NetApp Jenkins Plugin Documentation

Release 2.0

Akshay Patil

Aug 22, 2017

Contents

The NetApp-Jenkins integration is an end-to-end framework from creating code repository until zipping the successful builds in the artifactory location. All these processes run as Docker-containers and use persistent NetApp storage with NetApp Docker volume plugin (nDVP).

The advantage of running CI process on NetApp for Business or Asset owner are –

1. Improve Developer productivity
   (a) NetApp FlexClones allows to cut down the user workspace provisioning from hours and days to few seconds.
   (b) Each user workspace is pre-packaged with the source code and the dependencies like libraries, tools, compilers and config files for developers start working quickly. There are no long wait times for developers to prepare their workspaces.
2. Reduce build time
   (a) NetApp Snapshots for the CI data volumes allows developers to run incremental builds over full builds. Full builds are time consuming.
   (b) Incremental builds allow developers to test the changes quickly in their workspaces and provides consistency to the builds with reduced build times.
3. Reduce infrastructure costs
   (a) NetApp data volumes also known as FlexVols, Snapshots and FlexClones are thin provisioned. This allows to provision more build workloads with less storage footprint and optimize compute and network resources for parallel builds. Thus providing an improved Return of Investment (ROI) by doing "more with less" in the entire build farm.
   (b) Data Compaction and De-duplication for the source code repositories and the software builds during the CI process on NetApp provides a high degree of storage space efficiency. Data compression of build artifacts in the binary repository lso provides space savings.

CHAPTER 1

Contents

Pre-Requisites

- 1 running instance of NetApp Service Level Manager(NSLM).
- Docker Engine 1.12.5. installation on atleast 2 Linux Nodes
- NetAppDVP 1.13. installed and configured on all Linux Nodes with Docker-Engine
- 1 running instance of jfroginstall

Note:

- All validation in this solution has been done with RHEL 7.3, the source files can run on any flavour of Linux.
- Any number of nodes can be added in the swarm cluster, for demo purposes 2 node cluster considered in this validation.
- **Keep the following storage details handy:**
    1. Management LIF IP: _____
    2. Data LIF IP: _____
    3. Storage Virtual Machine(SVM) name: _____
    4. SVM Username: _____
    5. SVM Password: _____
    6. Aggregate Name: _____
- **The tools in this framework use following ports, make sure following are open within your firewall:**

Protocol	Port	Used By
TCP	2377	Docker Swarm
TCP and UDP	7946	Docker Swarm
TCP and UDP	4789	Docker Swarm
TCP	80	GitLab
TCP	1024	Jenkins
TCP	50000	Jenkins Slaves

Configuration

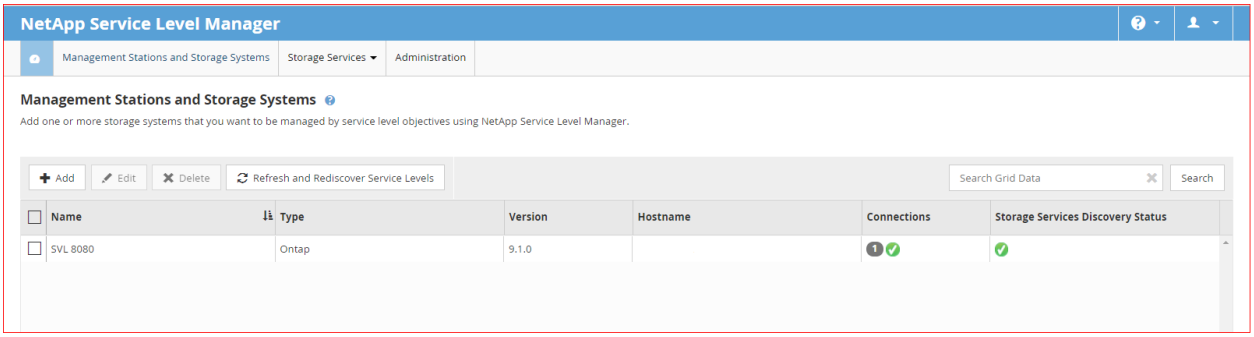
1. **Adding ONTAP Storage system in NetApp Service Level Manager**

   1.1) Open a web browser and enter the URL "https://xx.xx.xx.xx:8443/admin/" where xx.xx.xx.xx is the IP address of the host machine where NetApp SLM is installed.

