

Lorem Ipsum Dolor

Jenkins Pipelines for LCG

Pere Mato

8 March 2021

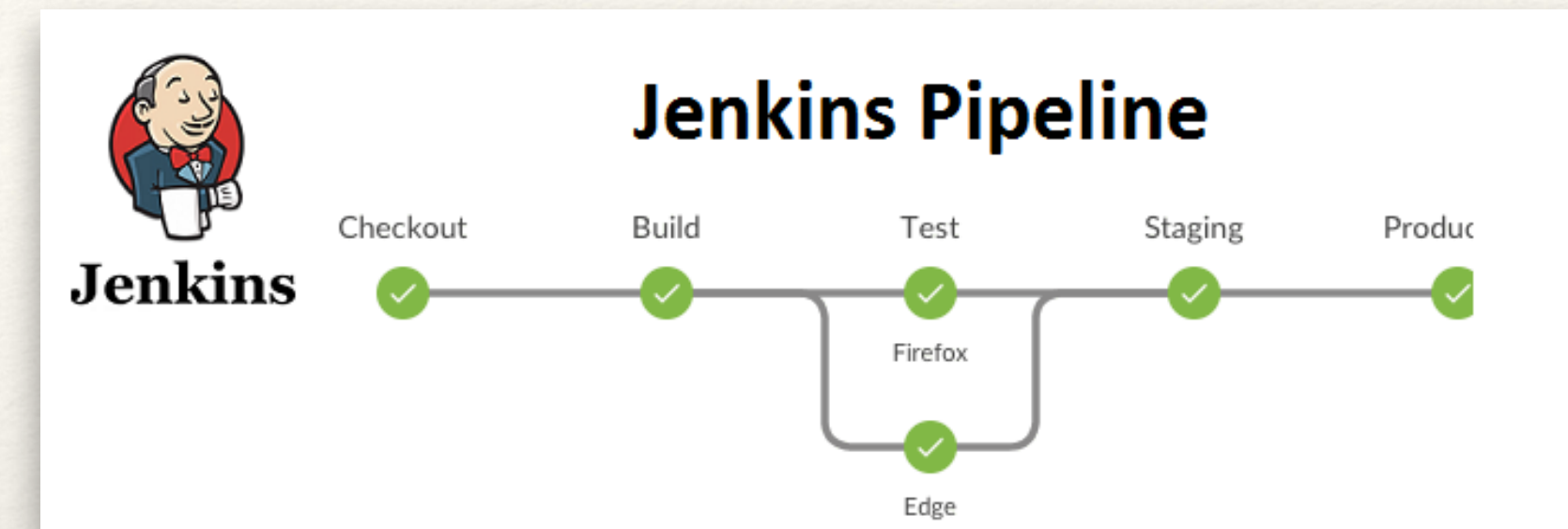
Motivation



- ❖ **Jenkins Pipeline** is a suite of plugins that supports implementing and integrating continuous integration and delivery pipelines into Jenkins
 - ❖ The *jenkinsfile* contains all the parameters, stages, steps to define the pipeline and is stored together with the source code of the project
- ❖ The Jenkins for the LCG nightlies and releases suffered from the following problems
 - ❖ Many independent jobs to implement the different stages (build, upload binaries, cvmfs install, test, etc.) needed be maintained (using a GUI), passing parameters between jobs has been a pain, several levels of scripts, ...
 - ❖ Not possible to mix docker and non-docker elements in a configuration matrix (ARM, MacOS, GPUs,...)
 - ❖ Keeping in sync several GIT repositories together with the Jenkins job configurations (not in GIT) using the Web GUI is complex and error prone

What is a Jenkins Pipeline?

