

# HPE NonStop and Modern DevOps

## Instructions Set for CI/CD

## Contents

About this document.....	4
Background.....	4
Tools .....	4
Recommended Setup .....	5
Continuous Integration Setup .....	6
SCM Setup and Access Control.....	6
GIT Installation and Access to GITHUB.....	6
NSGit Installation (Optional).....	6
Continuous Integration Tool Setup.....	7
Jenkins Installation .....	7
Jenkins Configuration.....	7
NonStop Node.....	7
Credentials.....	8
Static Analysis Tool Setup.....	9
Artifact Repository Tool Setup .....	9
Continuous Integration Pipeline.....	10
Pipelines in Jenkins.....	10
Descriptive versus Scripted Pipelines.....	10
Github Access in a Pipeline and automatic build trigger .....	10
Continuous Deployment Setup .....	12
Ansible .....	12
NSMPT .....	12
NSMF .....	12
Requisite Hardware and Software .....	12
Installation on NonStop Server (Managed Node) .....	12
SSH access to NonStop .....	13
Installation on Linux Server (Controller Node) .....	13
Ansible and Jenkins Integration.....	13
Mapping Ansible Node in Jenkins .....	13
SSH setup for playbook .....	14
Ansible playbook execution from Jenkins Pipeline.....	15
Java Starter Kit.....	17
Package Information .....	17

Using the Starter Kit .....	17
Downloading the project from HPE NonStop github .....	17
Uploading the project to SCM .....	18
Continuous Integration Pipelines using Java Starter Kit .....	18
Developer Pipeline .....	18
Nightly Build Pipeline .....	24
Java JNI Starter Kit .....	26
Package Information .....	26
Using the Starter Kit .....	27
Downloading the project from HPE NonStop github .....	27
Uploading the project to SCM .....	27
Continuous Integration Pipelines using Java JNI Starter Kit .....	28
JNIDeveloper Pipeline .....	28
Continuous Deployment starter kit .....	31
Package Information .....	31
Using the Starter Kit .....	31
Uploading the project to SCM .....	31
Continuous Deployment Pipeline using Starter Kit .....	32
Application Release and Deployment Pipeline .....	32
Python Starter Kit .....	41
Package Information .....	41
Using the Starter Kit .....	42
Uploading the project to SCM .....	42
Continuous Integration Pipelines using Python Starter Kit .....	42
Developer Pipeline .....	42
Trouble Shooting .....	49

## About this document

This document is a part for Modern DevOps on NonStop and describes the recommended setup for Continuous Integration, Continuous Delivery and Continuous Deployment.

## Background

DevOps is a set of principles, practices to build effective collaboration between Development & Operations teams. DevOps streamlines processes, propagates efficient releases, and ensures quality builds and relaxed deployments. Main principles of DevOps – automation, continuous, and quick feedback cycle.

## Tools

DevOps is achieved through automation and tools are necessary for Automation.

The following table gives the set of tools required to enable DevOps on NonStop.

Agile Management Tools	Coding and Development Tools	Build and Automation Tools	Testing Tools	Release Management Tools	Software Deployment Tools	Operational Monitoring Tools
JIRA	ECLIPSE , NSDEE and ADE Tools	MAVEN, GRADLE, MAKE	BOOST FRAMEWORK	NEXUS , ARTIFACTORY	ANSIBLE, NSMF	EMS, MEASURE
	GIT, GITHUB, NSGIT	JENKINS	ALM			PROGNOSIS, NAGIOS
	FORTIFY, SONARQUBE		SELENIUM, ROBO FRAMEWORK			
			JMETER, LOAD RUNNER			

The important tools for CI/CD are

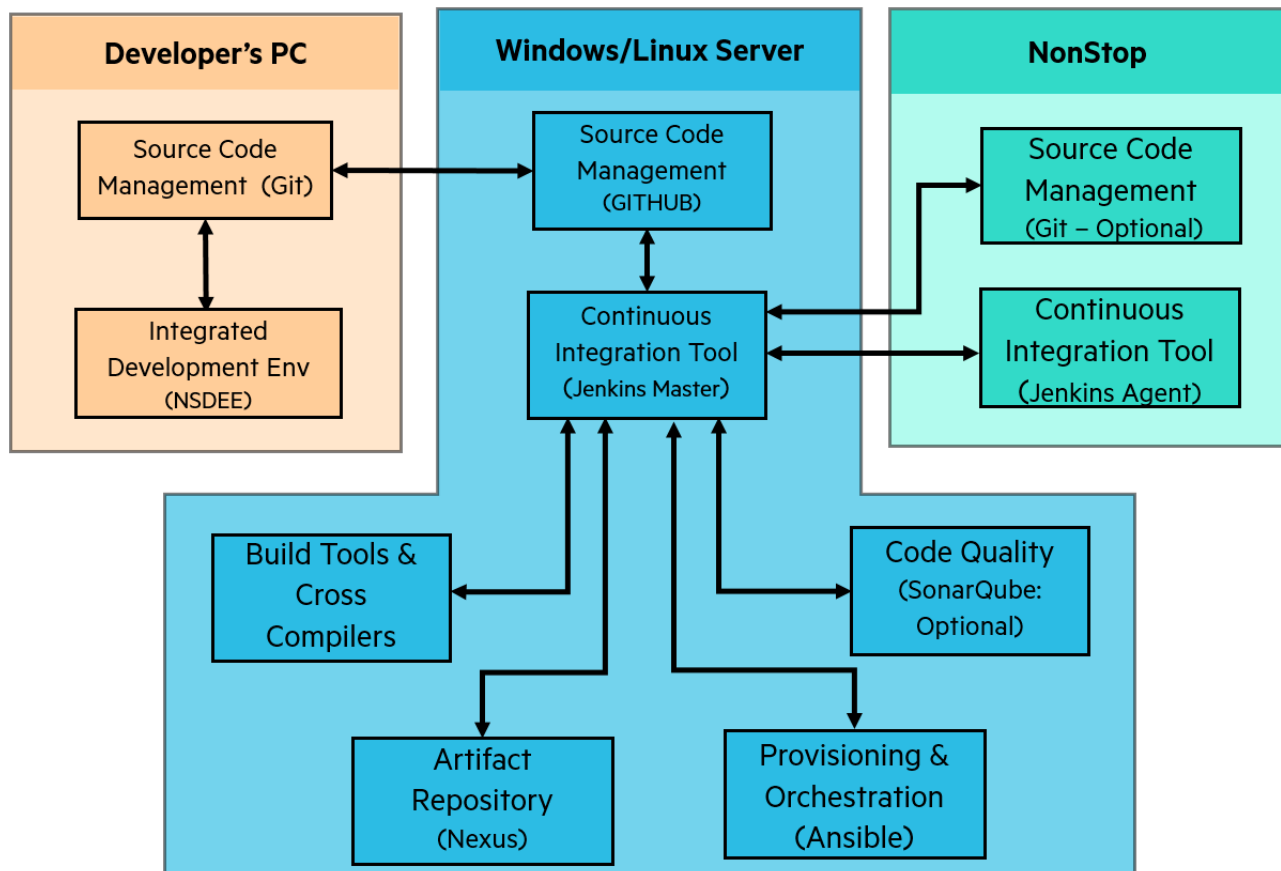
- 1) Software Configuration Management Tool (GITHUB/NSGIT)
- 2) Continuous Integration Tool (Jenkins)
- 3) Provision and Orchestration Tool (Ansible)

Configuration and usage of the CI tools and the Java Starter Kit for Continuous Integration is discussed in detail in this document.

## Recommended Setup

The Devops Setup recommended for NonStop is given below

- Setup SCM and CI Tools on Windows/Linux Server
- CI Tool Jenkins Master Node on Windows/Linux Server
- Jenkins Agent on NonStop Node. Needs SSH Access
- Use Cross Compilers and NSEEE to edit and compile
- Setup Artifact Repository and Provision and Orchestration software on Windows/Linux Server



## Continuous Integration Setup

### SCM Setup and Access Control

#### GIT Installation and Access to GITHUB

- Download the latest GIT from <https://git-scm.com/downloads>

Accept default components during installation.

Select the text editor of your choice for git's default editor.

If the installation is on Windows, the following are recommended

Keep "Use Git from the Windows Command Prompt" for path option.

Keep "Use OpenSSH" for ssh executable, if prompted

Keep "Use the OpenSSL library" if prompted

Ensure to select Unix-style line endings so that the files when moved to NonStop does not have ^M characters.

Keep "Use MinTTY"

Keep all the default "Extra options"

- Setup access to github instance:

Make sure that you can log on to <https://github.<youcompanyname>.com/>

using your credentials

- Generate SSH key pair for github:

<https://help.github.com/en/enterprise/2.21/user/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

- Install SSH key into github account:

<https://help.github.com/en/enterprise/2.21/user/articles/adding-a-new-ssh-key-to-your-github-account>

- Install ssh key for git bash:

<https://gist.github.com/bsara/5c4d90db3016814a3d2fe38d314f9c23>

#### NSGit Installation (Optional)

Refer to NSGit Manual

## Continuous Integration Tool Setup

### Jenkins Installation

- Download the latest version of Jenkins from <https://www.jenkins.io/download/>
- Go through the platform specific instructions from <https://www.jenkins.io/doc/book/installing/> to install Jenkins
- Select the default plug-ins during Installation (Mailer, Maven, Ant, etc)
- Create the administrator credentials
- Login using administrator credentials
- After logging in add additional users if required

### Jenkins Configuration

#### Plugins

- A number of Jenkins Plugin are required.
- Refer to <https://www.jenkins.io/doc/book/managing/plugins/> to install
- Preferable Use the GUI and click on Manage Jenkins -> Manage Plugin to install. Use the install without restart option
- Plugins required include
  - a. GIT, GIT Client, GIT HUB
  - b. Static Analysis (Static Analysis Utility, Static Analysis Collector, Warnings Next Generation)
  - c. SonarQube Scanner (optional)
  - d. Pipeline
  - e. Ansible & Ansible Tower (optional)
  - f. Nexus Platform (optional)
  - g. SSH (SSH Plugin, SSH Pipeline Steps)
  - h. Publish Over SSH (optional)

### NonStop Node

NonStop Systems are mapped and managed as nodes in the recommended setup. A Jenkins Agent will run on NonStop. SSH will be used for communication. Hence, SSH access to NonStop is required.

- Click on Manage Jenkins → Click on Manage Nodes section
- Create a new node
- Enter a node name
- Whether the node will run permanently - (Yes)
- Node Name
- Number of executors
- Root Directory where the agent will run
- Labels (to group one or more NonStop Systems or to group systems based on functionality/usage)
- Usage details – Use this node as much as possible

- Launch Method - Launch via SSH
- Host IP
- Credentials (Note – the credential should be added via Jenkins Credentials Only)
- Host Key Verification Strategy – Manually Trusted Key Verification
- Click on Advance Tab and set the following
- SSH port - retain the SSH port as 22
- Set the Java path
- Update the number of retries and connection timeout values
- Availability – Keep the agent online as much as possible
- [If the NonStop System is **not** a production system with live applications, then use this option. Otherwise it might affect the system performance as the agent will be constantly pinging the master]
- Provide Environment variables if required.

Name	cfot	
Description		
# of executors	1	
Remote root directory	/home/	
Labels	cfot	
Usage	Use this node as much as possible	
Launch method	Launch agents via SSH	
Host		
Credentials	super.super/***** (cfot) <span>Add</span>	
Host Key Verification Strategy	Manually trusted key Verification Strategy	
	<input type="checkbox"/> Require manual verification of initial connection	
Port	22	
JavaPath	/usr/tandem/nssjava/jdk180_180/bin/java	
JVM Options		
Prefix Start Agent Command		
Suffix Start Agent Command		
Connection Timeout in Seconds	60	
Maximum Number of Retries	10	
Seconds To Wait Between Retries	15	
Use TCP_NODELAY flag on the SSH connection	<input checked="" type="checkbox"/>	
Remoting Work directory		
Availability	Keep this agent online as much as possible	
Node Properties		
<input type="checkbox"/> Disable deferred wipeout on this node		
<input checked="" type="checkbox"/> Environment variables		
List of variables	<span>Add</span>	
<input checked="" type="checkbox"/> Tool Locations		
List of tool locations	<span>Add</span>	

## Credentials

- All NonStop user credentials have to be provided using the Jenkins Credentials.
- Refer to <https://www.jenkins.io/doc/book/using/using-credentials> for adding global credentials



## Static Analysis Tool Setup

### *SonarQube Installation (Optional)*

- Download SonarQube <https://www.sonarqube.org/downloads/>
- Refer to <https://docs.sonarqube.org/latest/setup/install-server/> for installing SonarQube
- Install SonarQube on same system as Jenkins Master Node
- Start SonarQube and make sure it is accessible (default location - <http://localhost:9000> )

## Artifact Repository Tool Setup

### *Nexus Artifact Installation (Optional)*

- Download the Nexus Artifact Repository from <https://www.sonatype.com/nexus/repository-oss>
- Install the repository on same system as Jenkins Master Node
- Start the artifact repository and make sure it is accessible (default location <http://localhost:8081>)
- To deploy application binary to Nexus Artifact Repository Update the settings.xml in maven with nexus credentials as given below

```
<server>
  <id>nexus</id>
  <username>admin</username>
  <password>XXXXXX</password>
</server>
```

## Continuous Integration Pipeline

While Tools are necessary, another important aspect of DevOps is connecting the tools together such that environment of different stages like code, build, test, release, deploy are automated with no manual intervention.

