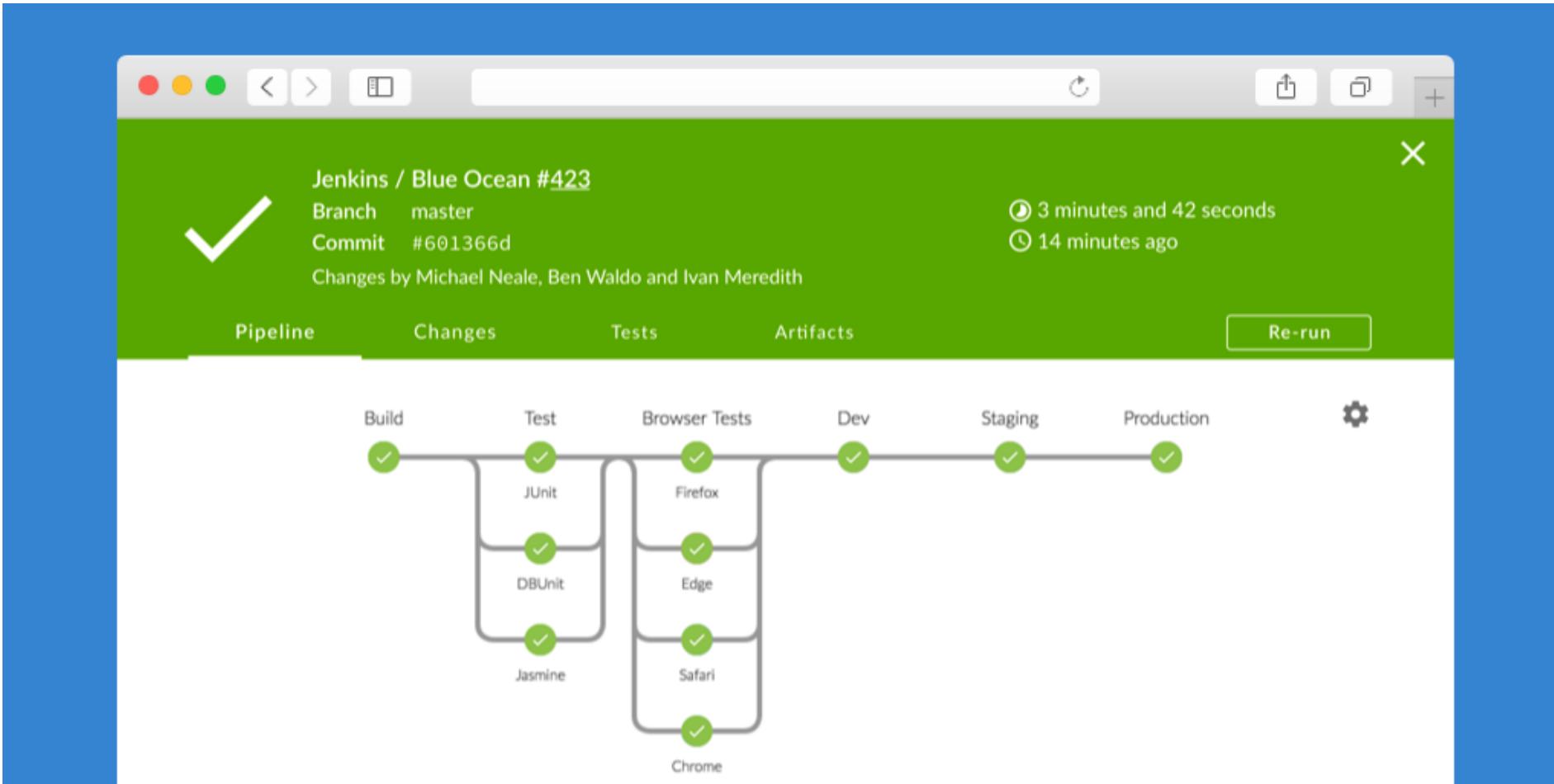
Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	Jenkinsfiles in SCM
Difficult to view job configurations and build steps	

Solution



Solution



Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	Jenkinsfiles in SCM
Difficult to view job configurations and build steps	Jenkinsfiles + Stage View

Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	Jenkinsfiles in SCM
Difficult to view job configurations and build steps	Jenkinsfiles + Stage View
Very difficult to make global changes	