- ❖ A *Jenkinsfile* can be written using two types of syntax: **Declarative** and **Scripted**
 - ❖ **Declarative Pipeline** is a more recent feature of Jenkins Pipeline which provides richer syntactical features over Scripted Pipeline syntax, and is designed to make writing and reading Pipeline code easier
- ❖ Jenkins job starts by getting the **Jenkinsfile** from the repository
 - ❖ It learns about the parameters, steps, options, instructions from the file itself



```
pipeline {
  agent any
  stages {
    stage('Stage 1') {
      steps {
        echo 'Hello world!'
      }
    }
  }
}
```

Jenkins Pipeline Concepts

- ❖ **Pipeline:** This is the fundamental block to the syntax of a declarative pipeline. It is a user-defined framework that includes all the processes like create, check, deploy, etc.
- ❖ **Agent:** It instructs Jenkins to assign the builds to an executor. Can be global or different for every **stage**. Several types of agents: Any, None, Label, Docker
- ❖ **Stage:** A stage block contains a series of steps in a pipeline. That is, the build, test, and deploy processes all come together in a stage. Generally, a stage block is used to visualize the Jenkins pipeline process.
- ❖ **Step:** A step is nothing but a single task that executes a specific process at a defined time. A pipeline involves a series of steps. **Groovy** can be used to write the scripts.

Creating Jenkins Pipeline Project

- ❖ Pipeline is a type of Jenkins project (if the Pipeline plugin is available in the instance)



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

- ❖ Write the directly the pipeline script or indicate the repository where the **Jenkinsfile** is

The screenshot shows the Jenkins Pipeline configuration interface. The 'Definition' section is set to 'Pipeline script from SCM'. Below this, the 'SCM' dropdown is set to 'None', the 'Script Path' is 'Jenkinsfile', and the 'Lightweight checkout' checkbox is checked. There are help icons (question marks) next to the SCM and Script Path fields. A 'Pipeline Syntax' link is visible at the bottom of the configuration area.

- ❖ That's all

With Jenkins Blue Ocean is supposed to be even easier!

LCG Nightlies use case

- ❖ Goal: Put all the LCG nightlies in **only two Jenkins jobs**
 - ❖ One with the **full pipeline** for a single version (dev3, devARM, etc.) and platform (compiler, architecture,.etc.) for all the steps (build, install, test, etc.)
 - ❖ One with a **full configuration** matrix that triggers N full pipeline jobs on schedule

Full Pipeline for a single Version/Platform

- ❖ Mixing docker and non-docker build and tests
 - ❖ Stages are executed on certain conditions (logical expressions)
- ❖ Installation in CVMFS is slightly different for MacOS and Linux
 - ❖ Pre-installation to fix RPATHs on MacOS

```
pipeline {
  //---Global Parameters---
  parameters {
  }
  //---Options---
  options {
  }
  //---Additional Environment Variables---
  environment {
  }
  agent none
  stages {
    //---Environment Stage to load and set global parameters---
    stage('Environment') {
    }
    //---Build stages---
    stage('InDocker') {
    }
    stage('InBareMetal') {
    }
    //---CVMFS Installation stage---
    stage('MacPreinstall') {
    }
    stage('CVMFSInstall') {
    }
    //---Testing stages---
    stage('TestInDocker') {
    }
    stage('TestInBareMetal') {
    }
  }
}
```


Global Parameters

```
//---Global Parameters-----
parameters {
  string(name: 'LCG_VERSION', defaultValue: 'dev4cuda', description: 'LCG stack version to build')
  choice(name: 'COMPILER', choices: ['gcc8', 'gcc9', 'gcc10', 'clang8', 'clang10', 'native'])
  choice(name: 'BUILDTYPE', choices: ['Release', 'Debug'])
  choice(name: 'LABEL', choices: ['centos7', 'centos8', 'slc6', 'macos1015'])

  string(name: 'LCG_INSTALL_PREFIX', defaultValue: '/cvmfs/sft.cern.ch/lcg/latest:/cvmfs/sft.cern.ch/lcg/releases', description: 'Location to look for already installed packages matching version and hash value. Leave blank for full rebuilds')
  string(name: 'LCG_ADDITIONAL_REPOS', defaultValue: 'http://lcgpackages.web.cern.ch/lcgpackages/tarFiles/latest', description: 'Additional binary repository')
  string(name: 'LCG_EXTRA_OPTIONS', defaultValue: '-DLCG_SOURCE_INSTALL=OFF;-DLCG_TARBALL_INSTALL=ON', description: 'Additional configuration options to be added to the cmake command')
  string(name: 'LCG_IGNORE', defaultValue: '', description: 'Packages already installed in LCG_INSTALL_PREFIX to be ignored (i.e. full rebuild is required). The list is \'single space\' separated.')
  booleanParam(name: 'USE_BINARIES', defaultValue: true, description: 'Use binaries in repositories')
  string(name: 'VERSION', defaultValue: 'master', description: 'LCGMake branch to use for the build (master, experimental, ...)')
  string(name: 'TARGET', defaultValue: 'all', description: 'Target to build (all, <package-name>, ...)')
  choice(name: 'BUILDMODE', choices: ['nightly', 'release'], description: 'CDash mode')

  booleanParam(name: 'CLEAN_INSTALLDIR', defaultValue: false, description: 'Instruct to clean the install directory before bulding')
  booleanParam(name: 'CVMFS_INSTALL', defaultValue: true, description: '')
  booleanParam(name: 'RUN_TESTS', defaultValue: true, description: '')
  booleanParam(name: 'VIEWS_CREATION', defaultValue: true, description: '')
  string(name: 'CVMFS_DEPLOYER', defaultValue: 'cvmfs-sft-nightlies', description: 'CVMFS publisher node label')
}
```