Pipelines conceptually connect the various stages together seamlessly and the key to creating pipelines is the continuous integration and continuous deployment tools

## Pipelines in Jenkins

Pipelines in Jenkins requires a set of plug-in and enable to connect the various stages of build through some code. This code can be either declarative or scripted.

To understand the concept of pipelines go through <https://www.jenkins.io/doc/book/pipeline/>

To get started with creating pipelines go through <https://www.jenkins.io/doc/book/pipeline/getting-started/>

### Descriptive versus Scripted Pipelines

Both descriptive and scripted Pipelines have their own advantages. For projects that involve multiple developers contributing to the same end product, it is recommended to use declarative Pipelines for a Developer's workflow and Scripted Pipelines can be stored along with the product in SCM for Product Nightly/Weekly build workflow.

### Github Access in a Pipeline and automatic build trigger

The builds can be automatically triggered in Jenkins after a SCM check-in. To do this GitHub Hooks or Poll SCM build triggers can be setup. Refer to <https://plugins.jenkins.io/git/> for details

### GitHub Access and credentials

While SSH is the best mechanism to authenticate to SCM. If HTTPS access is required for some reason, personal access token can be used in a declarative pipeline.

For this refer to <https://docs.github.com/en/github/authenticating-to-github/creating-a-personal-access-token> to setup the Personal Access Token.

The SCM link should be setup in the following format

<https://user:<PersonalToken>@github.<YourOrg>.com/user/<YourRepo>.git>

## Pipeline sample for SCM Access using HTTPS

Sample code snippet of providing the SCM link in a Jenkins Pipeline

```
pipeline {
    agent any
    stages {
        stage('Preparation:SCM Activity') {
```

```

        steps {
            git 'https://devops-
user:432c909d9851092fadb9895625fb926ad89616e @github.hpe.com/ devops-
user/NSDevOpsJavaQuickStart.git/'
        }
    }
}

```

In the above example `devops-user` is the SCM user and `432c909d9851092fadb9895625fb926ad89616e` is the Personal Access Token.

## Pipeline sample for NonStop Node Access

In the Jenkins Configuraiton Section, detailed steps were given to setup NonStop as a node. The sample code snippet below shows how to run a sample command on the NonStop Node using ssh.

```

node {

    def remote = [:]
    remote.credentialsId = '8003acb0-336d-41b3-b820-edce541afb0p'
    remote.name = 'cfot'
    sshagent (credentials: ['remote']) {
        sh 'pwd'
    }
}

```

In the above example `cfot` is the Managed Node and `'8003acb0-336d-41b3-b820-edce541afb0p'` is the credentials ID created for accessing the node. Note that for different actions such as build and testing, different credentials IDs could be used. The script need to provide ID that needs to be used.

## Continuous Deployment Setup

### Ansible

Ansible is an open-source software provisioning, configuration management, and application-deployment tool enabling infrastructure as code. Ansible's orchestration allows you to define your infrastructure once and use it wherever and however you need in a multi-node production.

To setup Ansible on a Linux Server (controller node), refer a section called “Installing and Configuring NSMF” in Nonstop Manageability Framework (NSMF) 1.1 reference manual.

For more details on Ansible refer to link <https://www.ansible.com>

### NSMPT

The NonStop Manageability plugins for TS/MP (NSMPT) is a python module which contains the python scripts and manifest files to manage the TS/MP products. NSMPT follows and implements the declarative configuration and management standards. NSMPT provides manifest file for each subsystem. The manifest file and NSMPT performs the requested action and return the result.

### NSMF

The NonStop Manageability Framework (NSMF) provides solutions to automate common operational tasks such as install, deploy, start, stop, restart, configure, and check the status of software infrastructure using Ansible.

NSMF uses NSMPT which provides TS/MP manageability such as automating operational tasks for ACS Subsystem, Pathmon, Pathway and Serverclass entities.

### Requisite Hardware and Software

NonStop Server	T1155L01^AAB – NSMF (Nonstop Manageability Framework)
	T0977L01^AAK – NSMPT (Nonstop Manageability Plugins for TS/MP)
	T1156L01^AAB – Python Modules & Toolkit
	T0993L01 – Python3
	SUT L20.05 and above
Linux Server	Ubuntu version 16.04.1 LTS or later
	Ansible 2.9.x, not later than 2.9.
	Python version 2.7.12 or later

### Installation on NonStop Server (Managed Node)

Place T1156PAX on NonStop Server at \$SYSTEM.ZOSSUTL

Place T0977PAX on NonStop Server at \$SYSTEM.ZUTIL

## SSH access to NonStop

Ansible is agentless and uses SSH connection to manage managed node. Hence it is essential to setup SSH access. To know how to setup this, refer a section called “Setting up Password-free Logins” in NSMF 1.1 reference manual.

## Installation on Linux Server (Controller Node)

Untar T1155TAR (NSMF) file on Linux Server. These contents are referred as NSMF\_HOME. To configure and setup NSMF kindly refer NSMF 1.1 Reference Manual and README packaged along with T1155L01^AAB.

After configuring and setting up Ansible and NSMF invoke the following Ansible command from <NSMF\_HOME> to install dependencies and NSMF product on the NonStop Server.

```
ansible-playbook install_nonstop_plugins.yml <-i inventory file if any> -v
```

## Ansible and Jenkins Integration

Ansible when used in Jenkins pipeline makes continuous deployment a reality. While Ansible allows re-use of roles and playbooks for provisioning, Jenkins acts as the process orchestrator and Ansible acts as the infrastructure orchestrator. To use Ansible in Jenkins Pipeline, download and install the Ansible Plug-in. For more details refer to <https://www.jenkins.io/doc/book/managing/plugins>

## Mapping Ansible Node in Jenkins

Mapping Ansible Node in Jenkins is only required if the Jenkins Master Node is not on the same Linux machine as Ansible. Follow the instructions below to Map to Linux Node to Jenkins. A Jenkins agent will run on the Linux machine that hosts Ansible and will communicate through SSH. Ensure SSH access to Linux is available.

- Click on Manage Jenkins → Click on Manage Nodes section
- Create a new node
- Enter a node name
- Whether the node will run permanently - (Yes)
- Node Name
- Number of executors
- Root Directory where the agent will run
- Labels (to group one or more Linux Systems or to group systems based on functionality/usage)
- Usage details – Use this node as much as possible
- Launch Method - Launch via SSH
- Host IP

- Credentials (Note – the credential should be added via Jenkins Credentials Only)
- Host Key Verification Strategy – Manually Trusted Key Verification
- Click on Advance Tab and set the following
- SSH port - retain the SSH port as 22
- Set the Java path
- Update the number of retries and connection timeout values
- Availability – Keep the agent online as much as possible
- Provide Environment variables if required.

## SSH setup for playbook

Refer this link <https://www.jenkins.io/doc/book/using/using-credentials/> or credentials section of the document to add SSH credentials and section of SSH Username with private key

Snippet – How to add SSH credentials for a node to connect using SSH.



The screenshot shows the Jenkins 'Add Credentials' page. The breadcrumb trail is 'Jenkins > Credentials > System > Global credentials (unrestricted)'. On the left, there are links for 'Back to credential domains' and 'Add Credentials'. The main form is titled 'SSH Username with private key'. It includes a 'Kind' dropdown set to 'SSH Username with private key', a 'Scope' dropdown set to 'Global (Jenkins, nodes, items, all child items, etc)', and input fields for 'ID', 'Description', 'Username', 'Private Key' (with a radio button for 'Enter directly'), and 'Passphrase'. An 'OK' button is at the bottom.

As shown in the above snippet,

- Select SSH Username with private key from the drop down list.
- Scope can be Global.
- ID can be given as per the user's choice, if left blank then automatically generated by Jenkins
- Description is optional, which gives more information about the credentials are for what
- Username for whom this SSH credentials are created
- Private Key is SSH private key

After adding all the required details, it will look as shown below

Scope

Global (Jenkins, nodes, items, all child items, etc)

?

ID

8973acb0-336d-41b3-b820-edce541afb0d

?

Description

?

Username

dev

Private Key

☒ Enter directly

Key

 Concealed for Confidentiality

Replace

Passphrase

## Ansible playbook execution from Jenkins Pipeline

To run a playbook, the pre-requisites are controller node setup, Refer to a section “Mapping Ansible Node” in Jenkins above. Given below is a sample code snippet to demonstrate how to run a playbook from controller node using ssh.

```
node('controller') {
    ansiColor('xterm') {
        def remote = [:]
        remote.credentialsId = '8973acb0-336d-41b3-b820-edce541afb0d',
        remote.name = 'dev'
        stage('Deliver and Deploy in DEV Environment') {
            sshagent (credentials: ['remote']) {
                ansiblePlaybook(
                    playbook: 'sample.yml',
                    inventory: 'inventory/dev/hosts',
                    credentialsId: '8973acb0-336d-41b3-b820-edce541afb0d',
                    extras: '-v')
            }
        }
    }
}
```

In the above example ‘controller’ is the Controller Node and “8973acb0-336d-41b3-b820-edce541afb0d” is the credentials ID created for accessing the node. Note that for different actions such as build and testing, different credentials IDs could be used. The script need to provide ID that needs to be used.

ansiblePlaybook is the block which executes playbook on the managed node, that is NonStop Node from the controller node. As shown in the above example –

playbook – name of the playbook to be executed

inventory – name of the inventory file

credentialsId – SSH credentials to connect to the NonStop node from the controller node

extras – any other extra parameters user need to provide to the playbook execution

For more information on syntax refer at: <https://www.jenkins.io/doc/pipeline/steps/ansible/>



## Java Starter Kit

The Java Starter Kit package is designed to demonstrate the Continuous Integration of a Purely Platform Agnostic Application profile of NonStop DevOps through a javaquickstart application.

The setup, tooling, gating criteria and artifacts in this package can be used to automate the Continuous Integration Phase of any Java Project. This package acts as a starter kit for users.

This is a beginner level starter kit.

## Package Information

The Java Starter Kit package consists on a javaquickstart maven project with a simple client and server Java application and a README file.

This package does not contain the pre-requisite software for Continuous Integration Setup. The pre-requisite software have to be downloaded and installed as per the instructions in the above section.

### Pre-requisites software

The following are the pre-requisites on the system identified for the Java Starter Kit.

Software	Description	Version
Jenkins	Build Management System	2.222.3 or later
GITHUB	Source Code Control	
GIT	Source Code Control	2.26.2-64-bit or later
Java	Java Development and Runtime	1.8.0_181 or later version of JDK 8.
Maven	Standard Build Tool	3.6.3 or later
SonarQube	Static Analysis Tools	8.3.0.xx or later
Junit	Unit Testing Framework	1.5.1 (included in the package)
Nexus Repository Manager	Binary Artifact Repository	3.17.0.01 or later

## Using the Starter Kit

The Java Starter Kit requires SCM, Jenkins, SonarQube and Artifact Repository. Perform the setup using the instructions set provided in the last section.

### Downloading the project from HPE NonStop github

Log on to GITHUB and browse to <https://github.com/HewlettPackard/NonStop>

Click on Clone and download the HPE NonStop Samples as ZIP file using Download Zip option

Unzip the nsdevops folder that contains the NonStop DevOps starter kits.

Browse to the java for the Java Starter Kit.

## Uploading the project to SCM

Make sure that you can log on to <https://github.<yourcompanyname>.com/>

Create a new repository in GITHUB (NSDevOpsJavaQuickStart)

Copy the “java” folder contents into a folder of your choice.

Using the GIT CLI upload the java quick start to the github using the following commands

```
git init
git add *
git commit * -m "<Message>"
git remote add origin https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsJavaQuickStart>.git
git push -u origin master
```

where

NSDevOpsJavaQuickStart is the name of your GITHUB repository

devops-user is the username

<your-personal-token> is the SCM Personal Token

github.<yourcompanyname>.com is the GitHub repository

Replace the items mentioned within <> and marked in red font with values appropriate to your setup.

## Continuous Integration Pipelines using Java Starter Kit

### Developer Pipeline

This workflow triggers when a developer checks-in the code. The pipeline performs scm checkout, build, checkstyle analysis using sun\_checks.xml and unit testing using Junit.

The required software are packaged into nsdevopsjavaquickstart .

Since many developers may be involved in development and each of their remote NonStop Node and environments vary, this workflow is given as a scripted pipeline job.

A template could be created for the project and individual developers can use the template and update their specific environment details.

### Jenkins Job Setup

- Create a Jenkins Pipeline Job SampleDeveloperPipeline
- In the General tab
  - Provide the Description
  - Select GitHub Project and provide the SCM Link
 

```
https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsJavaQuickStart>.git
```

Update the items marked in red with values appropriate to your setup.

Note – To access GIT, the Jenkins Master will required access to

The screenshot shows the Jenkins configuration page for 'SampleDeveloperPipeline'. The 'General' tab is selected. The 'Description' field contains the following text: 'Automated build triggered (Jenkins PollSCM build trigger)', 'Automated code check-out (by Jenkins from GITHUB)', 'Automated Build (Jenkins Pipeline Script and build automation tools)', 'Binaries copied to NonStop', 'Static Analysis of code (CheckStyle)', and 'Unit Tests run on NonStop (JUnit)'. Below the description, there are several checkboxes: 'Discard old builds', 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the master restarts', 'GitHub project' (checked), 'Pipeline speed/durability override', 'Preserve stashes from completed builds', 'This project is parameterized', and 'Throttle builds'. The 'Project url' field is set to 'https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsJavaQuickStart>.git'. At the bottom, there are 'Save' and 'Apply' buttons.