   1.2) Enter your NSLM username and password when prompted.

   1.3) Click on +Add and enter the details of your ONTAP Cluster.

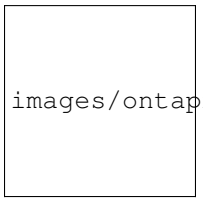
   1.4) Verify that the instance of ONTAP is added as shown in the screenshot.



2. **Building a docker image for your environment**

   2.1) Get the source code from NetApp Jenkins Framework Github Repo.

   2.2) Edit the /Jenkins_Master/ontap-nas.json file with appropiate values.



   2.3) After the ontap-nas.json file is configured, build the Jenkins Master docker image using following command:

```
>>docker build -t image_name:tag
```

   2.4) Once the build is complete, push the Docker image to a registry using following command:

```
>>docker push registry/imagename:tag
```

3. **Setting up a Docker Swarm Cluster** 3.1)Login into a Linux Node with Docker-Engine installed.

   3.2)Initialize Swarm cluster using following command:

```
>>docker swarm init
```

```
[root@linux-node13 ~]# docker swarm init
Swarm initialized: current node (2ont2f9xhhnstb0g1yr88aftb) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-4weve7a175zh8vm1jqkk6q6j1w7xq4tsh0v7yf1lapyiltsaxr-4fyyanfzupjmw75l9gizwdrhe \
    10.192.39.33:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Note:

- If there are multiple ethernet ports(eth0...ethn) configured on the host then "–advertise-addr" <IP-Address of Swarm Host> argument needs to be provided with swarm initialization.

```
[root@linux-node12 ~]# docker swarm init --advertise-addr 10.192.39.32
Swarm initialized: current node (br8o5qozu25w3i8v5r084n5gk) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-2csuepy6eg3w5rfbkh5a5bcpe6mhxdwdwppq8dq4dnf3b01uei-8u2u077fkix495le61lnwq215 \
    10.192.39.32:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

- This node will act as a Swarm manager and all swarm commands run only on the manager node.

3.3) Copy the swarm token generated by the aboce command and run it on a new Linux node. This new Linux node will join the swarm cluster as a swarm worker node.

```
[root@linux-node12 ~]#  docker swarm join \
>     --token SWMTKN-1-4weve7a175zh8vm1jqkk6q6j1w7xq4tsh0v7yf1lapyiltsaxr-4fyyanfzupjmw75l9gizwdrhe \
>     10.192.39.33:2377
This node joined a swarm as a worker.
[root@linux-node12 ~]#
```

Note: Any number of hosts can be added in your swarm cluster, this validation demonstrates use of a 2 node swarm cluster.

3.4)Verify the status of our Swarm cluster by running the command:

```
>>docker node ls
```

Both the nodes should be visible in the node list

4. **Running NetAppDVP on the Docker-Engine Hosts** 4.1) Make sure NetAppDVP is running on all the Linux nodes added in swarm cluster. A netappdvp process should be seen on the host

```
[root@linux-node13 ~]# docker node ls
ID                           HOSTNAME                              STATUS  AVAILABILITY  MANAGER STATUS
2ont2f9xhhnstb0g1yr88aftb *  linux-node13.openenglab.netapp.com  Ready   Active        Leader
d0u0yn9vfvl0qjxc9mykeg4cd    linux-node12                        Ready   Active
[root@linux-node13 ~]#
```

```
>>ps ax | grep netappdvp
```

```
[root@linux-node13 ~]# ps ax | grep netappdvp
 6551 ?        Ssl    0:00 /usr/local/bin/netappdvp --config=/etc/netappdvp/ontap-nas.json &
12450 pts/0    Sl     0:00 netappdvp --config=/etc/netappdvp/ontap-nas.json
13055 pts/0    S+     0:00 grep --color=auto netappdvp
[root@linux-node13 ~]#
```