Solution

```
def call(config) {  
    stepCheckout()  
    stepBuild()  
    ....  
}
```



```
def call(config) {  
    stepCheckout()  
    if(config.machine == 'docker') {  
        stepDockerfileLint()  
    }  
    stepBuild()  
}
```

yourBuildWrapper.groovy

Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	Jenkinsfiles in SCM
Difficult to view job configurations and build steps	Jenkinsfiles + Stage View
Very difficult to make global changes	

Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	Jenkinsfiles in SCM
Difficult to view job configurations and build steps	Jenkinsfiles + Stage View
Very difficult to make global changes	*Library changes have an effect on all builds

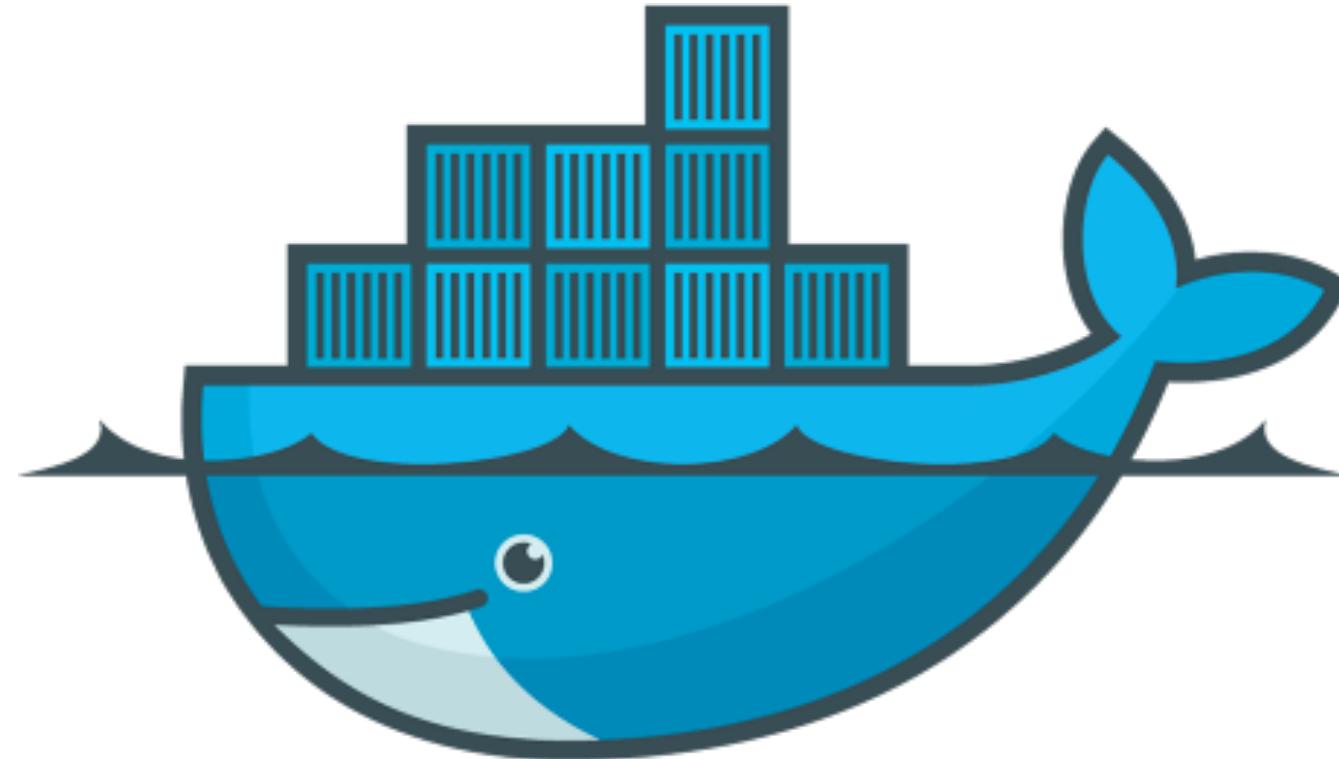
Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	Jenkinsfiles in SCM
Difficult to view job configurations and build steps	Jenkinsfiles + Stage View
Very difficult to make global changes	*Library changes have an effect on all builds
Monolithic Build Agents	