- In the BuildTriggers Tab,
  - enable the PollSCM option and provide the schedule how often the SCM has to poll. Providing \* \* \* \* \* will poll every minute.
  - Note the spaces in the above pattern.

The screenshot shows the Jenkins configuration page for 'SampleDeveloperPipeline', specifically the 'Build Triggers' tab. The 'Build Triggers' section has the following options: 'Build after other projects are built', 'Build periodically', 'GitHub hook trigger for GITScm polling', and 'Poll SCM' (checked). The 'Schedule' field is set to '\* \* \* \* \*'. Below the schedule field, there is a warning message: 'Do you really mean "every minute" when you say "\* \* \* \* \*" ? Perhaps you meant "H \* \* \* \*" to poll once per hour. Would last have run at Tuesday, December 15, 2020 1:02:25 PM IST; would next run at Tuesday, December 15, 2020 1:02:25 PM IST.' There are also checkboxes for 'Ignore post-commit hooks', 'Disable this project', 'Quiet period', and 'Trigger builds remotely (e.g., from scripts)'. At the bottom, there are 'Save' and 'Apply' buttons.

- In the Advanced Project Options,
  - Under Pipeline Definition, choose the Pipeline Script option.

- Copy paste the script below after making appropriate changes to the repositories, node name, Jenkins workspace location \* everything marked in red in the script below
- Please note the node names are case sensitive. Use the names as per the Jenkins configuration

```
properties(
  [
    [
      $class: 'BuildDiscarderProperty',
      strategy: [$class: 'LogRotator', numToKeepStr: '10']
    ],
  ]
)
node ('master') {
  def mvnHome
  def remote = [:]
  remote.name = '<nsdev>'
  remote.host = '<XX.YYY.ZZ.AA>'
  remote.user = '<devops.user>'
  remote.password = '<XXXX>'
  remote.allowAnyHosts = true
  stage('Code Checkout') { // for display purposes
    // Get some code from a GitHub repository
    git 'https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsJavaQuickStart>.git/'
    // Get the Maven tool.
    // ** NOTE: This 'M3' Maven tool must be configured
    // **          in the global configuration.
    mvnHome = tool 'Maven'
  }
  stage('Automated Build') {
    // Run the maven build.
    // If Jenkins master is behind a firewall use the
    // -DproxySet=true -DproxyHost=<host> -DproxyPort=<port>
    withEnv(["MVN_HOME=$mvnHome"]) {
      if (isUnix()) {
        sh "$MVN_HOME/bin/mvn" clean package
      } else {
        bat("/%M2_HOME%\bin\mvn" clean package /)
      }
    }
  }
  stage('Static Analysis') {
    // If Jenkins master is behind a firewall use the
```

```
// -DproxySet=true -DproxyHost=<host> -DproxyPort=<port>

    if (isUnix()) {
        sh "$MVN_HOME/bin/mvn" checkstyle:checkstyle'
    } else {
        bat ("%M2_HOME%\bin\mvn" checkstyle:checkstyle /)
    }
}

stage('Binaries To NonStop') {
    //Replace C:/Applications/Jenkins/ to the location where
    //Jenkins is installed in your system
    sshCommand remote: remote, command: "mkdir -p
/home/devopsuser/javaquickstart"
    sshPut remote: remote, from: 'C:/Applications/Jenkins/workspace/
SampleDeveloperPipeline/server/target/javahelloserver-1.0.0-
SNAPSHOT.jar', into:
'/home/devopsuser/javaquickstart/javahelloserver-1.0.0-SNAPSHOT.jar'
    sshPut remote: remote, from: 'C:/Applications/
Jenkins/workspace/SampleDeveloperPipeline/junit-platform-console-
standalone-1.5.1.jar' , into:
'/home/devopsuser/javaquickstart/junit-platform-console-standalone-
1.5.1.jar'
    sshCommand remote: remote, command: "cd /home/devopsuser
/javaquickstart;ls -lrt"
}
stage('Unit Test') {
    sshCommand remote: remote, command: "cd
/home/devopsuser/javaquickstart;/usr/tandem/nssjava/jdk180_180/bin/j
ava -jar junit-platform-console-standalone-1.5.1.jar -cp
javahelloserver-1.0.0-SNAPSHOT.jar -c
com.hpe.nsk.nsdevops.server.JavaHelloServer"
}
} //node
```

**Note :** Update the items marked in red in the above syntax with values appropriate to your setup

Jenkins > SampleDeveloperPipeline >

General Build Triggers **Advanced Project Options** Pipeline

**Advanced Project Options**

Advanced...

**Pipeline**

Definition Pipeline script

Script

```

17 }
18 stage('Automated Build') {
19     // Run the maven build
20     withEnv(["MVN_HOME=${mvnHome}"]) {
21         if (isUnix()) {
22             sh "'$MVN_HOME/bin/mvn' clean package"
23         } else {
24             bat("%MVN_HOME%\bin\mvn" package /)
25         }
26     }
27 }
28
29 stage('Static Analysis') {
30     bat("%MVN_HOME%\bin\mvn" checkstyle:checkstyle /)
31 }
32
33

```

☒ Use Groovy Sandbox

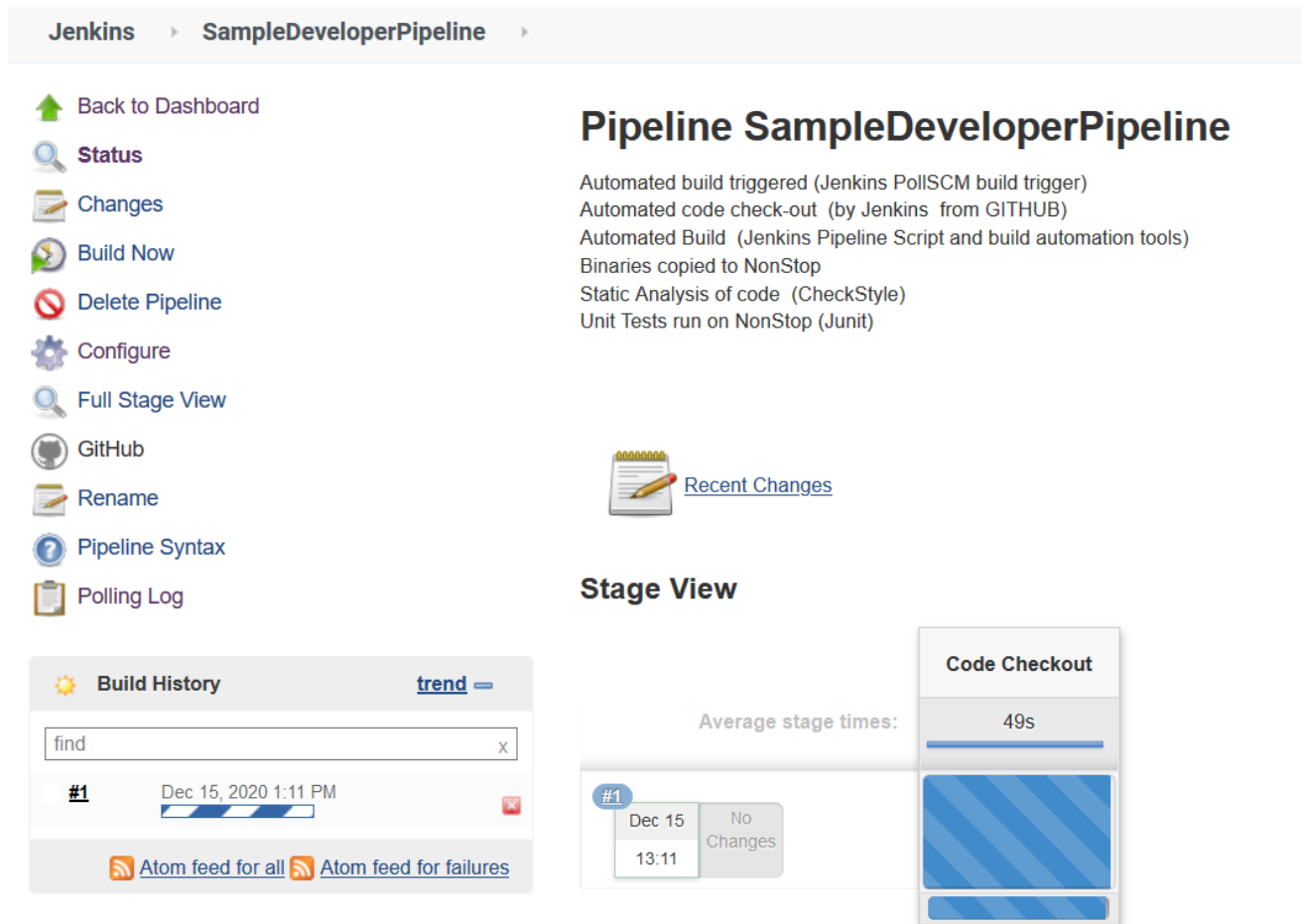
[Pipeline Syntax](#)

- Apply the changes and Save the Job.

## Jenkins Job Setup

Since the Poll SCM option is turned on, when the developer makes a change and commits the code, the build is triggered. The Developer can monitor the Job through the Jenkins Job Console.

Alternately, the job can be triggered manually by clicking the Build Now option in the Jenkins SampleDeveloperPipeline



The screenshot shows the Jenkins interface for the **SampleDeveloperPipeline**. The breadcrumb navigation at the top reads **Jenkins > SampleDeveloperPipeline**.

**Left Sidebar (Actions):**

- [Back to Dashboard](#)
- [Status](#)
- [Changes](#)
- [Build Now](#)
- [Delete Pipeline](#)
- [Configure](#)
- [Full Stage View](#)
- [GitHub](#)
- [Rename](#)
- [Pipeline Syntax](#)
- [Polling Log](#)

**Main Content Area:**

### Pipeline SampleDeveloperPipeline

Automated build triggered (Jenkins PollSCM build trigger)  
 Automated code check-out (by Jenkins from GITHUB)  
 Automated Build (Jenkins Pipeline Script and build automation tools)  
 Binaries copied to NonStop  
 Static Analysis of code (CheckStyle)  
 Unit Tests run on NonStop (JUnit)

[Recent Changes](#)

### Stage View

**Build History** (trend =)

find x

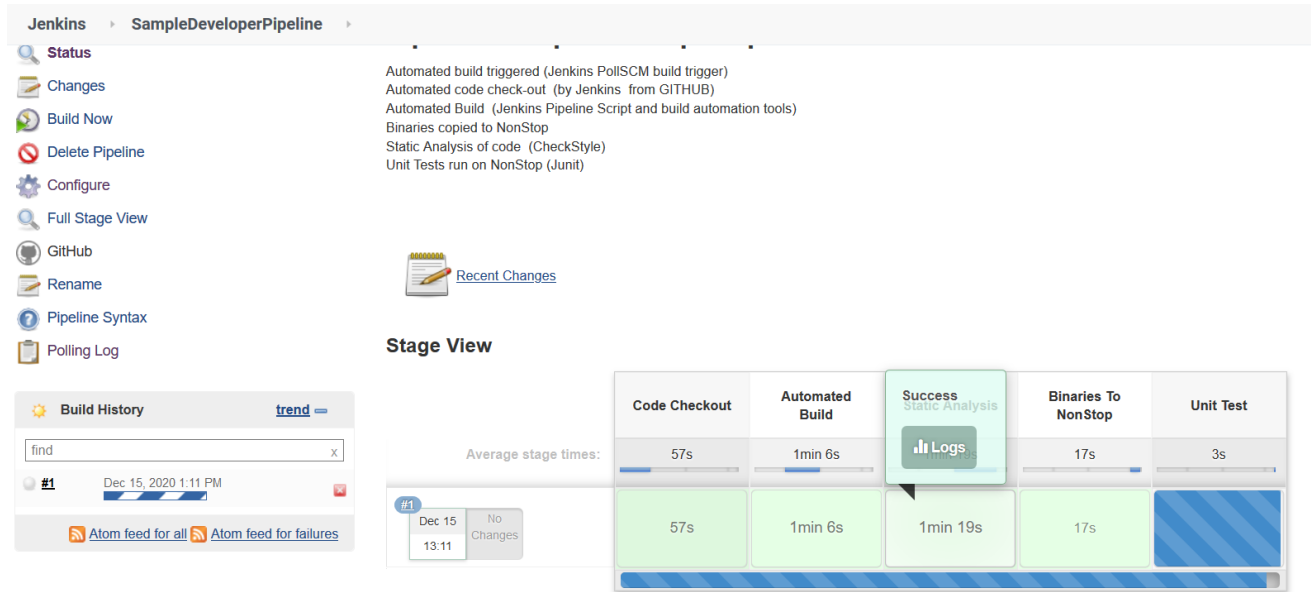
#1 Dec 15, 2020 1:11 PM

[Atom feed for all](#) [Atom feed for failures](#)

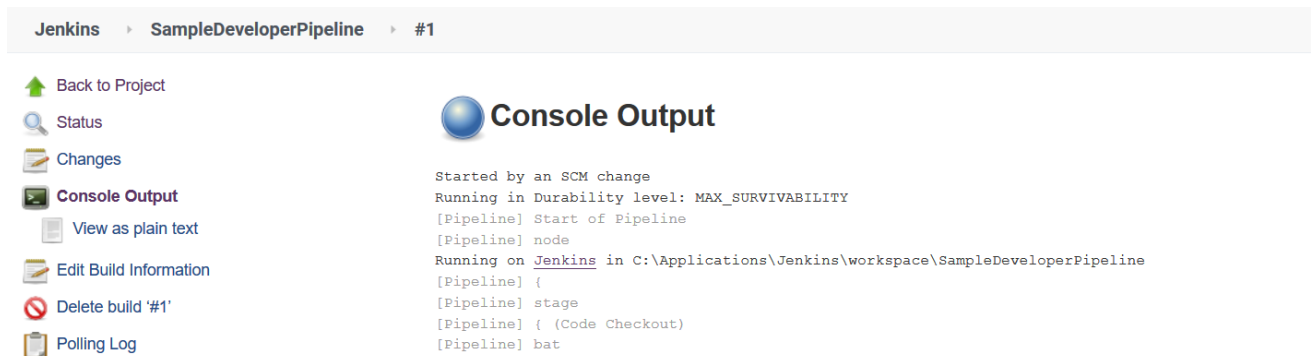
**Stage Details:**

- Code Checkout:** 49s
- Average stage times:**
- #1:** Dec 15 13:11, No Changes

Each stage can be monitored by looking at the stage and logs per stage can be viewed.