5. **Start the NetApp-Jenkins Docker Service** 5.1) Start the NetApp-Jenkins-Master Docker Service using following command. Use the docker image built in step 3

```
>>docker service create --replicas 1
--mount "type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock"
--constraint 'node.role==manager' --restart-condition on-failure
--mount "type=volume,source=Jenkins_home,volume-driver=netapp,target=/var/
↪jenkins_home"
--publish 50000:50000
--publish 1024:8080
--name jenkins registry/imagename:tag
```

```
[root@linux-node13 ~]# docker service create --replicas 1 \
> --mount "type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock" \
> --constraint 'node.role==manager' \
> --restart-condition on-failure \
> --mount "type=volume,source=Jenkins_home,volume-driver=netapp,target=/var/jenkins_home"  \
> --publish 50000:50000 --publish 1024:8080 \
> --name jenkins \
> devopsnetapponaws/netapp-jenkins-plugin-2.0:oss
```

5.2)This command will create a docker service of a Jenkins instance with the Jenkins_Home directory mounted on a NetApp volume.To check if the Service is created successfully use the following command

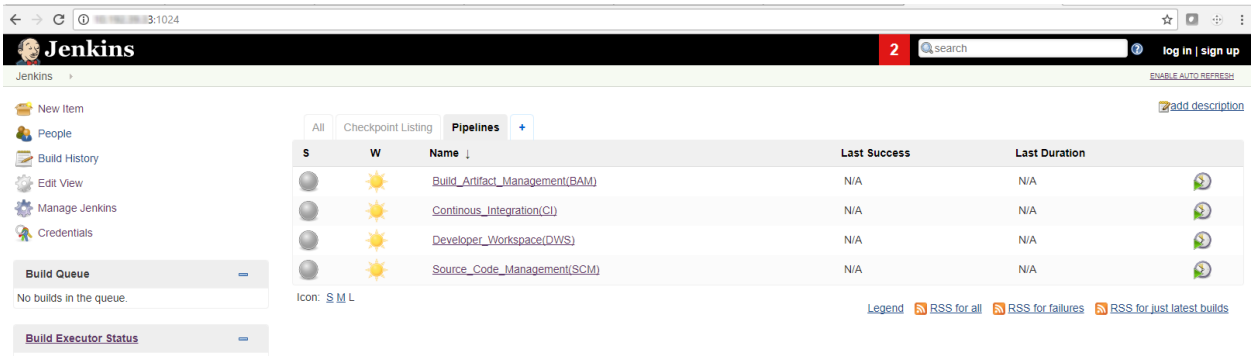
```
>>docker service ps jenkins
```

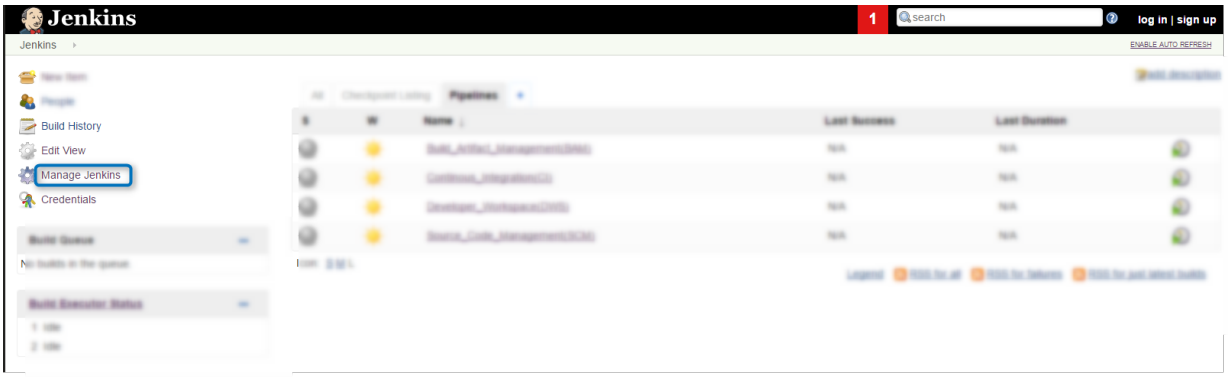
6. **Access Jenkins UI from a Web browser** 6.1) Once the Jenkins Docker Service is up, Navigate to the URL: **http://xx.xx.xx.xx:1024/** ,where xx.xx.xx.xx is Jenkins URL