Starting a Parametrised job

- ❖ We can then trigger a build using the usual Jenkins GUI panels
- ❖ It reads the **Jenkinsfile** and offers all parameters to the User
- ❖ Parameter definition and default values stays in the Jenkinsfile

Pipeline lcg_nightly_pipeline

This build requires parameters:

LCG_VERSION	<input type="text" value="dev4cuda"/> <small>LCG stack version to build</small>
COMPILER	<input type="text" value="gcc7"/>
BUILDTYPE	<input type="text" value="Release"/>
LABEL	<input type="text" value="centos7"/>
DOCKER_LABEL	<input type="text" value="docker-host"/> <small>Label for the the nodes able to launch docker images</small>
ARCHITECTURE	<input type="text"/> <small>Complement of the architecture (instruction set)</small>
LCG_INSTALL_PREFIX	<input type="text" value="/cvmfs/sft.cern.ch/lcg/latest:/cvmfs/sft.cern.ch/lcg/releases"/> <small>Location to look for already installed packages matching version and hash value. Leave blank for full rebuilds</small>
LCG_ADDITIONAL_REPOS	<input type="text" value="https://lcgpackages.web.cern.ch/tarFiles/latest"/> <small>Additional binary repository</small>
LCG_EXTRA_OPTIONS	<input type="text" value="-DLCG_SOURCE_INSTALL=OFF;-DLCG_TARBALL_INSTALL=ON"/> <small>Additional configuration options to be added to the cmake command</small>
LCG_IGNORE	<input type="text"/> <small>Packages already installed in LCG_INSTALL_PREFIX to be ignored (i.e. full rebuild is required). The list is 'single space' separated.</small>
TARGET	<input type="text" value="all"/> <small>Target to build (all, , ...)</small>
BUILDMODE	<input type="text" value="nightly"/> <small>CDash mode</small>
	<input type="checkbox"/> CLEAN_INSTALLDIR <small>Instruct to clean the install directory before building</small>
	<input checked="" type="checkbox"/> CVMFS_INSTALL
	<input checked="" type="checkbox"/> RUN_TESTS
	<input checked="" type="checkbox"/> VIEWS_CREATION
CVMFS_DEPLOYER	<input type="text" value="cvmfs-sft-nightlies"/> <small>CVMFS publisher node label</small>
BRANCH	<input type="text" value="master"/> <small>LCGMake branch to use for the build (master, experimental, ...)</small>

Build

Docker Integration

- ❖ Docker builtin agent ready to be used
 - ❖ host and container share the same Workspace
- ❖ Support for kubernetes is also available :-)

```
stage('InDocker') {  
  when {  
    beforeAgent true  
    expression { params.LABEL.matches('centos.*|ubuntu.*') }  
  }  
  agent {  
    docker {  
      image "gitlab-registry.cern.ch/sft/docker/${LABEL}"  
      label 'docker-host'  
      args """-v /cvmfs:/cvmfs  
            -v /ccache:/ccache  
            -v /ec:/ec  
            --hostname ${LABEL}-docker  
            """"  
    }  
  }  
}
```