Migrating Monolithic Build Agents

- No configuration management
- What packages and versions were installed?
- Who runs on what agent?
- Changes on a node may affect another team sharing that same agent

Solution to Monolithic Build Agents



Sample Dockerfile

```
FROM centos:centos7

# install Puppet5 repo
RUN yum install -y https://yum.puppetlabs.com/puppet5/puppet5-release-el-6.noarch.rpm

RUN yum install -y puppet-agent \
    rubygems && \
    yum clean all

ENV PATH="/opt/puppetlabs/puppet/bin:$PATH"

RUN gem install puppet-lint \
    puppet-syntax \
    yaml-lint && \
    gem cleanup all

WORKDIR "/code"
```

```
FROM faas/el7-python:base

RUN yum -y install \
    czmq-devel \
    gcc-c++ \
    python-virtualenv \
    python-pip \
    prelink \
    postgresql-devel \
    mysql-devel \
    libpcap-devel \
    libffi-devel \
    rpm-build \
    git && \
    yum clean all

RUN pip install fabric
```

Why Pipeline and Shared Libraries?

Problem	Solution
Overhead with Freestyle Jobs	Use a pipeline library wrapper; Github Organizations
Jobs were not DRY	Use a pipeline library wrapper
Job configuration drift	Jenkinsfiles in SCM
Difficult to view job configurations and build steps	Jenkinsfiles + Stage View
Very difficult to make global changes	*Library changes have an effect on all builds
Monolithic Build Agents	Docker

What about...

- Jenkins Job Builder
 - Simple jobs defined by relatively complex YAML
 - Verbose
 - Needs abstraction for novice users
- Job DSL Plugin
 - Programmatically seeds/creates Jenkins jobs
 - Jenkins Pipeline + Github Organizations can be used instead
 - No logic between jobs (ie: no Pipeline ‘input’ step)

Starting Point

- New Jenkins 2.x instance
- Stable build nodes built with Puppet
 - CentOS 6/7 hosts
- Working Docker integration

BEFORE PIPELINE

Phase 1 (Pre-migration)

						add description
S	W	Name ↓	Last Success	Last Failure	Last Duration	
		app1-develop	N/A	N/A	N/A	
		app1-prod	N/A	N/A	N/A	
		app2-develop	N/A	N/A	N/A	
		component1	N/A	N/A	N/A	
		component2-develop	N/A	N/A	N/A	
		free1	N/A	N/A	N/A	
		free2	N/A	N/A	N/A	
		free3	N/A	N/A	N/A	
		free4	N/A	N/A	N/A	