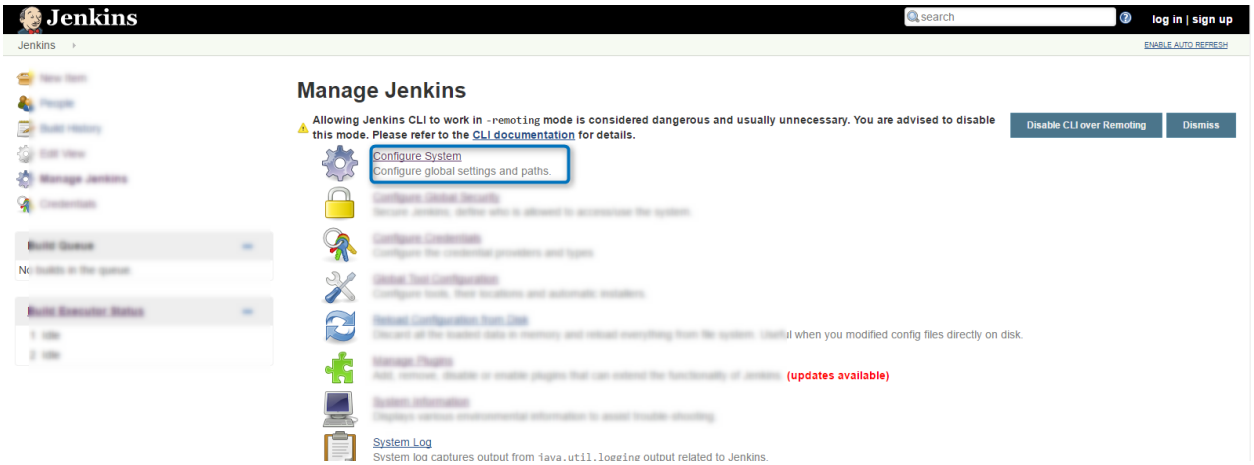
7. **Setting up environment variables and JFrog Artifactory in Jenkins** 7.1) In Jenkins UI navigate to Manage Jenkins>Configure System (http://xx.xx.xx.xx:1024/configure), where xx.xx.xx.xx is Jenkins URL

   7.2) Configure the following environment variables:

```
[root@linux-node13 ~]# docker service ps jenkins
ID                  NAME       IMAGE                                                NODE                                DESIRED STATE
 CURRENT STATE          ERROR
9wmltpmh2xp5n5u33y3bgbtzi  jenkins.1  devopsnetapponaws/netapp-jenkins-plugin-2.0:services  linux-node13.openenglab.netapp.com  Running
 Running 5 minutes ago
[root@linux-node13 ~]#
```







Variable Name	Default Value	What it does?
APISERVER	xx.xx.xx.xx:8443	URL of you NetApp Service Level Manager Installation
APIUSER	admin	Username of NSLM installation
APIPASS	Password@123	Password of NSLM installation
SLAVE	devopsnetapponaws/netapp-jenkins_slave:autodiscover	Jenkins Slave Docker image name
GITLABIM-AGE	devopsnetapponaws/netapp-jenkins_gitlab	SCM Docker Image name
VOLSIZE	4096	Size of volumes created in MB's
VS	lab2	Storage Tenant to create volumes
REPOUSER-NAME	admin	Username for private docker registry
REPOPASS-WORD	password	Password for private docker registry
ART_URL	xx.xx.xx.xx:5001	URL of your private registry(IP:PORT)
ART_REPO	docker-dev	Repository name to push docker images and zip files

Sample configuration:

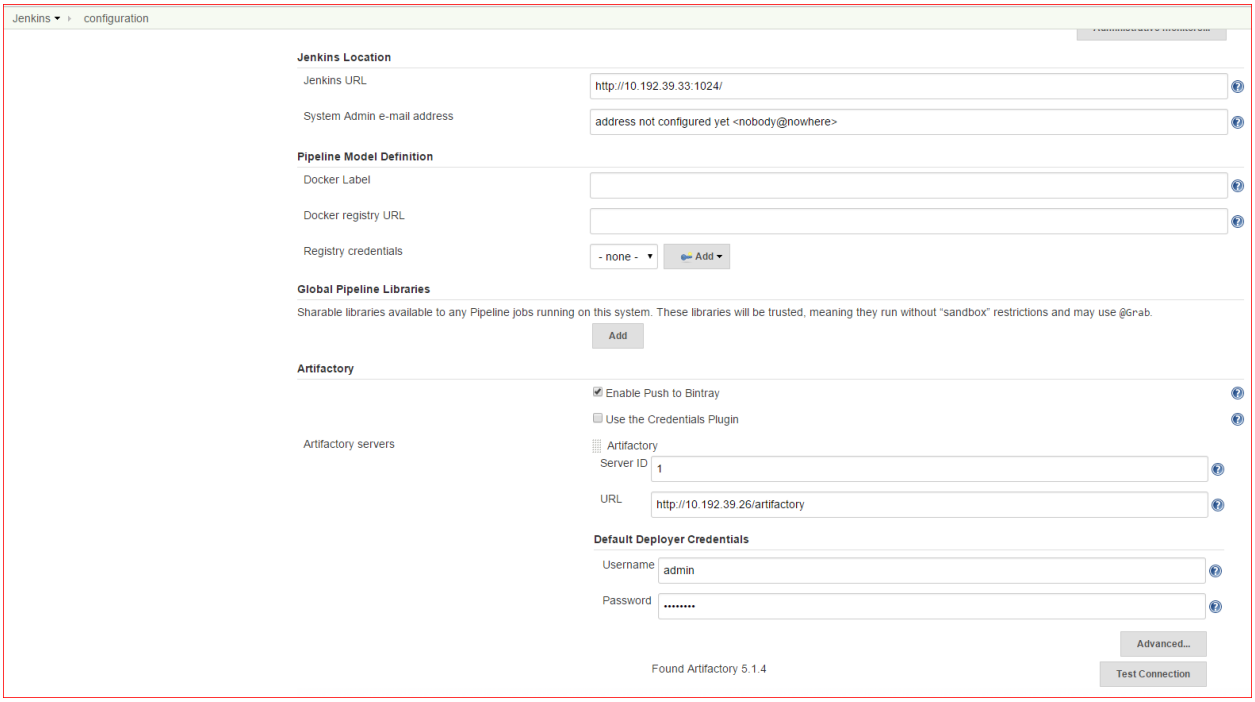
8. **Setting Up Artifactory** 8.1) In Jenkins UI navigate to Manage Jenkins>Configure System (http://xx.xx.xx.xx:1024/configure) , where xx.xx.xx.xx is Jenkins URL

   8.2) Scroll down until the Artifactory section

8.3) Enter your SERVER ID (eg: 1)

8.4) Enter your Artifactory Server Link

8.5) Click Save



Note: Check if the Jenkins URL in Jenkins Location tab maps to the URL of the Linux node running the Jenkins Master Docker Service.

---

9. **Configure Maven Home in Jenkins** 9.1) In the Jenkins Slave Image we already have installed maven at the default location i.e /usr/share/maven

---

Note: As part of this validation, a Maven sample project is used, If the production environment has any other type of build, that needs to be configured here.