Build and Copy Stages

- ❖ Any shell command can be used
- ❖ Some variables can be updated after the build and transferred to subsequent stages
 - ❖ e.g. BUILDHOST, CTEST_TAG, CVMFS revision, ...

```
stages {
  stage('Build') {
    steps {
      script { ...
      }
      sh """
      source lcgcmake/jenkins/jk-setup.sh $BUILDTYPE $COMPILER
      env | sort | sed 's/:/:?/g' | tr '?' '\n'
      ctest -V -DCTEST_LABELS=$CTEST_LABELS -S lcgcmake/jenkins/lcgcmake-build.cmake
      """
      script { ...
      }
    }
  }
  stage('CopyToEOS') {
    steps {
      sh """
      export PLATFORM=${PLATFORM}
      set +x
      source /cvmfs/sft.cern.ch/lcg/releases/LCG_98/xrootd/4.12.3/${PLATFORM}/xrootd-env.sh
      set -x
      lcgcmake/jenkins/copytoEOS.sh
      lcgcmake/jenkins/copytoLatest.sh
      """
    }
  }
}
```


Test Stages

- ❖ Test can be run on different nodes
- ❖ Variables and parameters are shared within all the pipeline
- ❖ Facilitates communication between stages

```
steps: {
  sh: label: 'wait-for-cvmfs',
    script: """
      export PLATFORM=${PLATFORM}
      lcgcmake/jenkins/wait_for_cvmfs.sh
    """
  dir('lcgtest'): {
    git: branch: 'master',
      url: https://gitlab.cern.ch/sft/lcgtest.git
  }
  sh: label: 'test-with-views',
    script: """
      export PLATFORM=${PLATFORM}
      export BUILDHOSTNAME=${BUILDHOSTNAME}
      export IGNORE_PYTHON_MODULE="\${IGNORE_PYTHON_MODULE}"
      [-d test_build] && rm -rf test_build
      mkdir -p test_build/Testing
      echo "${CTEST_TAG}" > test_build/Testing/TAG
      source /cvmfs/sft.cern.ch/lcg/views/$LCG_VERSION/latest/$PLATFORM/setup.sh
      ctest -V -DCTEST_LABELS=".*" -S lcgcmake/jenkins/lcgtest-test.cmake
    """
}
```


Pipeline Stage View

Stage View		Environment	InDocker	Build	CopyToEOS	InBareMetal	Build	CopyToEOS	MacPreinstall	CVMFSInstall	TestInDocker	WaitForCVMFS	TestWithViews	TestShell	TestPythonImport	TestInBareMetal	WaitForCVMFS	TestWithViews	TestShell	TestPythonImport
Average stage times: (Average full run time: ~4h 15min)		13s	4s	52min 49s	17s	0ms	0ms	0ms	0ms	3h 16min	1min 11s	6min 3s	1min 16s	8s	3min 53s	21s	0ms	0ms	0ms	0ms
#5239 devgeantv-centos7-gcc9-opt Mar 05 06:06 1 commit		684ms	1s	27s	2s					3h 12min	40s	10min 13s	36s	10s	26s					
#5238 devgeantv-centos7-clang10-opt Mar 05 05:59 1 commit		719ms	23s	11min 51s	6s					3h 9min	1min 6s	5min 7s	43s	9s	29s					
#5237 devgeantv-centos7-gcc7-opt Mar 05 05:59 1 commit		17s	2s	39min 16s	11s					3h 14min	20s	5min 7s	35s	8s	26s					
#5236 devgeantv-centos7-gcc9-opt Mar 05 05:39 1 commit		3s	2s	1min 32s	4s					3h 5min	3min 9s	5min 7s	33s	9s	23s					
#5235 devgeantv-centos7-gcc8-opt Mar 05 05:38 1 commit		47s	1s	43s	2s					3h 5min	2min 42s	5min 7s	42s	10s	26s					
#5234 dev3cuda-centos7-gcc8-opt Mar 05 05:33 1 commit		5s	2s	1h 1min	20s					3h 16min						42s	10min 13s	1min 49s	10s	6min 41s
#5233 dev3-centos8-gcc10-dbg Mar 05 05:23 1 commit		2s	2s	2h 7min	27s					3h 21min	3s	5min 7s	1min 25s failed	37ms failed	47ms failed	25ms	33ms failed	35ms failed	35ms failed	35ms failed
#5232 dev3-centos7-gcc9-dbg Mar 05 05:12 1 commit		42s	1s	2h 0min	55s					3h 28min	1min 33s	5min 7s	3min 12s	4s	12min 3s					
#5231 dev4-centos7-gcc8-dbg Mar 05 05:12 1 commit		13s	1s	1h 39min	32s					3h 38min	1min 0s	5min 7s	3min 4s	14s	10min 34s					
#5230 dev4-centos7-gcc9-opt Mar 05 05:10 1 commit		1s	1s	1h 5min	14s					3h 15min	1s	10min 13s	1min 19s	14s	11min 12s					