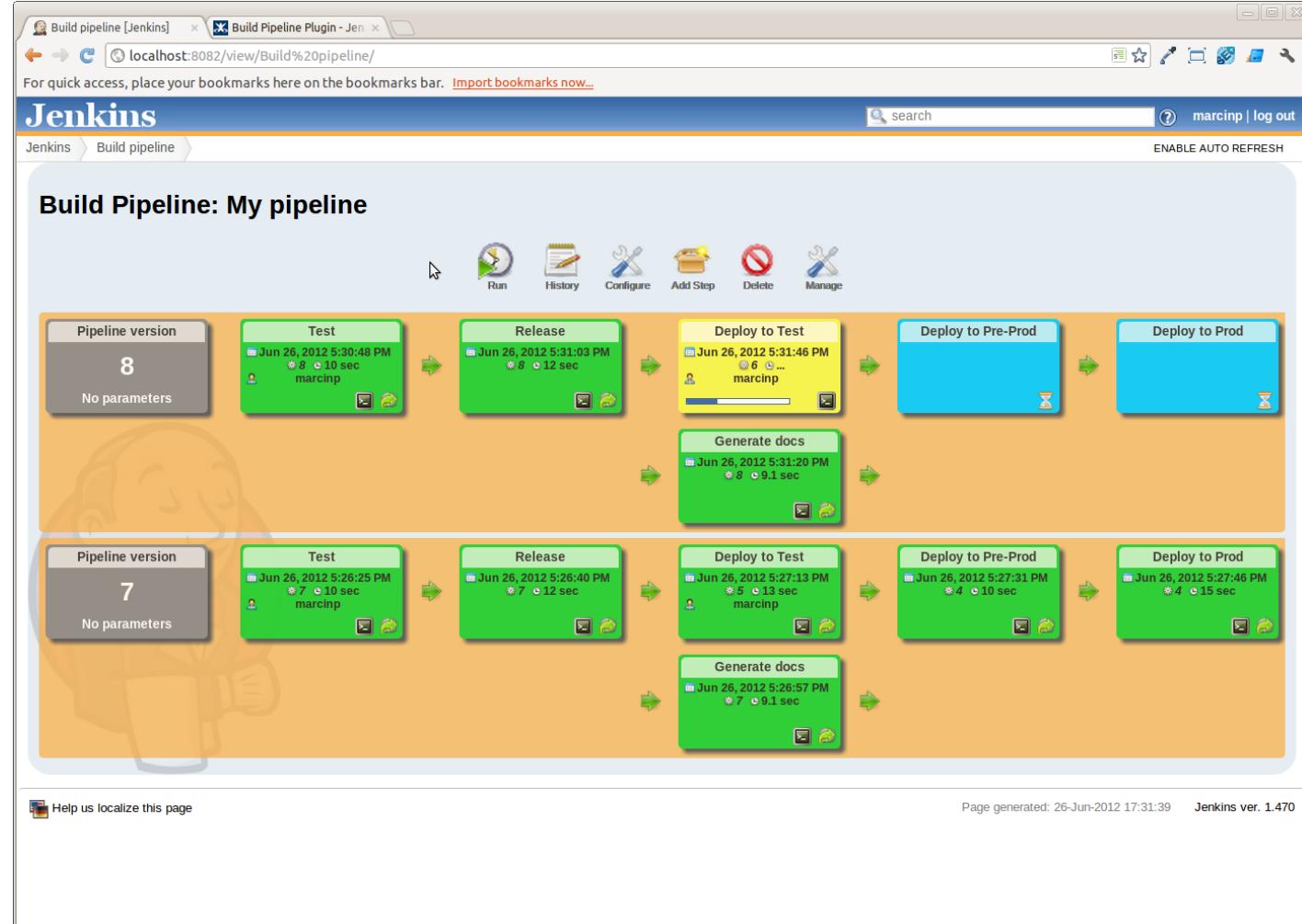
Phase 2 (Pre-migration)

All	Demo v1.0 (Jobs + Views)	Demo v2 (Job Templates)	app1	app2	automation	dev	project1	project2	qa	+
S	W	Name ↓	Last Success		Last Failure		Last Duration			
		app1-develop	N/A		N/A		N/A			
		app1-prod	N/A		N/A		N/A			
		app2-develop	N/A		N/A		N/A			
		base-template	N/A		N/A		N/A			
		component1	N/A		N/A		N/A			
		component2-develop	N/A		N/A		N/A			
		dev-build-template	N/A		N/A		N/A			
		env-setup-template	N/A		N/A		N/A			
		env-test-template	N/A		N/A		N/A			
		free1	N/A		N/A		N/A			
		free2	N/A		N/A		N/A			
		free3	N/A		N/A		N/A			
		free4	N/A		N/A		N/A			
		postbuild-template	N/A		N/A		N/A			

Icon: [S](#) [M](#) [L](#)