9.2) Navigate to Manage Jenkins > Global Tool Configuration (http://xx.xx.xx.xx:1024/configureTools/) , where xx.xx.xx.xx is Jenkins URL

9.3) Scroll down till the Maven Section

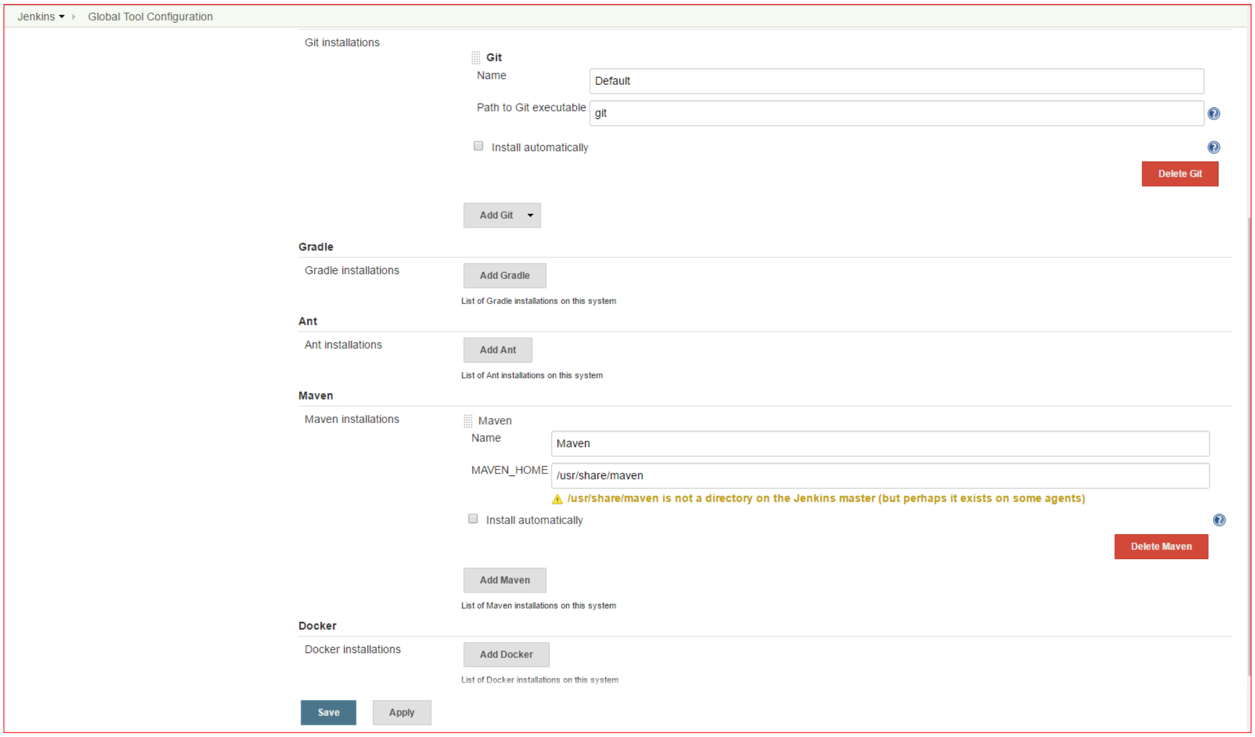
9.4) Click Add Maven

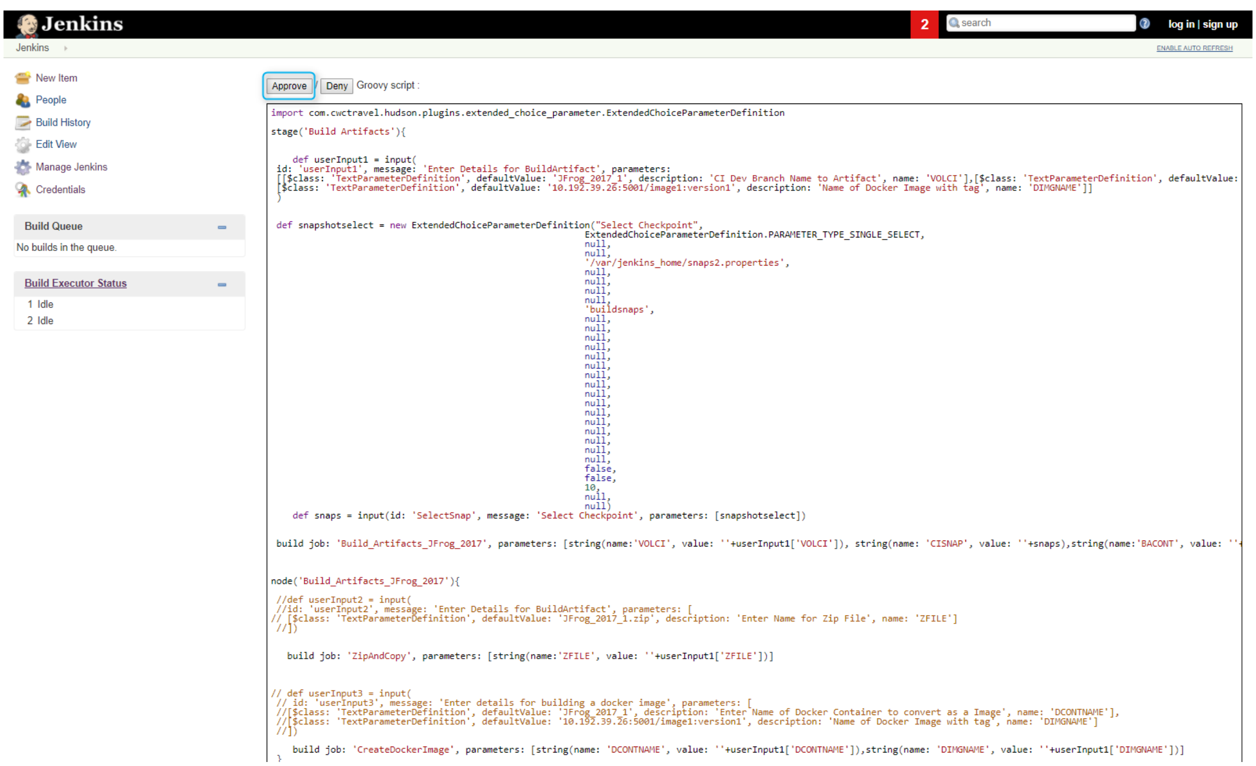
9.5) Enter Maven Installation name (eg: Maven)

9.6) Enter Maven Path as /usr/share/maven , as shown in below screenshot

10. **Approving the NetApp Groovy Pipelines** 10.1) In Jenkins UI navigate to Manage Jenkins>In-Process Script Approval (http://xx.xx.xx.xx:1024/configure) , where xx.xx.xx.xx is Jenkins URL

    10.2) As per the latest Jenkins security update, any external script in Jenkins needs to be approved. Click Approve for the 4 NetApp Groovy Scripts


Jenkins
Global Tool Configuration
Git installations
Git
Name
Default
Path to Git executable
git
Install automatically
Delete Git
Add Git
Gradle
Gradle installations
Add Gradle
List of Gradle installations on this system
Ant
Ant installations
Add Ant
List of Ant installations on this system
Maven
Maven installations
Maven
Name
Maven
MAVEN_HOME
/usr/share/maven
/usr/share/maven is not a directory on the Jenkins master (but perhaps it exists on some agents)
Install automatically
Delete Maven
Add Maven
List of Maven installations on this system
Docker
Docker installations
Add Docker
List of Docker installations on this system
Save
Apply


Jenkins
2
search
log in | sign up
ENABLE AUTO REFRESH
New Item
People
Build History
Edit View
Manage Jenkins
Credentials
Build Queue
No builds in the queue.
Build Executor Status
1 Idle
2 Idle
Approve
Deny
Groovy script :
import com.cwctravel.hudson.plugins.extended_choice_parameter.ExtendedChoiceParameterDefinition
stage('Build Artifacts'){

Workflow

The CI workflow is defined as following stages in this solution:

1. Source Code Management
2. **Continuous Integration**
    - Continous Integretion Environment Setup
    - Continuous Integration Build Setup
3. Developer Workspace Creation
4. Build Artifact Management

Predefined Pipelines and Jobs

As an example, following nomenclature for the Jenkins job names is used :

Job Task	Default Name used:
SCM Setup	JFrog_OSS_Repo
CI Environment Setup	JFrog_2017_1
CI Build	JFrog_CI_Build
Developer Workspace Name	Dev1_JFrog_2017_1
Build Artifact Container Name	Build_Artifacts_JFrog_2017