Nightly Configuration Matrix

- ❖ Single matrix for all LCG nightlies
 - ❖ Global view
 - ❖ No details on how to build / test each cell
 - ❖ Spawn N pipeline jobs

Configuration Matrix			dev3	dev3python2	dev4	dev4python2	dev3cuda	dev4cuda	devARM	devBE	devAdePT
centos7	native	Release	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
		Debug	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
	gcc8	Release	●	●	●	●	●	●	⬤	●	●
		Debug	●	●	●	●	⬤	⬤	⬤	⬤	⬤
	gcc9	Release	●	●	●	●	⬤	⬤	⬤	⬤	⬤
		Debug	●	●	●	●	⬤	⬤	⬤	⬤	⬤
	gcc10	Release	●	●	●	●	⬤	⬤	⬤	⬤	⬤
		Debug	●	●	●	●	⬤	⬤	⬤	⬤	⬤
	clang10	Release	●	●	●	●	⬤	⬤	⬤	⬤	⬤
		Debug	●	●	●	●	⬤	⬤	⬤	⬤	⬤
centos8	native	Release	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
		Debug	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
	gcc8	Release	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
		Debug	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
	gcc9	Release	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
		Debug	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
	gcc10	Release	●	⬤	●	⬤	⬤	⬤	⬤	⬤	⬤
		Debug	●	⬤	●	⬤	⬤	⬤	⬤	⬤	⬤
	clang10	Release	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤
		Debug	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤	⬤

Release Pipeline

- ❖ Implemented a 'release' pipeline with slightly different stages (publish LCGinfo DB, create and publish RPMs, testing RPMs, etc.)
- ❖ The **nightly** and **release** pipeline share the same set of instructions
 - ❖ File with Groovy functions and variables