## Sample Console Output for the build



Note – In this sample, the master branch has been used. However, in a typical application development scenario, the developers might work on specific branches. These can be integrated with the main/master branch if the build succeeds. This step can be automated in the developer pipeline. So that Nightly Build can be triggered.

## Nightly Build Pipeline

This workflow triggers at a given time of the day. The pipeline performs scm checkout, build, checkstyle analysis using sun\_checks.xml, static analysis using SonarQube, unit testing using JUnit and uploads the binaries into artifact repository.

Ensure SonarQube and Nexus Repository Manager are up and accessible.

### Jenkins Job Setup

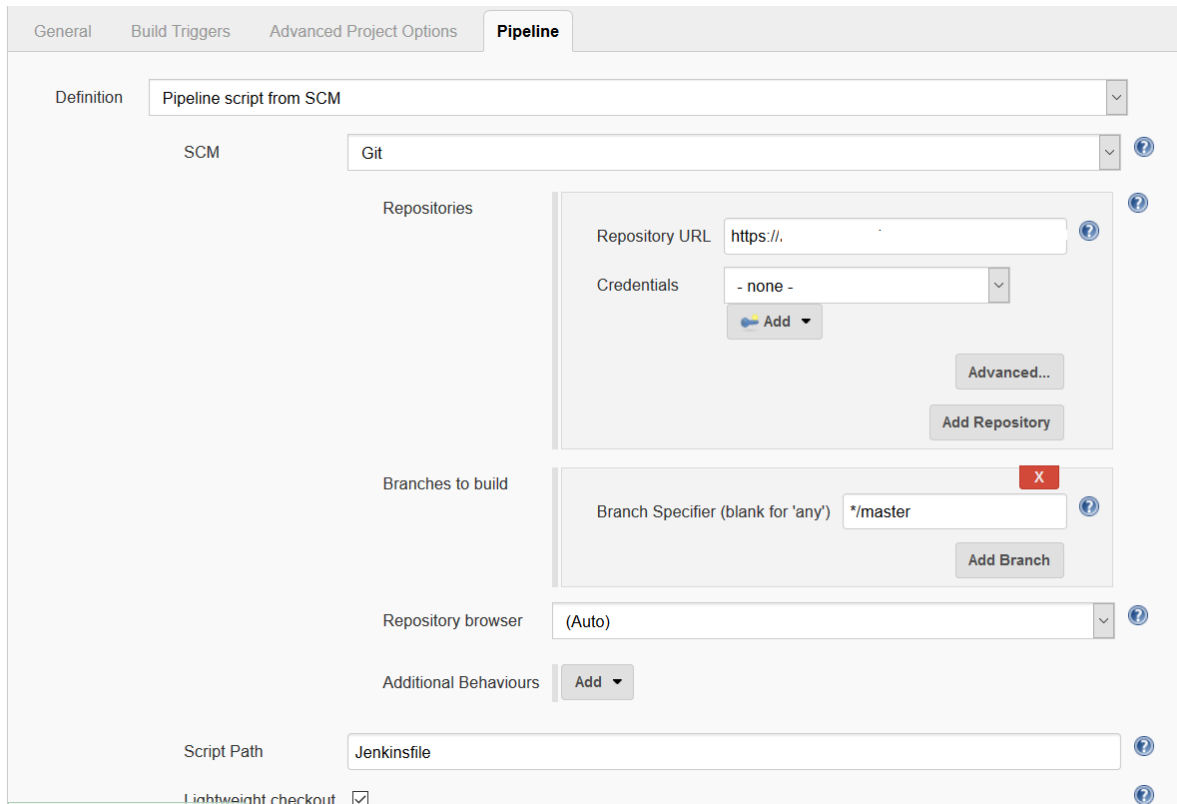
- Create a Jenkins Pipeline Job SampleNightlyBuildPipeline
- In the General tab
  - Provide the Description
  - Select GitHub Project and provide the SCM Link



```
https://<devops-user>:<your-personal-
token>@github.<yourcompanyname>.com/<devops-
user>/<NSDevOpsJavaQuickStart>.git
```

Update the items marked in red with values appropriate to your setup.

- In the BuildTriggers Tab,
  - Select the Build periodically option and provide the schedule how often the Job be should run. Providing H 10 \* \* \* will run the job at 10 am every day.
  - Note the spaces in the above pattern.
- In the Advanced Project Options,
- Under Pipeline Definition, choose the Pipeline Script from SCM option.
  - Select the SCM as GIT
    - Provide the GIT repository URL (`https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsJavaQuickStart>.git`)
    - Since the Git Authentication is based on Personal Token, select “none” for Repository Credentials.
    - Select /master for the branches to build and Auto for repository browser
  - Select script path as `Jenkinsfile`



- Apply the changes and Save the Job.

## Jenkins Job Setup

Since the Build Periodically option is turned on, when the build get triggered automatically at the specified time. The Developer can monitor the Job through the Jenkins Job Console.

Alternately, the job can be triggered manually by clicking the Build Now option in the Jenkins SampleNightlyBuildPipeline

The Java Starter Kit is provided as a sample. The pipelines can be modified and additional such as copying tests cases from repository and test environment setup can be added to SampleNightlyBuildPipeline to execute regressions tests for your application release pipelines.

## Java JNI Starter Kit

The Java JNI Starter Kit package is designed to demonstrate the Continuous Integration of a Partially Platform Agnostic Application profile of NonStop DevOps through a javajniquickstart application.

The setup, tooling, gating criteria and artifacts in this package can be used to automate the Continuous Integration Phase of any Java JNI Project. This package acts as a starter kit for users.

This is an intermediate level starter kit.

## Package Information

The Java JNI Starter Kit package consists on a javajniquickstart maven project with a simple client and server Java JNI application and a README file.

This package does not contain the pre-requisite software for Continuous Integration Setup. The pre-requisite software have to be downloaded and installed as per the instructions in the above section.

### Pre-requisites software

The following are the pre-requisites on the system identified for the Java Starter Kit.

Software	Description	Version
Jenkins	Build Management System	2.222.3 or later
GITHub	Source Code Control	
GIT	Source Code Control	2.26.2-64-bit or later
Java	Java Development and Runtime	1.8.0_181 or later version of JDK 8. (Don't use JDK 9 or 11)
Maven	Standard Build Tool	3.6.3 or later
Boost	Testing Framework	1.5.6 or later
Junit	Unit Testing Framework	1.5.1 (included in the package)

## Using the Starter Kit

The Java Starter Kit requires SCM, Jenkins. Perform the setup using the instructions set provided in the last section. Please note, when SCM is setup it is important to select to select Unix-style line endings as mentioned earlier. This is because, if this option is not selected, the Makefile when moved to NonStop does not have ^M characters which will cause errors. Hence, make sure to have Unix-style line endings.

### Downloading the project from HPE NonStop github

Log on to GITHUB and browse to <https://github.com/HewlettPackard/NonStop>

Click on Clone and download the HPE NonStop Samples as ZIP file using Download Zip option

Unzip the nsdevops folder that contains the NonStop DevOps starter kits.

Browse to the javajni for the Java JNI Starter Kit.

### Uploading the project to SCM

Make sure that you can log on to <https://github.<yourcompanyname>.com/>

Create a new repository in GITHUB (NSDevOpsJavaJNIQuickStart)

Copy the “javajni” contents into a folder of your choice.

Using the GIT CLI upload the javajni starter kit to the github using the following commands

```
git init
git add *
git commit * -m "<Message>"
git remote add origin https://<devops-user>:<your-personal-
token>@github.<yourcompanyname>.com/<devops-
user>/<NSDevOpsJavaJNIQuickStart>.git
git push -u origin master
```

where

NSDevOpsJavaJNIQuickStart is the name of your GITHUB repository

devops-user is the username

<your-personal-token> is the SCM Personal Token

github.<yourcompanyname>.com is the GitHub repository

Replace the items mentioned within <> and marked in red font with values appropriate to your setup.

## Continuous Integration Pipelines using Java JNI Starter Kit

### JNIDeveloper Pipeline

This workflow triggers when a developer checks-in the code. The pipeline performs scm checkout, builds on NonStop Node and unit testing using Junit & Boost Framework

The required software are packaged into nsdevopsjavajniquickstart .

Since many developers may be involved in development and each of their remote NonStop Node and environments vary, this workflow is given as a scripted pipeline job.

A template could be created for the project and individual developers can use the template and update their specific environment details.

#### Jenkins Job Setup

- Create a Jenkins Pipeline Job JNIDeveloperPipeline
- In the General tab
  - Provide the Description
  - Select GitHub Project and provide the SCM Link
 

```
https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsJavaJNIQuickStart>.git
```

Update the items marked in red with values appropriate to your setup.

- In the BuildTriggers Tab,
  - Enable the PollSCM option and provide the schedule how often the SCM has to poll. Providing \* \* \* \* \* will poll every minute.
  - Note the spaces in the above pattern.
- In the Advanced Project Options,
  - Under Pipeline Definition, choose the Pipeline Script option.
  - Copy paste the script below after making changes to the repositories, node name appropriately.
  - Please note the node name is case sensitive. Use the name as per the Jenkins configuration

```
properties(
    [
        [
            $class: 'BuildDiscarderProperty',
            strategy: [$class: 'LogRotator', numToKeepStr: '10']
        ],
    ]
)
node ('master') {
    def mvnHome
    def remote = [:]
    remote.name = '<nsdev>'
    remote.host = '<XX.YYY.ZZ.AA>'
```

```
remote.user = '<devops.user>'
remote.password = '<XXXX>'
remote.allowAnyHosts = true
stage('Code Checkout') { // for display purposes
    // Get some code from a GitHub repository
    git 'https://<devops-user>:<your-personal-
token>@github.<yourcompanyname>.com/<devops-
user>/<NSDevOpsJavaJNIQuickStart>.git/'
    mvnHome = tool 'Maven'
}
stage('Copy To NonStop, Build & Test') {

    sshCommand remote: remote, command: "mkdir -p
/home/devopsuser/javajniquickstart"
    sshPut remote: remote, from: 'C:/Applications/Jenkins/workspace/
JNIDeveloperPipeline/', into: '/home/devopsuser/javajniquickstart'

    sshCommand remote: remote, command: "cd
/home/devopsuser/javajniquickstart; make"
}

} //node
```

Note : Update the items marked in red in the above syntax with values appropriate to your setup

- Apply the changes and Save the Job.

### Jenkins Job Setup

Since the Poll SCM option is turned on, when the developer makes a change and commits the code, the build is triggered. The Developer can monitor the Job through the Jenkins Job Console.

Alternately, the job can be triggered manually by clicking the Build Now option in the Jenkins JNIDeveloperPipeline

Each stage can be monitored by looking at the stage and logs per stage can be viewed.

Note – In this sample, the master branch has been used. However, in a typical application development scenario, the developers might work on specific branches. These can be integrated with the main/master branch if the build succeeds. This step can be automated in the developer pipeline. So that Nightly Build can be triggered.

## Continuous Deployment starter kit

The Continuous Deployment Starter Kit package is designed to demonstrate the Continuous Delivery and Deployment of a Purely Platform Agnostic Application profile of NonStop DevOps through a *bankapp* application.

The setup, tooling and artifacts in this package can be used to automate the Continuous Delivery and Deployment Phase of any Project. This package acts as a starter kit for users.

This is an intermediate level starter kit.

### Package Information

The Continuous Deployment Starter Kit package consists of a sample application called BankApp (bankapp-1.0.jar executable jar file). BankApp is a java based server and client application. The starter kit also consists of Ansible artifacts used in this kit and a README file.

This package does not contain the pre-requisite software for Continuous Deployment Setup. The pre-requisite software have to be downloaded and installed as per the instructions in the above section.

#### Pre-requisites software

The following are the pre-requisites on the system identified for the Continuous Deployment Starter Kit.

Software	Description	Version
Jenkins	Build Management System	2.222.3 or later
GITHub	Source Code Control	
Nexus Repository Manager	Binary Artifact Repository	3.17.0.01 or later
Ansible	Continuous Deployment Tool	2.9.11 or later

## Using the Starter Kit

The Continuous Deployment Starter Kit requires SCM, Jenkins, Ansible, NSMF and Artifact Repository. Perform the setup using the instructions set provided in the previous section.

#### Updating the Ansible Artifacts for your environment

Refer to [CDStarterKitUsageInstructions.docx](#) for package details and changes required in the Ansible Artifacts folder in yml and hosts scripts to customize it to your environment.

#### Uploading the project to SCM

Make sure that you can log on to <https://github.<youcompanyname>.com/>

Create a new repository in GITHUB (NSDevOpsCDQuickStart)

Copy the content of “cd/Ansible artifacts/BankApp” into a folder of your choice.