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

Visualization

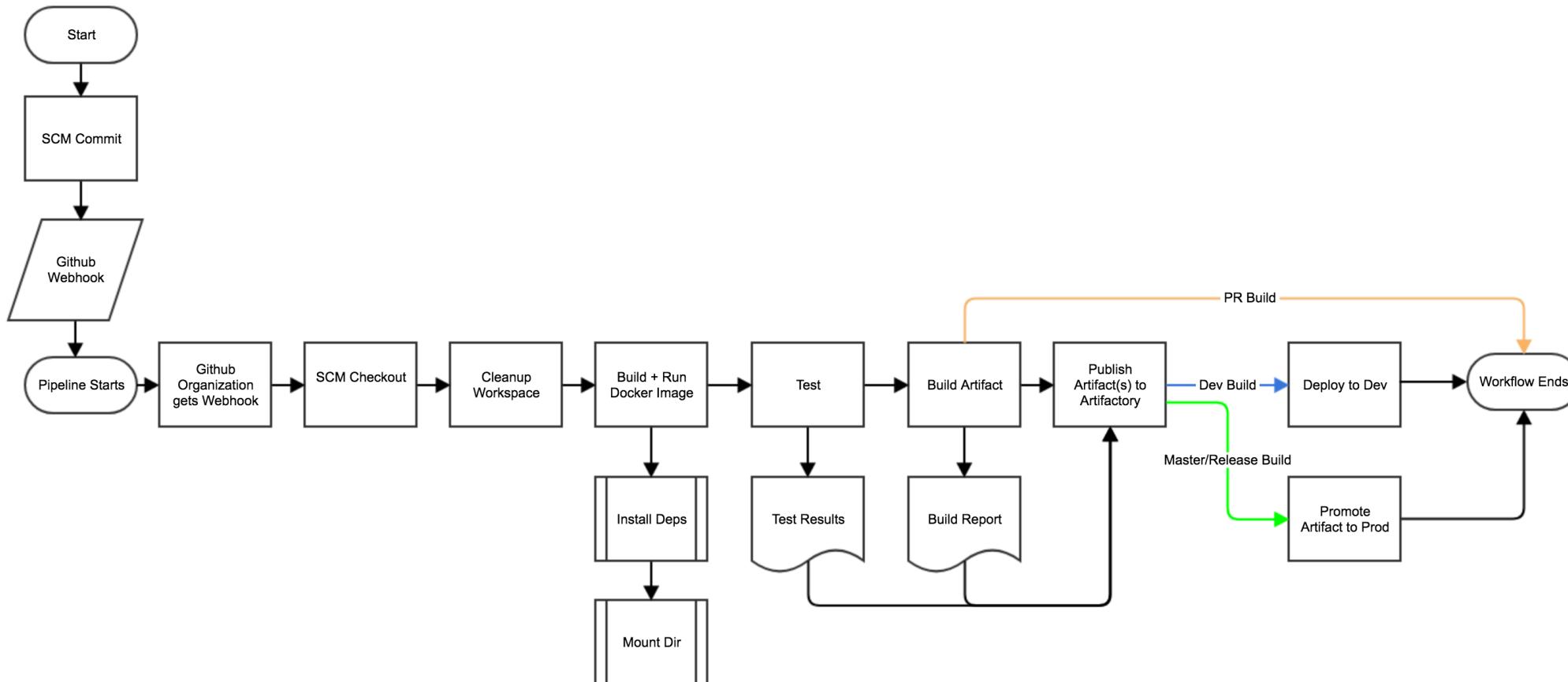


THE NEW WORLD

Inspiration

- <https://github.com/jenkinsci/simple-build-for-pipeline-plugin> -michaelneale
- Build a wrapper called simpleBuild that takes in parameters as a closure and runs logic

Jenkins Pipeline Workflow



Example Jenkinsfile

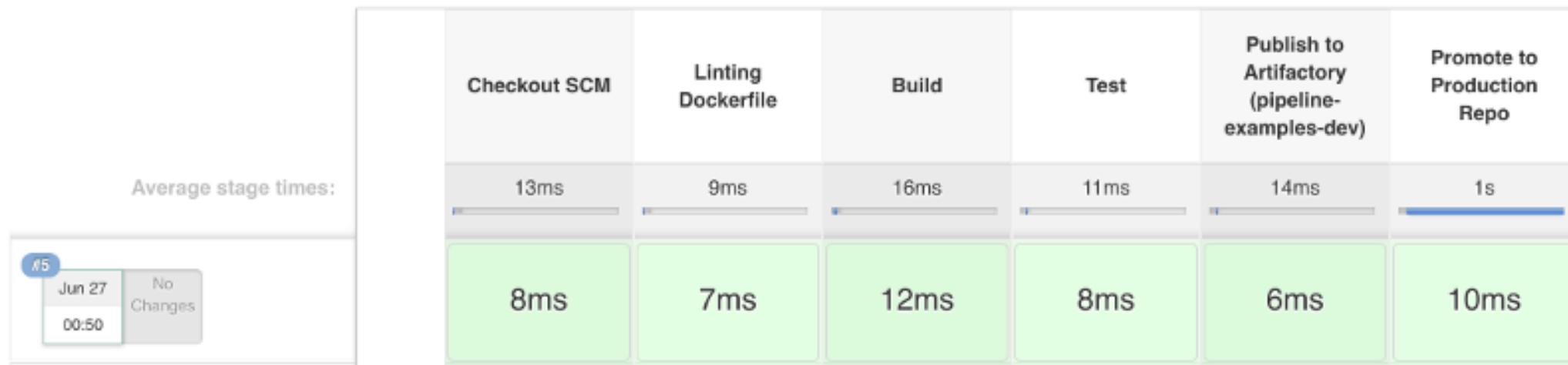
```
standardBuild {  
    machine          = 'docker'  
    dev_branch       = 'develop'  
    release_branch   = 'master'  
    artifact_apttern = '*.rpm'  
    html_pattern     = [keepAll: true, reportDir: '.', reportFiles: 'output.html', reportName: 'OutputReport']  
    dev_repo         = 'pipeline-examples-dev'  
    prod_repo        = 'pipeline-examples-prod'  
    pr_script        = 'make prs'  
    dev_script       = 'make dev'  
    release_script   = 'make release'  
}
```