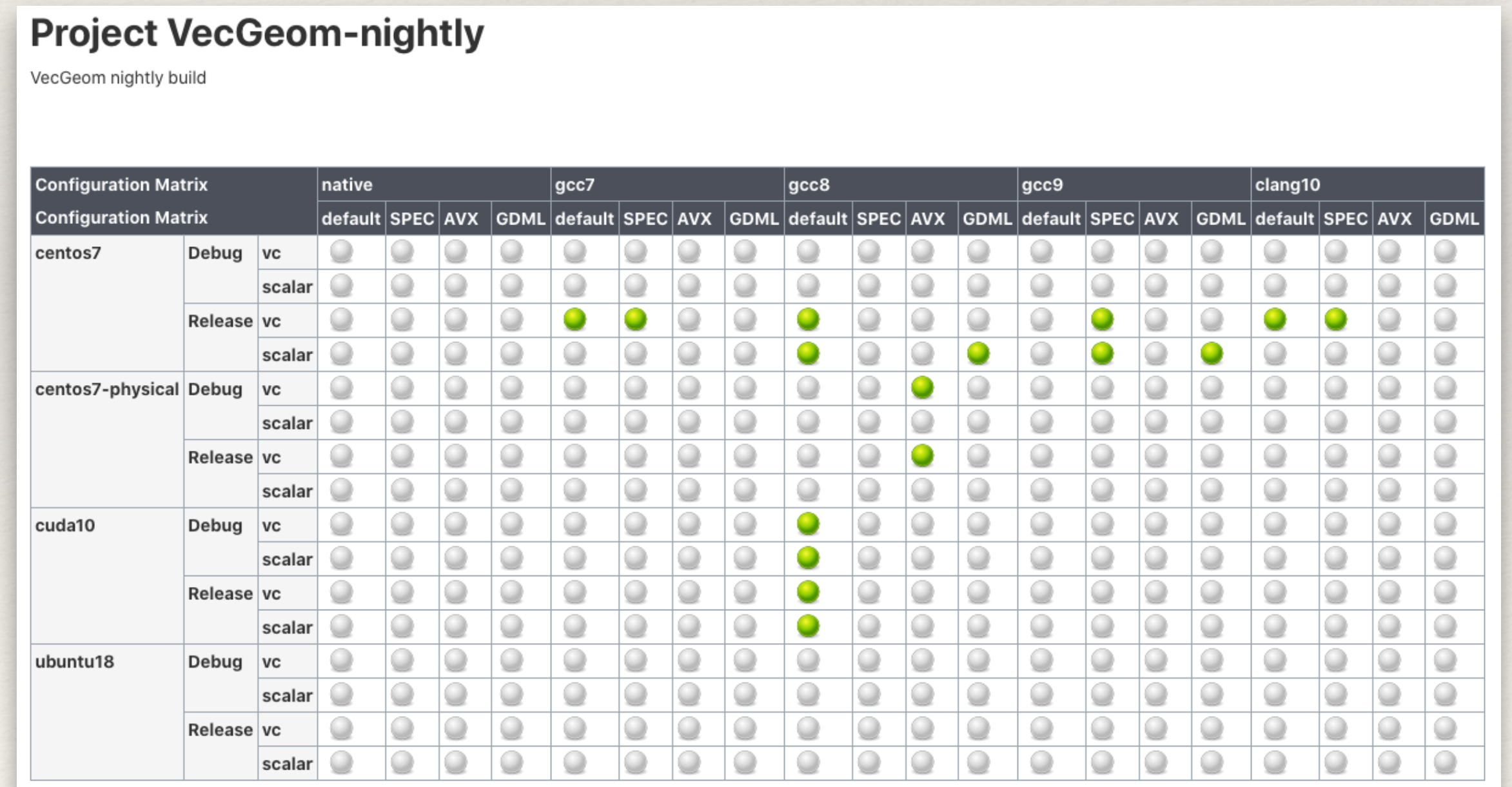
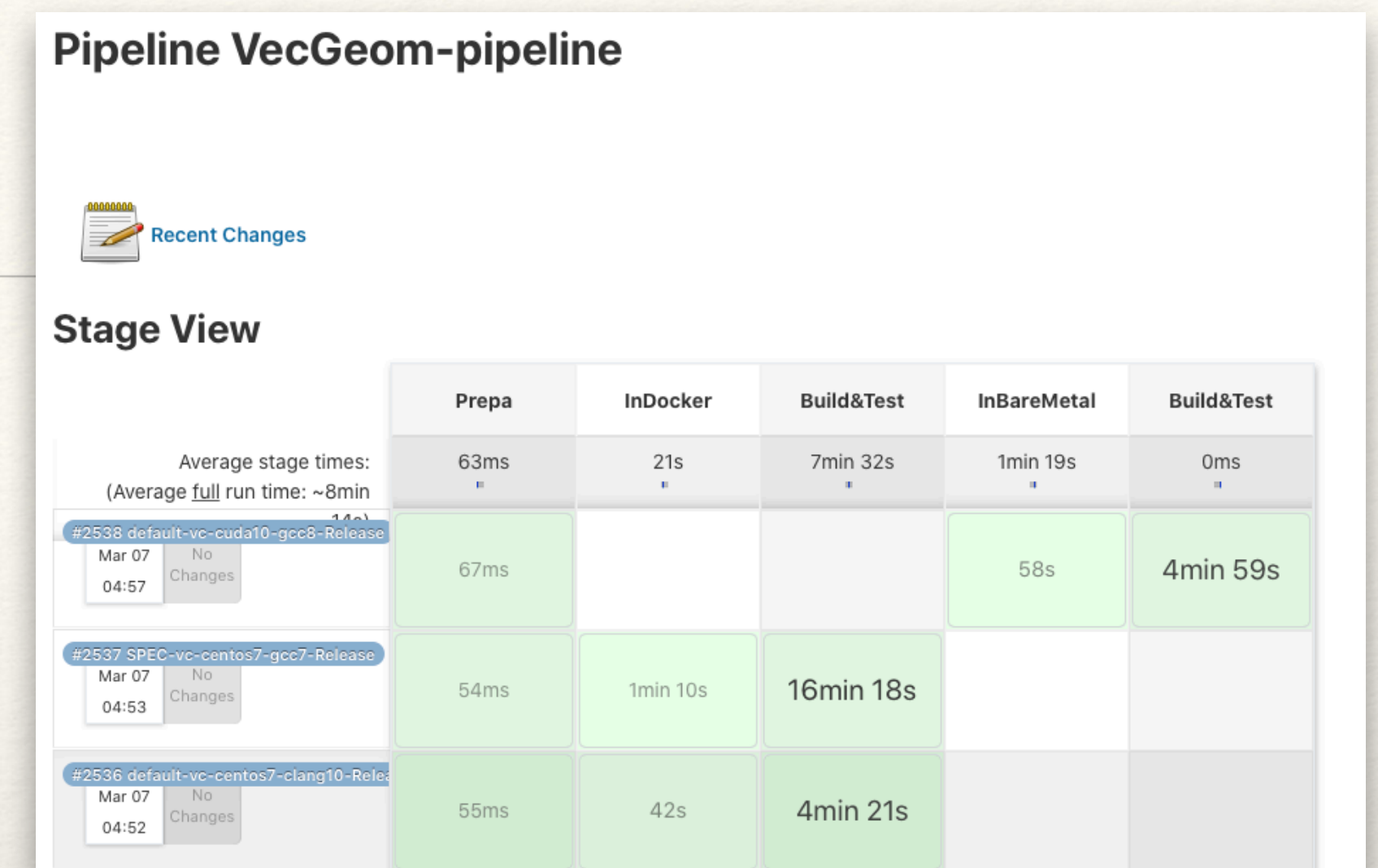
```
stage('Environment') {
  agent { label "$DOCKER_LABEL" }
  steps {
    script {
      //---Load common variables and functions between several pilelines---
      lcg = load 'lcgmake/jenkins/Jenkinsfunctions.groovy'
    }
  }
}
...
stage('Build') {
  steps {
    script {
      lcg.buildPackages()
    }
  }
}
```