Using the GIT CLI upload the “Ansible artifacts” to the github using the following commands

```
git init
git add *
git commit * -m "<Message>"
git remote add origin https://<devops-user>:<your-personal-
token>@github.<yourcompanyname>.com/<devops-user>/<
NSDevOpsCDQuickStart>.git
git push -u origin master
```

where

NSDevOpsCDQuickStart is the name of your GITHUB repository

devops-user is the username

<your-personal-token> is the SCM Personal Token

github.<yourcompanyname>.com is the GitHub repository

Replace the items mentioned within <> and marked in red font with values appropriate to your setup.

## Continuous Deployment Pipeline using Starter Kit

In DevOps, this phase is broadly addressed into development, staging and production environments. This is demonstrated by a single Jenkins pipeline called Application Release Pipeline.

This pipeline could be triggered once new artifacts such as binaries are available in the Nexus artifact repository. As soon as new version or updated application is available on the URL, this pipeline could be triggered. For more information on this refer to <https://plugins.jenkins.io/urltrigger/>

## Application Release and Deployment Pipeline

This pipeline demonstrates the following workflow

1. Deliver and Deploy an application in DEV environment
2. Run unit tests
3. Promote an application to staging
4. Deliver and Deploy an application in QA environment
5. Run QA tests
6. Promote an application to production
7. Deliver and Deploy an application in PRODUCTION environment



## Jenkins Job Setup and Pipeline Script

Once you create a pipeline job in Jenkins, while configuring the job select pipeline script for a pipeline definition. Add the following script in a script section of a pipeline, after making appropriate changes to it. To know on how to setup pipelines in Jenkins, refer a section called "Pipelines in Jenkins" of this document. Update the items marked in red with values appropriate to your setup.

```
node('master') {
    ansiColor('xterm') {
        def remote = [:]
        remote.credentialsId = '<credentialsId>'
        remote.name = 'dev'
        git credentialsId: '<credentialsId>', url: 'git@github.<your company
name>.com:<repository name>/NSDevOpsCDQuickStart.git'
        stage('Deliver and Deploy in DEV Environment') {
            sshagent (credentials: ['remote']) {

                ansiblePlaybook(
                    playbook: 'Ansible
artifacts/BankApp/bankapp.yml',
                    inventory: 'Ansible
artifacts/BankApp/inventory/dev/hosts',
                    credentialsId: '<credentialsId >',
                    hostKeyChecking: false,
                    extras: '-e sc_name=scdev -e
pathmon_name=\$dev -e role=dev -v',
                    vaultCredentialsId: '<vaultCredentialsId >',
                )
            }
        }
    }
}

node('cfot') {

    def remote = [:]
    remote.name = 'cfot'
    remote.host = 'XX.YYY.ZZ.AAA'
    remote.user = '<user name>'
    remote.password = 'XXXXXX'
    remote.allowAnyHosts = true
    stage('Unit Tests') {

        sshCommand remote: remote, command: "cd /home/dev/app; export
JAVA_HOME=/usr/tandem/nssjava/jdk180_180;/usr/tandem/nssjava/jdk180_180/bi
n/java -classpath .:bankapp-1.0.jar -Dserver.socket.port=5000
com.hpe.nsk.nsdevops.client.Client"
```

```

        echo "Unit Tests are completed"
    }
}

node ('master') {
    stage('Promote to Stage'){
        artifactPromotion artifactId: 'bankapp', debug: false, groupId:
        'com.hpe.nsk.nsdevops', extension: 'pom', promoterClass:
        'org.jenkinsci.plugins.artifactpromotion.NexusOSSPromotor', releasePW:
        'XXXXX', releaseRepository: '
        http://XX.YYY.ZZ.AAA:<Port_No>/repository/maven-releases/', releaseUser:
        'qa', stagingPW: 'XXXXX', stagingRepository: '
        http://XX.YYY.ZZ.AAA:<Port_No >/repository/maven-releases/', stagingUser:
        'dev', version: '1.0'
    }
    echo 'Application is successfully promoted to staging'
}
node ('master') {
    ansiColor('xterm') {

        def remote = [:]
        remote.credentialsId = '<credentialsId >'
        remote.name = 'qa'

        git credentialsId: '<credentialsId >', url: ' git@github.<your company
        name>.com:<repository name>/NSDevOpsQuickStart.git '
        stage('Deliver and Deploy in QA Environment') {
            sshagent (credentials: ['remote']) {
                ansiblePlaybook(
                    playbook: 'Ansible
artifacts/BankApp/bankapp.yml',
                    inventory: 'Ansible
artifacts/BankApp/inventory/qa/hosts',
                    credentialsId: '<credentialsId >',
                    hostKeyChecking: false,
                    extras: '-e sc_name=scqa -e pathmon_name=\$qa
-e role=qa -v',
                    vaultCredentialsId: '<vaultCredentialsId >'

                )
            }
        }
    }
}
node ('cfot') {
    def remote = [:]
    remote.name = 'cfot'
    remote.host = 'XX.YYY.ZZ.AAA'
    remote.user = '<user name>'

```

```

    remote.password = 'XXXXX'
    remote.allowAnyHosts = true
    stage('QA Tests') {
        sshCommand remote: remote, command: "cd /home/qa/app; export
        JAVA_HOME=/usr/tandem/nssjava/jdk180_180;/usr/tandem/nssjava/jdk180_180/bi
        n/java -classpath .:bankapp-1.0.jar -Dserver.socket.port=5001
        com.hpe.nsk.nsdevops.client.Client"
        echo "QA Tests are completed"
    }
}
node ('master') {
    stage('Promote to Production'){
        artifactPromotion artifactId: 'bankapp', debug: false, groupId:
        'com.hpe.nsk.nsdevops', extension: 'pom', promoterClass:
        'org.jenkinsci.plugins.artifactpromotion.NexusOSSPromotor', releasePW:
        'XXXXX', releaseRepository: '
        http://XX.YYY.ZZ.AAA:<Port_No>/repository/maven-releases/', releaseUser:
        'qa', stagingPW: 'XXXXX', stagingRepository: '
        http://XX.YYY.ZZ.AAA:<Port_No>/repository/maven-releases/', stagingUser:
        'dev', version: '1.0'
    }
    echo 'Application is successfully promoted to production'
}

node ('master'){
    ansiColor('xterm') {
        def remote = [:]
        remote.credentialsId = '<credentialsId >'
        remote.name = 'production'

        git credentialsId: '<credentialsId>', url: ' git@github.<your company
        name>.com:<repository name>/NSDevOpsQuickStart.git '

        stage('Deliver and Deploy in Production Environment') {
            sshagent (credentials: ['remote']) {
                ansiblePlaybook(
                    playbook: 'Ansible
                    artifacts/BankApp/bankapp.yml',
                    inventory: 'Ansible
                    artifacts/BankApp/inventory/production/hosts',
                    credentialsId: '<credentialsId >',
                    hostKeyChecking: false,
                    extras: '-e role=production -e
                    pathmon_name=\$prod -e sc_name=scprod -v',
                    vaultCredentialsId: '<vaultCredentialsId>'

                )
            }
        }
    }
}

```

```
}
}
```

Following is the snapshot of a pipeline script configured for the Jenkins job for your reference.

The screenshot shows the Jenkins Pipeline configuration interface. The 'Pipeline' tab is active, and the 'Definition' section shows a 'Pipeline script'. The script is as follows:

```
1 node('master') {
2   ansiColor('xterm') {
3     def remote = [:]
4     remote.credentialsId = '8973acb0-336d-41b3-b820-edce541afb0d'
5     remote.name = 'dev'
6     git credentialsId: 'ef83dbc5-0707-4ceb-b54a-bbd85d2300d4', url: 'git@github.hpe.com:NSDi/Bar
7
8
9   stage('Deliver and Deploy in DEV Environment') {
10     sshagent (credentials: ['remote']) {
11
12
13
14     ansiblePlaybook(
15       playbook: 'bankapp.yml',
16       inventory: 'inventory/dev/hosts'
17     )
18   }
19 }
```

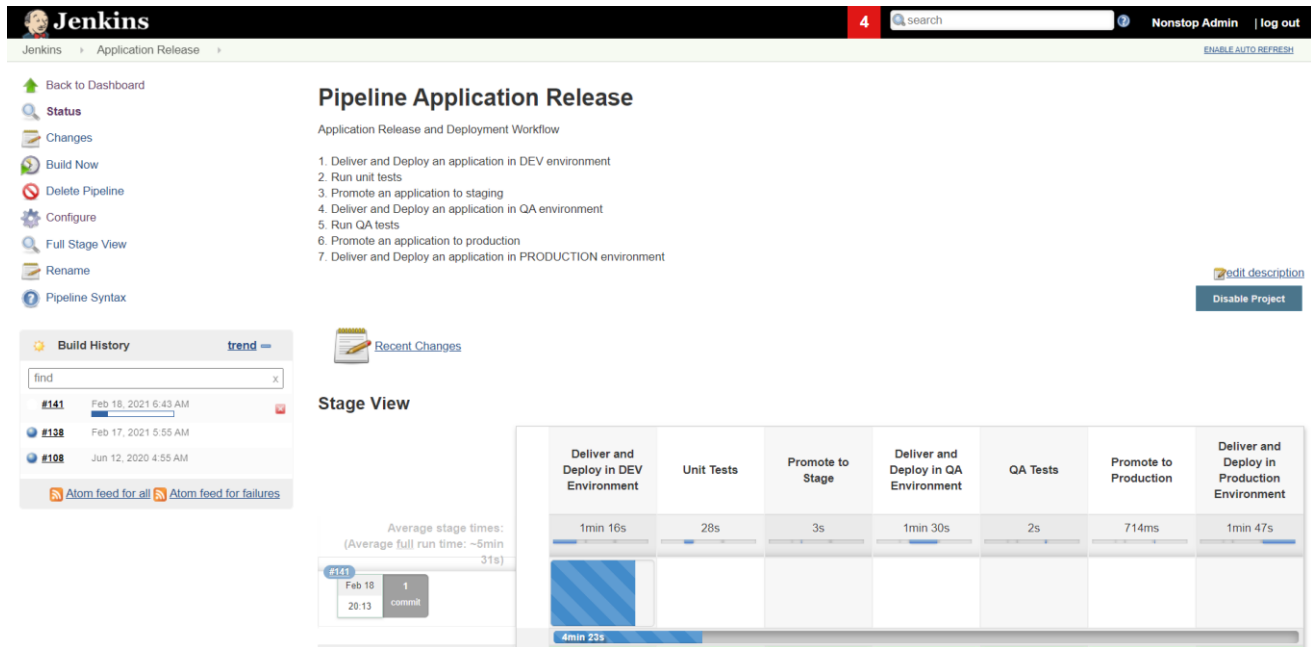
Below the script, the 'Use Groovy Sandbox' checkbox is checked. A link for 'Pipeline Syntax' is also visible.

### Validation of Jenkins Pipeline Job

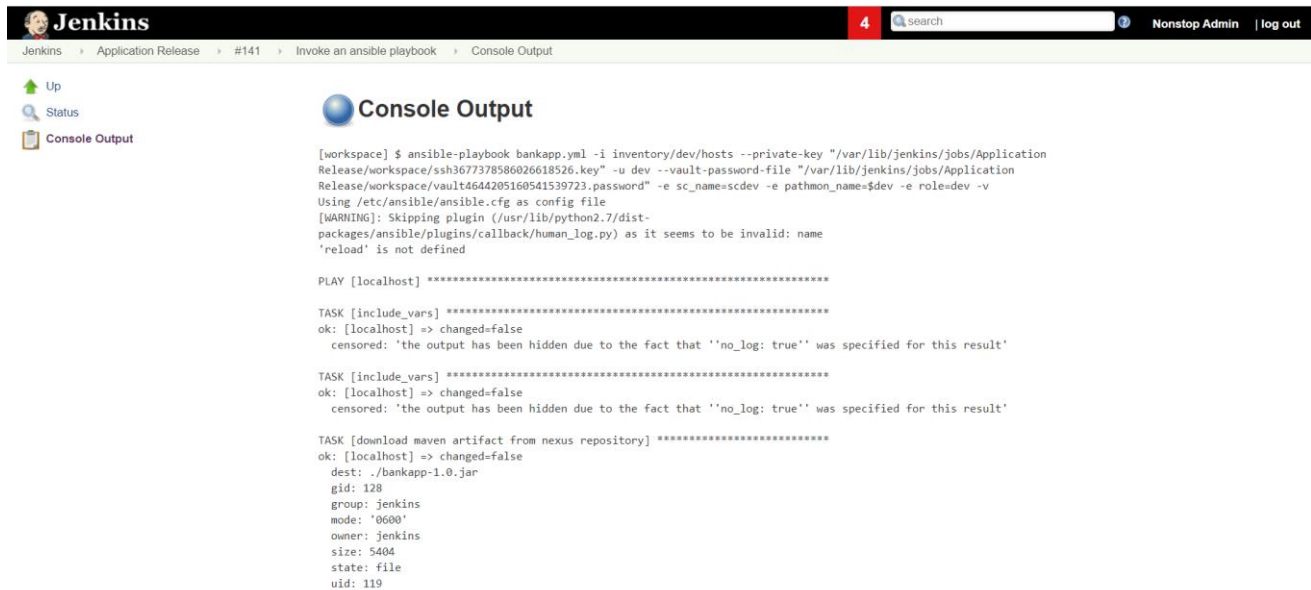
Once this pipeline job is triggered, it executes all the phases of the pipeline one after the other. Following are some of the logs of each phase.