Example Dockerfile

```
FROM faas/el7-python:base

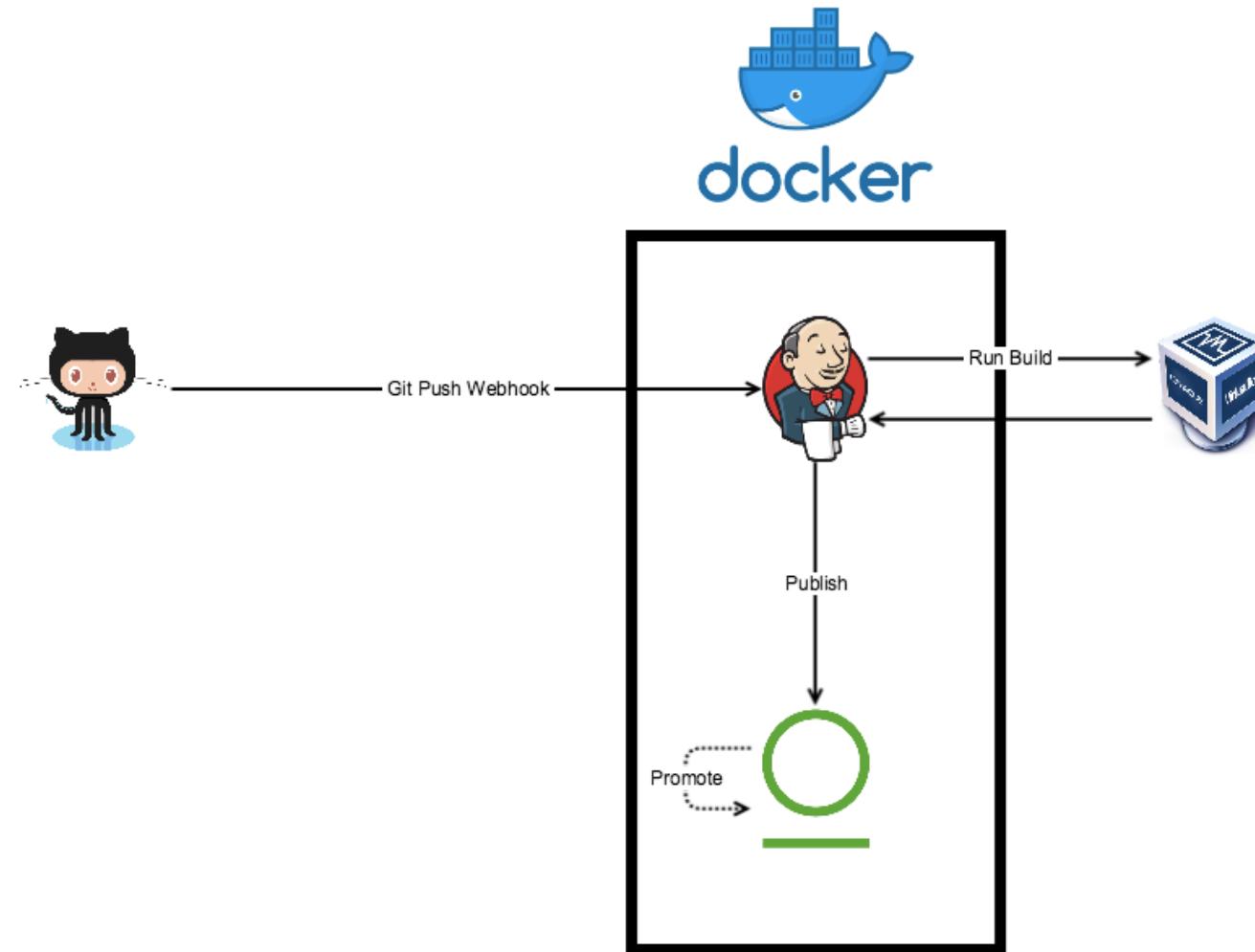
RUN yum install -y python-virtualenv \
    rpm-build && \
    yum clean all
```