Current LCG Status

- ❖ LCG Nightlies are done using the Jenkins Pipelines on a mix of docker images (slc6, centos7, centos8, ubuntu20) and bare-metal nodes (arm64, MacOS, GPU)
- ❖ The latest release, LCG_99, was done completely using the release pipeline with the possibility of doing all the steps in one go or selectively (via a number of boolean parameters)









VecGeom

- ❖ Implemented Jenkins Pipeline VecGeom
 - ❖ Added Jenkinsfile in GitHub repository
 - ❖ The same pipeline is triggered either by GitLab MRs or scheduled daily for the nightlies



AdePT

- ❖ Similar as VecGeom but the repository is in GitHub instead of GitLab
 - ❖ Added **Jenkinsfile** to repository
 - ❖ Added **Webhook** connecting to our Jenkins instance
 - ❖ Pipeline is triggered by a PR or interaction with the 'bot'
 - ❖ Missing the some reporting back from Jenkins to GitHub

S	W	Name ↓	Last Success
		AdePT-CI	3 days 2 hr - #75-drbenmorgan#87-gcc8-Release
		AdePT-nightly	17 hr - #49
		AdePT-pipeline	16 hr - #100 gcc8-Debug
		AdePT-PR-trigger	3 days 2 hr - #82

Summary

- ❖ Using Jenkins (declarative pipelines) has simplified enormously the LCG nightlies and releases
 - ❖ Only two Jenkins projects (instead of many: for the different stacks, platforms, stages, etc.)
 - ❖ For the producing the releases several manual interventions has been avoided
 - ❖ The **jenkinsfile** with the configuration and instructions sits in the repository itself, thus facilitating the maintenance
- ❖ Same logic has been replicated for other projects to implement CI: VecGeom (GitLab), AdePT (GitHub)