#### Deliver and Deploy application in DEV environment

Snapshot of Jenkins pipeline at this stage.



Snapshot of the Console Output at this stage. Here you can see the parameters passed to the Ansible command are all specific to the developer such as role=dev.



At the end of this phase it will print the following message at the end of console output. This makes sure that the application was delivered and deployed at DEV environment successfully.

```
Jenkins > Application Release > #141 > Invoke an ansible playbook > Console Output
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdm : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdm : remove serverclass] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdm : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdm : status serverclass] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdm : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdm : fail] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

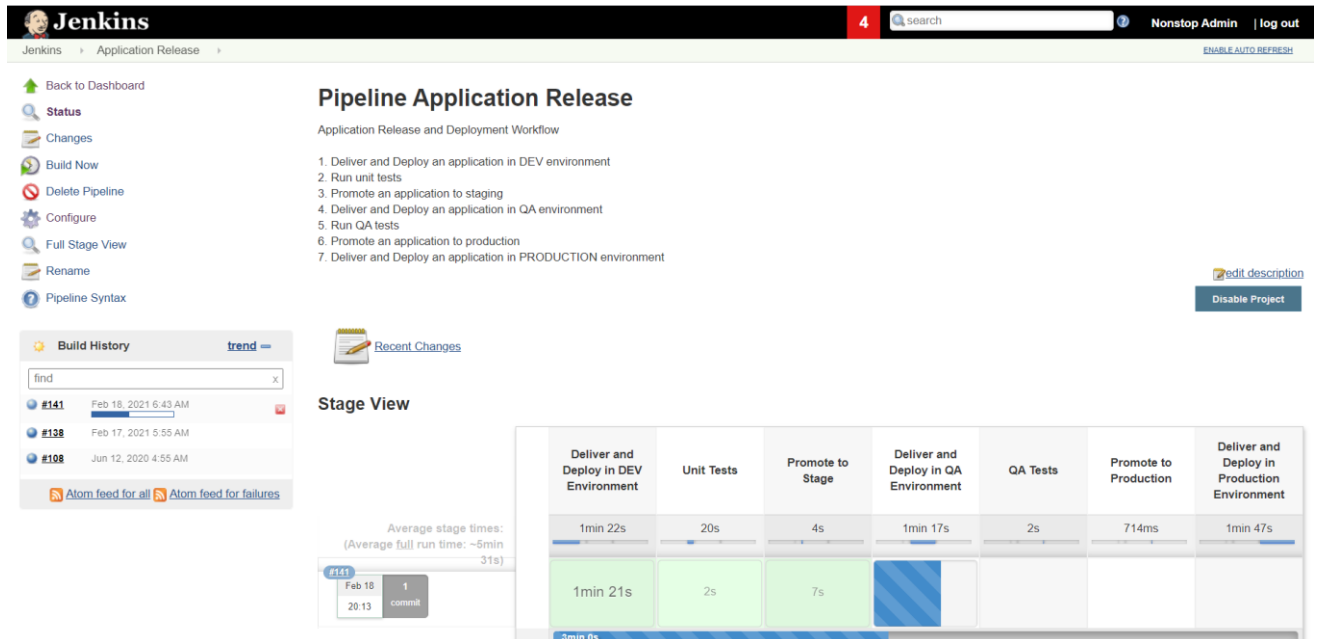
TASK [print the message] *****
ok: [cfot.in.rdlabs.hpecorp.net] => changed=false
msg: Application is successfully delivered and deployed at dev environment

PLAY RECAP *****
cfot.in.rdlabs.hpecorp.net : ok=15  changed=4  unreachable=0  failed=0  skipped=100  rescued=0  ignored=0
localhost                  : ok=3    changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0

Page generated: Feb 18, 2021 6:54:06 AM PST Jenkins ver. 2.190.3
```

## Deliver and Deploy application in QA environment

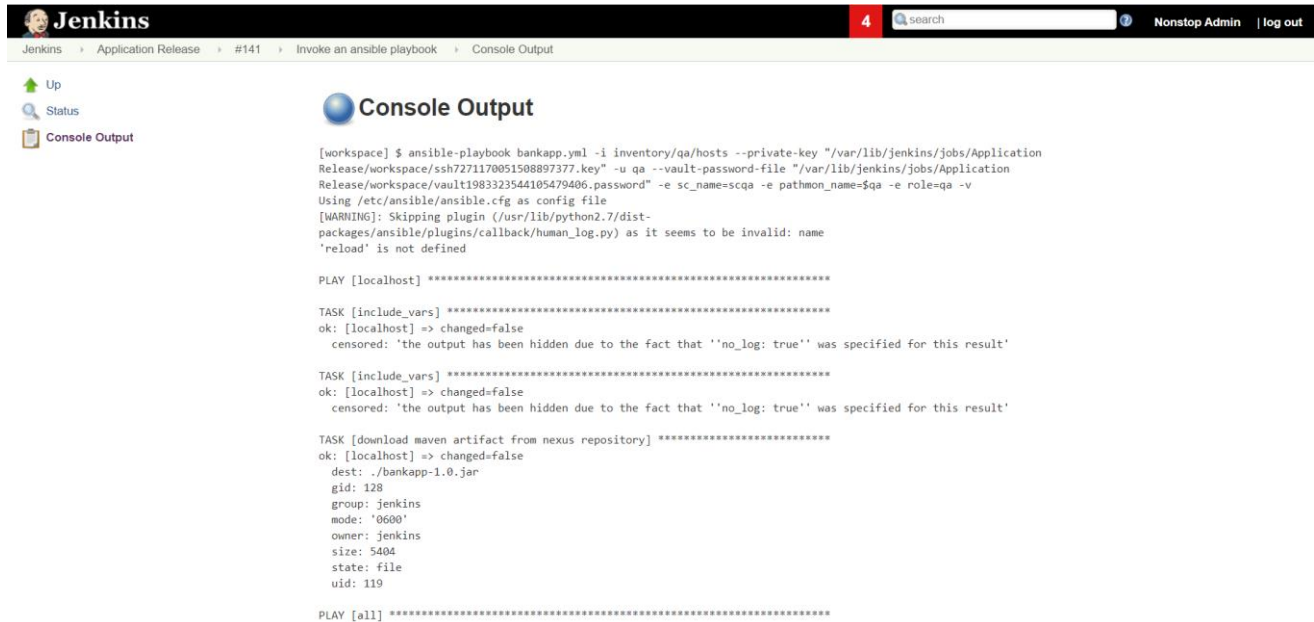
### Snapshot of Jenkins pipeline at this stage



The screenshot shows the Jenkins Pipeline Application Release interface. The pipeline is named "Application Release" and is currently in the "Build Now" state. The pipeline consists of 7 stages: 1. Deliver and Deploy an application in DEV environment, 2. Run unit tests, 3. Promote an application to staging, 4. Deliver and Deploy an application in QA environment, 5. Run QA tests, 6. Promote an application to production, and 7. Deliver and Deploy an application in PRODUCTION environment. The "Build History" section shows the latest build #141, which was completed on Feb 18, 2021 at 6:43 AM. The "Stage View" section shows the progress of the pipeline, with the "Deliver and Deploy in QA Environment" stage currently in progress. The "Average stage times" section shows the average run times for each stage: 1min 22s, 20s, 4s, 1min 17s, 2s, 714ms, and 1min 47s. The "Recent Changes" section shows the latest commit by user 1 on Feb 18, 2021.

Stage	Deliver and Deploy in DEV Environment	Unit Tests	Promote to Stage	Deliver and Deploy in QA Environment	QA Tests	Promote to Production	Deliver and Deploy in Production Environment
Average stage times	1min 22s	20s	4s	1min 17s	2s	714ms	1min 47s
Current Run	1min 21s	2s	7s	In Progress			
Total	5min 0s						

Snapshot of the Console Output at this stage. Here you can see the parameters passed to the Ansible command are all specific to the QA user such as role=qa.



```

Jenkins
Application Release
#141
Invoke an ansible playbook
Console Output

Up
Status
Console Output

[workspace] $ ansible-playbook bankapp.yml -i inventory/qa/hosts --private-key "/var/lib/jenkins/jobs/Application
Release/workspace/ssh7271170051508897377.key" -u qa --vault-password-file "/var/lib/jenkins/jobs/Application
Release/workspace/vault1983323544105479406.password" -e sc_name=scqa -e pathmon_name=$qa -e role=qa -v
Using /etc/ansible/ansible.cfg as config file
[WARNING]: Skipping plugin (/usr/lib/python2.7/dist-
packages/ansible/plugins/callback/human_log.py) as it seems to be invalid: name
'reload' is not defined

PLAY [localhost] *****

TASK [include_vars] *****
ok: [localhost] => changed=false
  censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

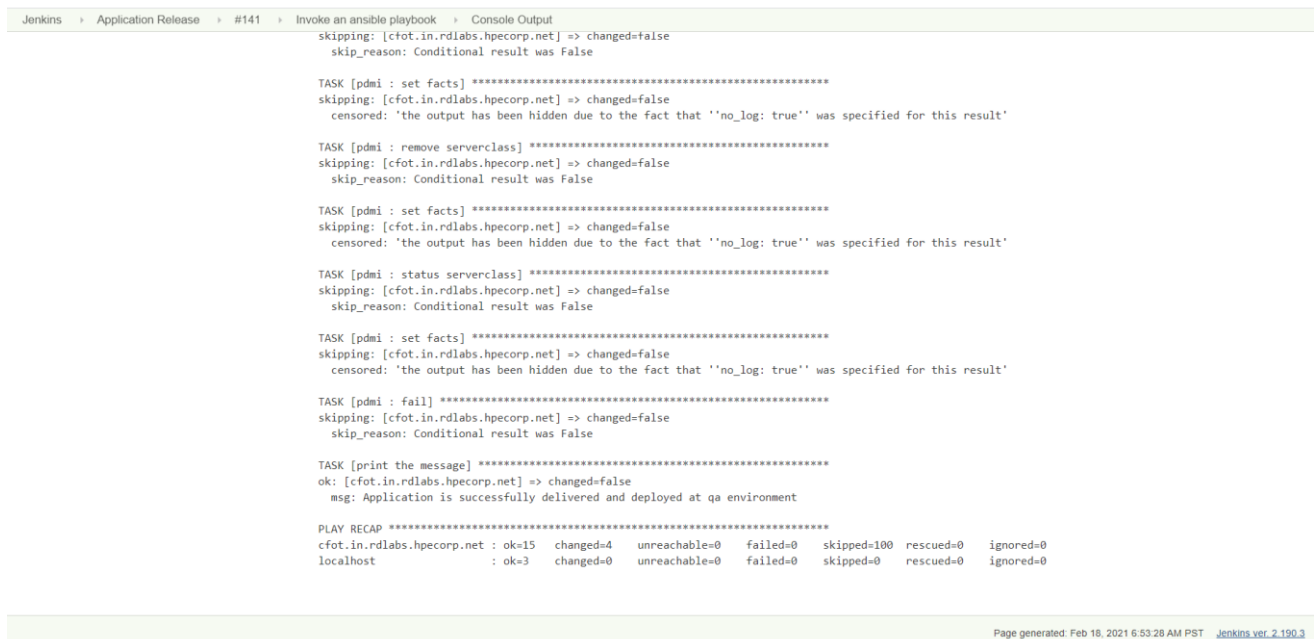
TASK [include_vars] *****
ok: [localhost] => changed=false
  censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [download maven artifact from nexus repository] *****
ok: [localhost] => changed=false
  dest: ./bankapp-1.0.jar
  gid: 128
  group: jenkins
  mode: '0600'
  owner: jenkins
  size: 5404
  state: file
  uid: 119

PLAY [all] *****