Resulting Pipeline

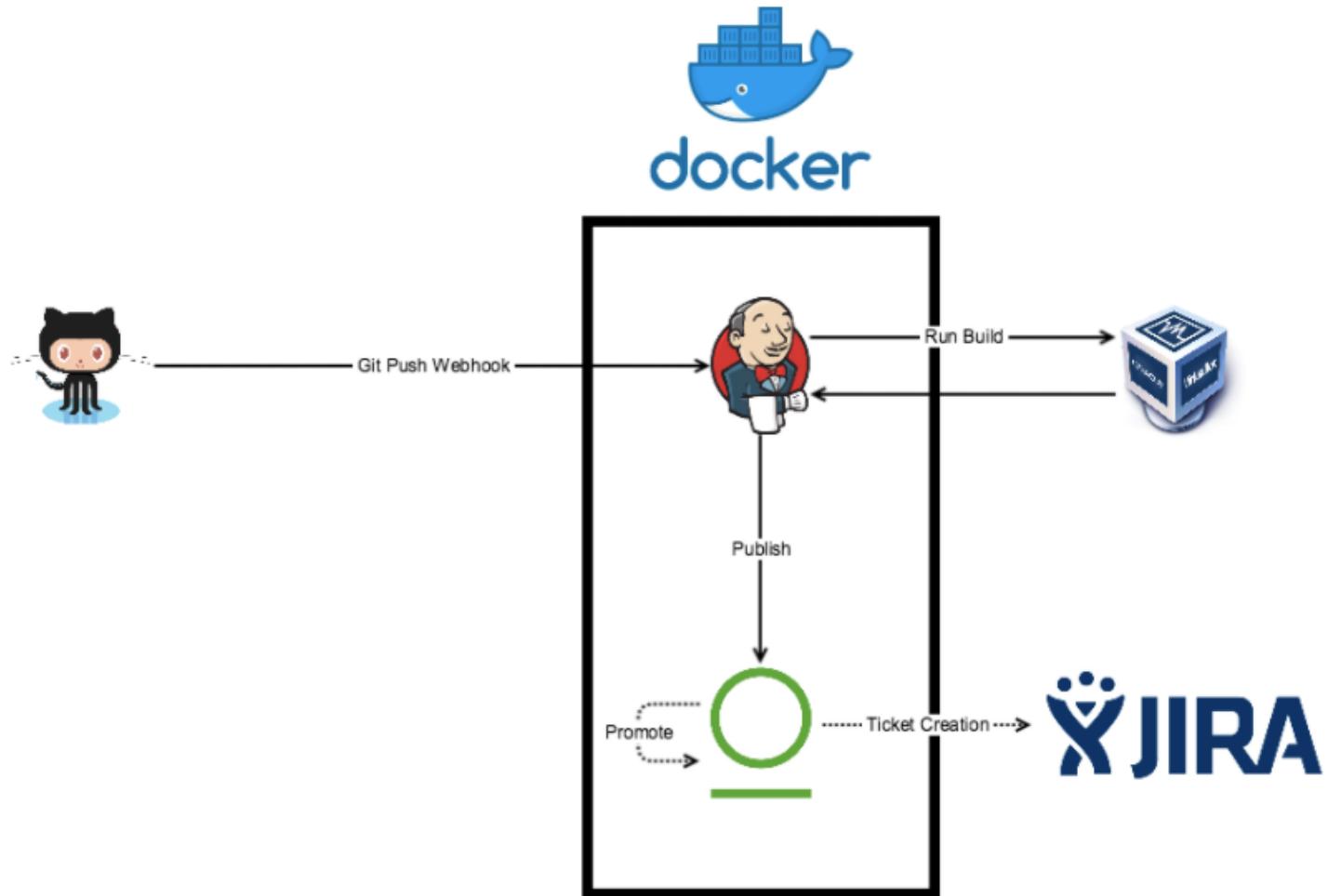


DEMO

Standard Build



Standard Build w/JIRA



How do we do this?