```

At the end of this phase it will print the following message at the end of console output. This makes sure that the application was delivered and deployed at QA environment successfully.



```

Jenkins
Application Release
#141
Invoke an ansible playbook
Console Output

skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdm : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
  censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdm : remove serverclass] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdm : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
  censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdm : status serverclass] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdm : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
  censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdm : fail] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

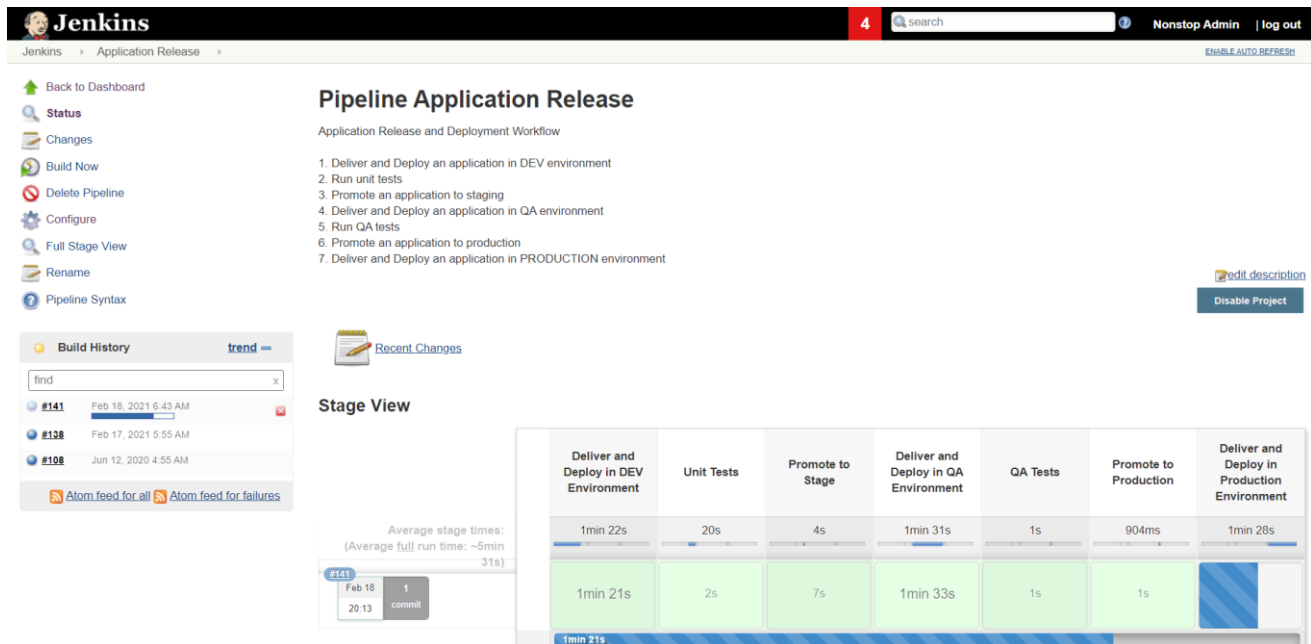
TASK [print the message] *****
ok: [cfot.in.rdlabs.hpecorp.net] => changed=false
  msg: Application is successfully delivered and deployed at qa environment

PLAY RECAP *****
cfot.in.rdlabs.hpecorp.net : ok=15  changed=4  unreachable=0  failed=0  skipped=100  rescued=0  ignored=0
localhost                  : ok=3    changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0

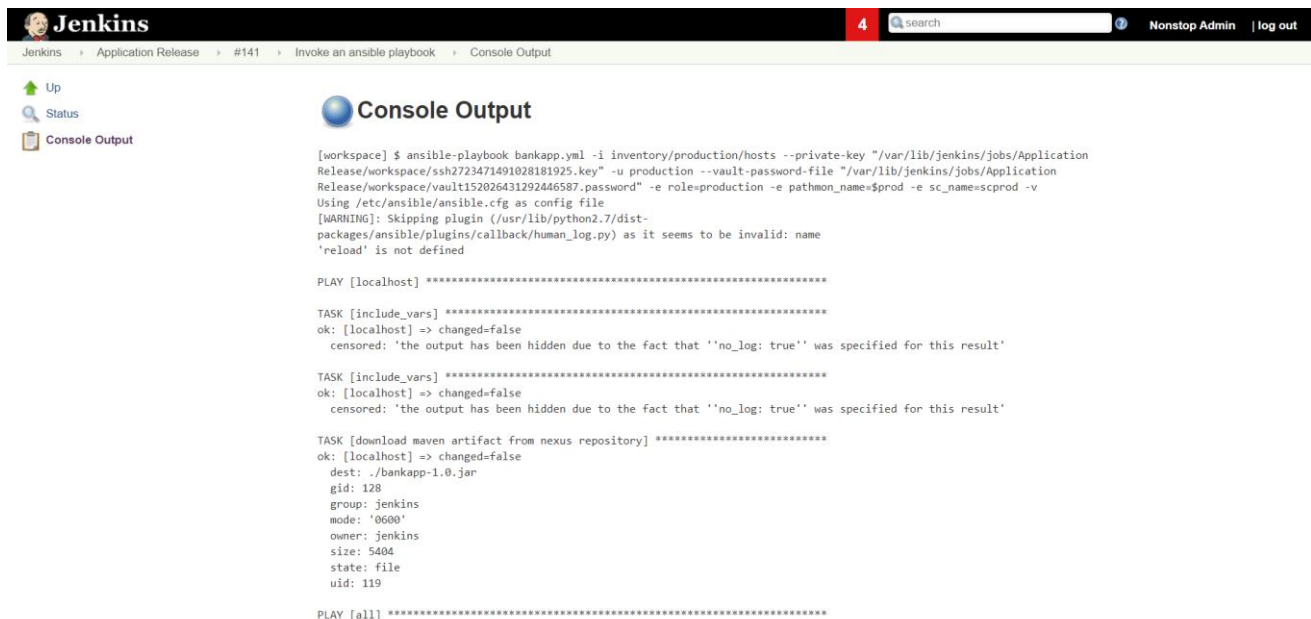
Page generated: Feb 18, 2021 6:53:28 AM PST  Jenkins ver. 2.190.3

```

**Deliver and Deploy application in Production environment**  
 Snapshot of Jenkins pipeline at this stage



Snapshot of the Console Output at this stage. Here you can see the parameters passed to the Ansible command are all specific to the production user such as role=production.



At the end of this phase it will print the following message at the end of console output. This makes sure that the application was delivered and deployed at Production environment successfully.



```
Jenkins > Application Release > #141 > Invoke an ansible playbook > Console Output
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdmi : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdmi : remove serverclass] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdmi : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdmi : status serverclass] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [pdmi : set facts] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
censored: 'the output has been hidden due to the fact that ''no_log: true'' was specified for this result'

TASK [pdmi : fail] *****
skipping: [cfot.in.rdlabs.hpecorp.net] => changed=false
skip_reason: Conditional result was False

TASK [print the message] *****
ok: [cfot.in.rdlabs.hpecorp.net] => changed=false
msg: Application is successfully delivered and deployed at production environment

PLAY RECAP *****
cfot.in.rdlabs.hpecorp.net : ok=15  changed=4  unreachable=0  failed=0  skipped=100  rescued=0  ignored=0
localhost                  : ok=3    changed=0  unreachable=0  failed=0  skipped=0    rescued=0  ignored=0

Page generated: Feb 16, 2021 6:48:52 AM PST  Jenkins ver. 2.190.3
```

## Python Starter Kit

The Python Starter Kit package is designed to demonstrate the Continuous Integration of a Purely Platform Agnostic Application profile of NonStop DevOps through a pythonquickstart application.

The setup, tooling, gating criteria and artifacts in this package can be used to automate the Continuous Integration Phase of any Python Project. This package acts as a starter kit for users.

This is a beginner level starter kit.

## Package Information

The Python Starter Kit package consists on a pythonquickstart maven project with a simple client and server python application and a README file.

This package does not contain the pre-requisite software for Continuous Integration Setup. The pre-requisite software have to be downloaded and installed as per the instructions in the above section.

## Pre-requisites software

The following are the pre-requisites on the system identified for the Python Starter Kit.

Software	Description	Version
Jenkins	Build Management System	2.222.3 or later
GITHub	Source Code Control	
GIT	Source Code Control	2.26.2-64-bit or later
Python	Python is an interpreted high-level general-purpose programming language.	Python 2.7.18 or later version
SonarQube	Static Analysis Tools	8.3.0.xx or later

Unit test or pytest	Unit Testing or Pytest Framework	Included in the python package (pip install pytest)
Nexus Repository Manager	Binary Artifact Repository	3.17.0.01 or later
Python Coverage	Coverage, used to create coverage.xml	version 5.5
Bandit	Source code security analysis tool for Python Script	version 1.5.1

### Using the Starter Kit

The Python Starter Kit requires SCM, Jenkins, SonarQube and Artifact Repository. Perform the setup using the instructions set provided in the last section.

### Uploading the project to SCM

Make sure that you can log on to <https://github.<yourcompanyname>.com/>

Create a new repository in GITHUB (NSDevOpsPythonQuickStart)

Copy the content of “python” into a folder of your choice

Using the GIT CLI upload the python quickstart to the github using the following commands

```
git init
git add *
git commit * -m "<Message>"
git remote add origin https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsPythonQuickStart>.git
git push -u origin master
```

where NSDevOpsPythonaQuickStart is the name of your GITHUB repository

devops-user is the username

<your-personal-token> is the SCM Personal Token

github. <yourcompanyname>.com is the GitHub repository

Replace the items mentioned within <> and marked in red font with values appropriate to your setup.

### Continuous Integration Pipelines using Python Starter Kit

#### Developer Pipeline

This workflow triggers when a developer checks-in the code. The pipeline performs scm checkout, build and unit testing using pytest or unit test which is available in the python package. Since many developers may be involved in development and each of their remote NonStop Node and environments vary, this workflow is given as a scripted pipeline job.

A template could be created for the project and individual developers can use the template and update their specific environment details.

### Jenkins Job Setup

- Create a Jenkins Pipeline Job PythonKStart
- In the General tab
  - Provide the Description
  - Select GitHub Project and provide the SCM Link  
`https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsPythonQuickStart>.git`

Update the items marked in red with values appropriate to your setup.

Dashboard > PythonKStart >

**General** Build Triggers Advanced Project Options Pipeline

NonStop DevOPS Python Starter Kit  
 Automated build triggered (Jenkins PollSCM build trigger)  
 Automated code check-out (by Jenkins from GITHUB)  
 Automated Build (Jenkins Pipeline Script and Build automation tools)  
 Static Analysis of Code  
 Unit Test or Pytest run on NonStop

[Plain text] [Preview](#)

Jira site

☐ Discard old builds

☐ Do not allow concurrent builds

☐ Do not allow the pipeline to resume if the master restarts

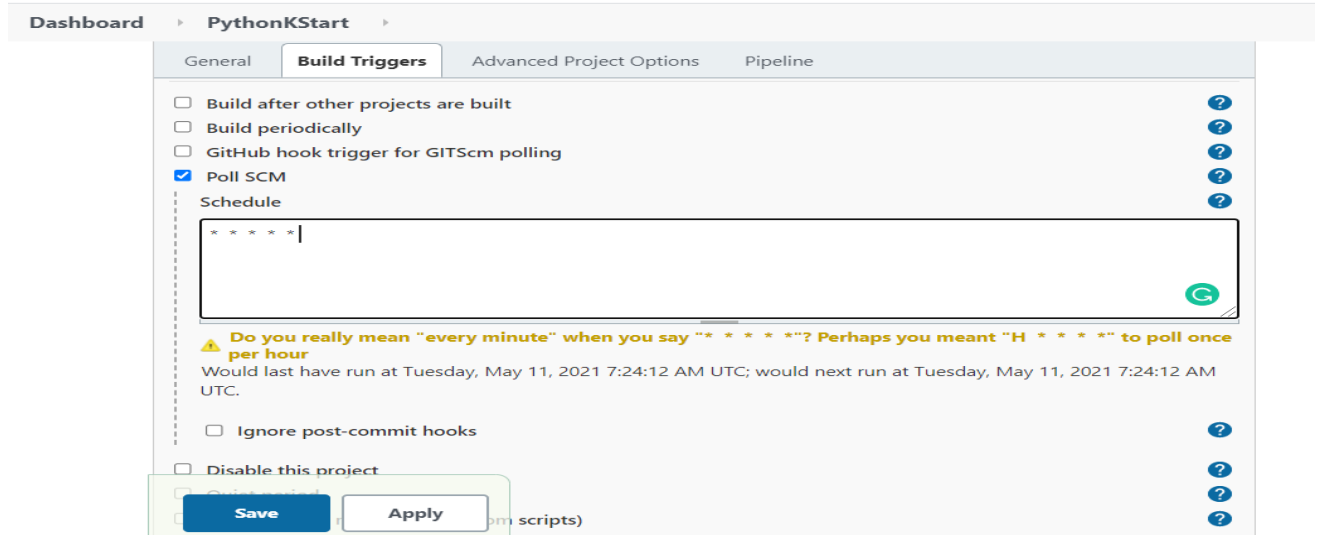
☒ GitHub project

Project url

`https://<devops-user>:<your-personal-token>@github.<yourcompanyname>.com/<devops-user>/<NSDevOpsPythonQuickStart>.git`

**Save** **Apply** **Advanced...**

- In the BuildTriggers Tab,
  - enable the PollSCM option and provide the schedule how often the SCM has to poll. Providing \* \* \* \* \* will poll every minute.
  - Note the spaces in the above pattern.



- In the Advanced Project Options,
  - Under Pipeline Definition, choose the Pipeline Script option.
  - Copy paste the script below after making changes to the repositories, node name appropriately.

```
properties(
[
[
$class: 'BuildDiscarderProperty',
strategy: [$class: 'LogRotator', numToKeepStr: '10']
],
]
)

node ('master') {
def remote = [:]
remote.name = '<nsdev>'
remote.host = '<XX.YYY.ZZ.AA>'
remote.user = '<devops.user>'
remote.password = '<XXXX>'
remote.allowAnyHosts = true

withEnv(["NEXUS_VERSION=nexus3", "NEXUS_PROTOCOL=http", "NEXUS_U
RL= XX.YYY.ZZ.AA:8081", "NEXUS_REPOSITORY=<python-nexus-
repo>", "NEXUS_CREDENTIAL_ID=<python-
user>", "NEXUS_CREDENTIAL_PASSWORD=<XXXX>"]){
stage('Code Checkout') { // for display purposes
// Get some code from a GitHub repository
```

```

        git 'https://<devops-user>:<your-personal-
token>@github.<yourcompanyname>.com/<devopsuser>/<NSDevOpsPythonQ
uickStart>.git/'
    }
    stage('Static Analysis') {
        // Run the pylint
        if (isUnix()) {
            sh "pylint --exit-zero --output-
format=colored ${env.WORKSPACE}/client/socketClient.py"
            sh "pylint --exit-zero --output-
format=colored ${env.WORKSPACE}/client/socketClient.py"
            // /usr/bin/bandit
            sh "/usr/bin/bandit -v -lll
${env.WORKSPACE}/server/socketServer.py"
            sh "/usr/bin/bandit -v -lll
${env.WORKSPACE}/client/socketClient.py"
        } else {
            bat("pylint --exit-zero --output-
format=colored ${env.WORKSPACE}\\client\\socketClient.py")
            bat("pylint --exit-zero --output-
format=colored ${env.WORKSPACE}\\client\\socketClient.py")
        }
    }
    stage("SonarQube Analysis") {
        // Run the maven build, not need for python
        application
        if (isUnix()) {
            sh "/opt/sonar-scanner/bin/sonar-scanner -
Dsonar.projectName=\"PythonStartKit\" -Dsonar.projectKey=<project
key> -
Dsonar.projectBaseDir=/var/jenkins_home/workspace/PythonKStart/ -
Dsonar.sources=server,client -
Dsonar.python.coverage.reportPATH=${env.WORKSPACE}/coverage.xml"
        } else {
            bat("C:\\Applications\\bin\\sonar-scanner -
Dsonar.projectName=\"PythonStartKit\" -Dsonar.projectKey=<project
key> -Dsonar.projectBaseDir=${env.WORKSPACE} -
Dsonar.sources=server,client -
Dsonar.python.coverage.reportPATH=${env.WORKSPACE}\\coverage.xml"
)
        }
    }
    stage('Binaries To NonStop') {
        sshCommand remote: remote, command: "mkdir -p
/tmp/PythonKStart/server"
    }

```

```

        sshCommand remote: remote, command: "mkdir -p
/tmp/PythonKStart/client"
        sshPut remote: remote, from:
'server/socketServer.py', into:
'/tmp/PythonKStart/server/socketServer.py'
        sshPut remote: remote, from:
'client/socketClient.py', into:
'/tmp/PythonKStart/client/socketClient.py'
        sshCommand remote: remote, command: "cd
/tmp/PythonKStart/;ls -lrt"
    }

    stage('Unit Test') {
        sshCommand remote: remote, command: "cd
/tmp/PythonKStart/;/usr/bin/python server/socketServer.py & "
        sshCommand remote: remote, command: "cd
/tmp/PythonKStart/client;/usr/bin/python socketClient.py"
        sshCommand remote: remote, command: "/bin/ps -eaf |
/bin/grep socketServer | /bin/grep -v grep | /bin/awk '{ print
\$2 }'| /bin/xargs /bin/kill -9"
    }
    stage('Regression Test') {
        //Add steps to copy regression test suite, setup
test env and run the regression tests
    }

    stage('Artifact Repository Upload'){
        if (isUnix()) {
            echo "Create python package"
            ...
            ...
            sh "tar -cvf /tmp/PythonKStart-
1.0.${BUILD_NUMBER}.tar /tmp/PythonKStart > tar.log 2>&1"
            sh "gzip /tmp/PythonKStart-
1.0.${BUILD_NUMBER}.tar"
            sh "twine upload --verbose /tmp/PythonKStart-
1.0.${BUILD_NUMBER}.tar.gz --repository-
url=http://XX.YYY.ZZ.AA:8081/repository/<python-nexus-repo>/ -u
<user name> -p <password>"
        } else {
            bat("7z a -ttar -so dwt.tar C:\\PythonKStart |
7z a -si PythonKStart-1.0.${BUILD_NUMBER}.tar.gz")
            bat("twine upload --verbose
c:\\TEMP\\PythonKStart-1.0.${BUILD_NUMBER}.tar.gz --repository-

```

```
url=http://XX.YYY.ZZ.AA:8081/repository/<python-nexus-repo>/ -u
<user name> -p <password>")
    }
}
} //node
```

Note : Update the items marked in red in the above syntax with values appropriate to your setup

- Apply the changes and Save the Job.

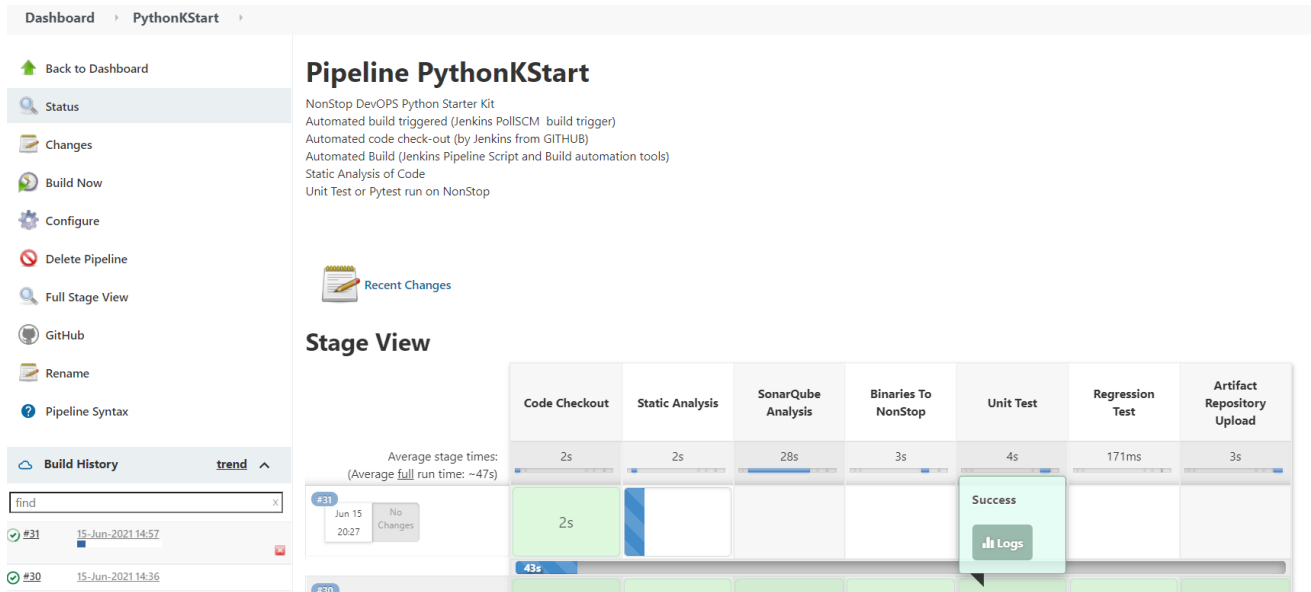
The screenshot shows the Jenkins configuration interface for a job named 'PythonKStart'. The 'Advanced Project Options' tab is active, and the 'Pipeline' section is expanded. The 'Definition' dropdown is set to 'Pipeline script'. The script content is as follows:

```
21 }
22
23 stage('Static Analysis') {
24     // Run the pylint
25     if (isUnix()) {
26         sh "pylint --exit-zero --output-format=colored ${env.WORKSPACE}/client/socketClient.py"
27         sh "pylint --exit-zero --output-format=colored ${env.WORKSPACE}/client/socketClient.py"
28     } else {
29         bat("pylint --exit-zero --output-format=colored ${env.WORKSPACE}\\client\\socketClient.py")
30         bat("pylint --exit-zero --output-format=colored ${env.WORKSPACE}\\client\\socketClient.py")
31     }
32 }
33
34 stage('SonarQube Analysis') {
35     // Run the maven build, not need for python application
36     if (isUnix()) {
37         sh "/opt/sonar-scanner/bin/sonar-scanner -Dsonar.projectName='PythonStartKit' -Dsonar.projectKey=753bf55e43"
38     }
```

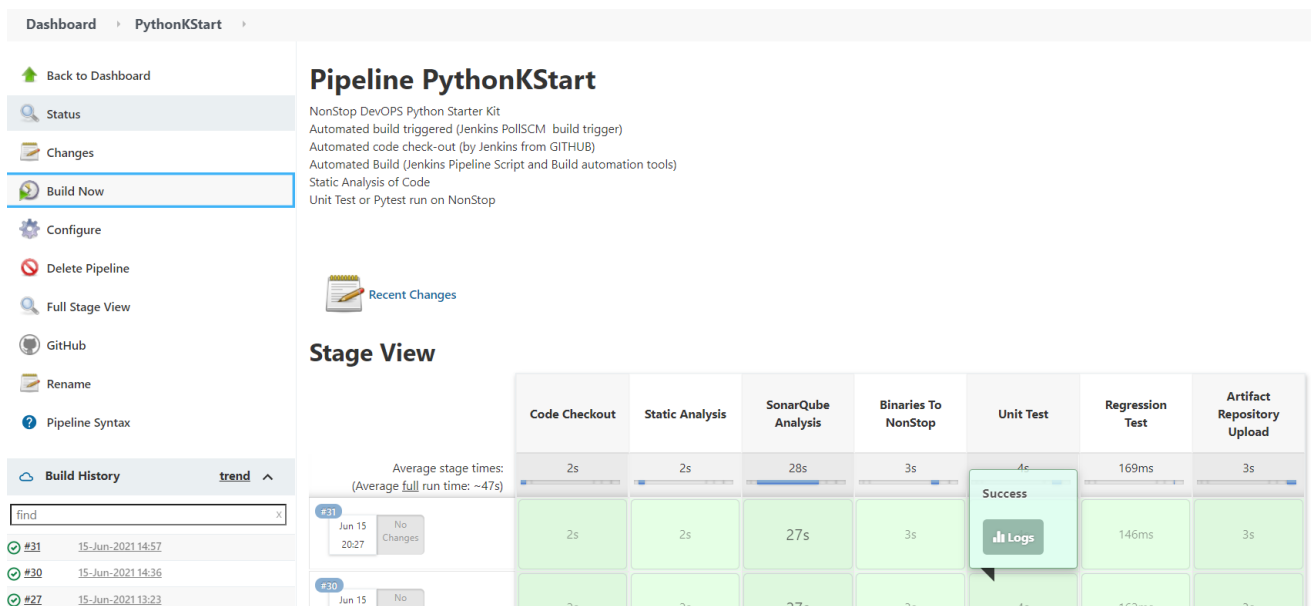
Below the script, the 'Use Groovy Sandbox' checkbox is checked. At the bottom of the configuration area, there are 'Save' and 'Apply' buttons.

Since the Poll SCM option is turned on, when the developer makes a change and commits the code, the build is triggered. The Developer can monitor the Job through the Jenkins Job Console.

Alternately, the job can be triggered manually by clicking the Build Now option in the Jenkins PythonKStart



Each stage can be monitored by looking at the stage and logs per stage can be viewed.



Sample Console Output for the build





Note – In this sample, the master branch has been used. However, in a typical application development scenario, the developers might work on specific branches. These can be integrated with the main/master branch if the build succeeds. This step can be automated in the developer pipeline. So that Nightly Build can be triggered.

At the end of this phase it will print the following message at the end of console output. This makes sure that the application was delivered and deployed at Production environment successfully.



Dashboard > PythonKStart > #31

- Back to Project
- Status
- Changes
- Console Output
- View as plain text
- Edit Build Information
- Delete build '#31'
- Git Build Data
- Replay

### Console Output

```

Started by user Root
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/PythonKStart
[Pipeline] {
[Pipeline] withEnv
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Code Checkout)
[Pipeline] sh
+ rm -rf client server README.md
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository

```

## Trouble Shooting

**PROBLEM** – SampleDeveloperPipeline of Java Starter Kit fails in “code checkout” stage .

**SYMPTOMS** - Failed with the following error(s)

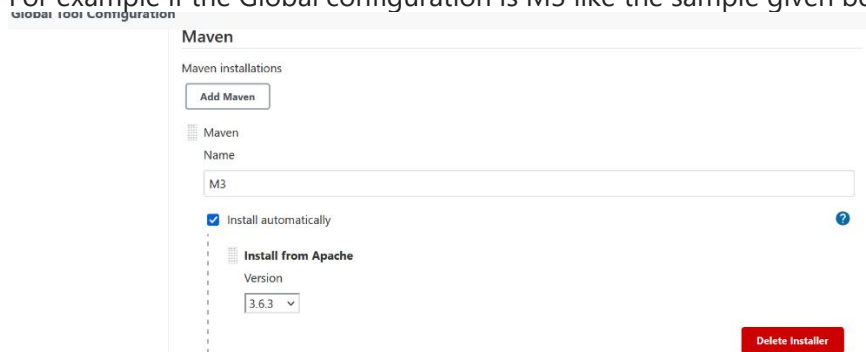
Use a tool from a predefined Tool Installation No tool named Maven found

**CAUSE** - The script has mvnHome = tool 'Maven' This needs to be changed according to your setup.

**SOLUTION** – Go to Jenkins DashBoard -> Manage Jenkins -> Global Tool Configuration -> Maven -> Maven Installations

Copy the name from the maven installation and replace the appropriate name in the script.

For example if the Global configuration is M3 like the sample given below



Global Tool Configuration

### Maven

Maven installations

[Add Maven](#)

Name	Version	Install automatically	Install from Apache
M3	3.6.3	<input checked="" type="checkbox"/>	<input type="checkbox"/>

[Delete Installer](#)

Change mvnHome = tool 'M3'

**PROBLEM** – SampleDeveloperPipeline of Java Starter Kit fails in “Automated Build or Static Analysis ” stage .

**SYMPTOMS** - Failed with the following error(s) repo.maven.apache.org:443

[repo.maven.apache.org/199.232.8.215] failed: Connection timed out (Connection timed out)

**CAUSE** - The Jenkins Master is behind a firewall and hence it is unable to access the maven repository

**SOLUTION** – Add -DproxySet=true -DproxyHost= <proxyhost> -DproxyPort= <proxyport> to GIT command line.

**PROBLEM** –The pipeline task does not start

**SYMPTOMS** - "Still waiting to schedule task 'Jenkins' doesn't have label 'MASTER'"

**CAUSE** - The Node names are case sensitive

**SOLUTION** – Ensure the node name in the script matches the node name configured in Jenkins