- Find a *passionate* liaison from each team
- Learn their workflow
 - There's probably a freestyle job already
- Start pipelining their workflow in modular pieces:
 - Checkout code
 - Run build script
 - Archive artifacts
 - Publish to Artifactory
 - Promote artifact
 - Send email
- Build a wrapper to call those functions
- Repeat for another team/app, reusing what you just made

Reap the Benefits

- Give Devs/QA their time back
- Standardization of jobs and workflows
- Standardization of build and test environments
- Jenkins maintenance and resilience made easy by decoupling tools

LESSONS LEARNED

Use Github Organizations

- Index every branch of every repo that has a Jenkinsfile
- Configure one Github hook on the organization level

The screenshot shows the Jenkins GitHub plugin interface. On the left, there is a sidebar with various navigation options: Up, Status, Configure, Scan Organization Now, Scan Organization Log, Organization Events, Delete Organization, People, Build History, GitHub, Pipeline Syntax, and Credentials. The main area displays the organization "alvin-huang-jenkinsworld-org" with a purple square icon. Below it, a table titled "Repositories (4)" lists four repositories: testrepo1, testrepo2, testrepo3, and testrepo4. Each repository entry includes a small icon, a status indicator (yellow sun), and a link to the repository details. At the bottom of the table, there are icons for sorting by S, M, L, a legend, and RSS feed links.

S	W	Name ↓	Description
		testrepo1	
		testrepo2	
		testrepo3	
		testrepo4	

Icon: [S](#) [M](#) [L](#)

[Legend](#) [RSS](#)

Make Steps Modular

- What does this step do?
 - Does A ‘AND’ B; break it apart
- Namespace them
 - stepCheckout.groovy – Custom Pipeline Step
 - standardBuild.groovy – Wrapper

Leverage the Snippet Generator

The screenshot shows the Jenkins Snippet Generator interface. The left sidebar contains links: Back, Snippet Generator (selected), Step Reference, Global Variables Reference, Online Documentation, and IntelliJ IDEA GDSL. The main content area has a header "Pipeline Syntax". It includes an "Overview" section with a brief description of the Snippet Generator's purpose, and a "Steps" section where a sample step "archiveArtifacts: Archive the artifacts" is selected. A "Files to archive" input field contains ".rpm". At the bottom, a "Generate Pipeline Script" button is shown, and the resulting Pipeline Script code is displayed as "archiveArtifacts '*.rpm'".

Benefits of Shared Libraries For Our Future

- Migration of Github
- Upgrade to Puppet 5
- Consolidation of Artifactory
- Application stacks in Openshift
- Deployments through Rundeck, Ansible or Fabric
- Scripts in ‘vars’ can be called in Declarative Pipeline

Upgrade to Puppet 5

```
FROM centos:centos6

# install Puppet5 repo
RUN yum install -y https://yum.puppetlabs.com/puppet5/puppet5-release-el-6.noarch.rpm

RUN yum install -y puppet-agent \
    rubygems && \
    yum clean all

ENV PATH="/opt/puppetlabs/puppet/bin:$PATH"

RUN gem install puppet-lint \
    puppet-syntax \
    yaml-lint && \
    gem cleanup all

WORKDIR "/code"
```

Dockerfile

```
def linter = ''
  docker build -t puppet-linter:latest .
  docker run -i --rm -v $(pwd):/code puppet-linter:latest bash linter.sh
  ...

standardBuild {
  machine      = "docker"
  artifact_pattern = null
  release_branch = '^(!PR-).*'
  release_script = linter
  dev_script    = linter
  pr_script     = linter
}
```

Jenkinsfile

Go Forth and Pipeline!



Thanks!

- alvin.huang@fireeye.com | @RealAlvinHuang | ahuang



THANK YOU

COPYRIGHT © 2017, FIREYE, INC. ALL RIGHTS RESERVED.