For purpose of explaining pipelines in this documentation, sample opensource scripts from JFrog are used to demonstrate a CI workflow.

```
https://github.com/JFrogDev/project-examples
```

This framework has following prefdefined pipelines and preconfigured jobs:

1. Predefined Pipelines

Pipelines	Tasks	Jobs Included in the Pipeline
Source Code Management	1)Spin up a GitLab Docker Container	JFrog_OSS_Repo
Continous Integration	1)Get the Local Git Repo URL 2)Spin up Container where CI Builds will Run 3)Pull the code from Gitlab to this container 4)Start the CI Build 5)Automatic Snapshot Creation for Every successful build.	JFrog_2017_1 Create_Build_Checkpoints List_Build_Checkpoints
Developer Workspace	1)Create prepackaged workspaces (containers) from snapshots	UserWorkspaces SCM_Checkpoints List_SCM_Checkpoints
Build_Artifact_Management	1)Spin up container to archive builds 2)Zip a Build Environment and push it to artifactory 3)Create a Docker image of Build environment and push it to a repo	Build_Artifacts_JFrog_2017 Zip_And_Copy Create_Docker_Image

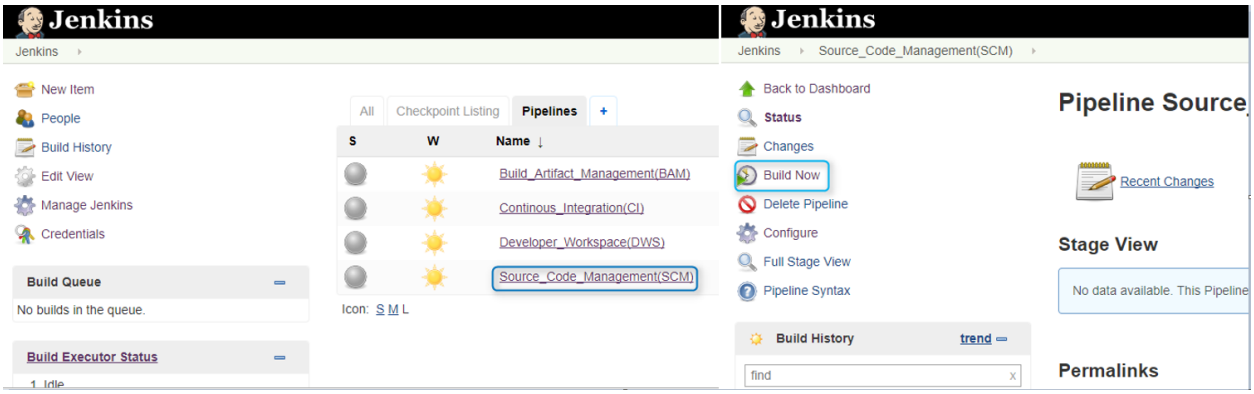
2. Preconfigured Jobs

Job Name	Tasks	Scripts Included
JFrog_OSS_Repo	1)Create 3 NetApp Volumes 2)Spin up a GitLab Container with its Logs,Data and configuration stored on NetApp Volumes	scmconfig2.py
JFrog_2017_1	1)Create a NetApp Volume 2)Create a Docker Service 3)Mount a NetApp Volume inside the Docker Service	CI_dev_bracbh_create2.py Jenkins_slave_create2.py
UserWorkspace	1)Create a FlexClone from a Build Snapshot 2)Create a Docker Service 3)Mount NetApp Clone inside the Docker Service	userworkspace_creation1.py Jenkins_slave_create2.py
Build_Artifacts_JFrog2017	1)Create a NetApp volume to store zip archives 2)Zip a Build Environment and push it to artifactory	Volume_create.py Build_Artifact_create.py
ZipandCopy	This jobs should always run after the Artifacts volume is created and should always run on a Jenkins Slave 1)Zip the contents of a clone volume 2)Move this zip to Artifact Volume 3)Push the zip to Artifactory 4)Delete the clone	build_artifact_exec.py clone_purge.py
CreateDockerImage	This job should always run on Jenkins Slave 1)Get container id from Docker Service Name 2)Commit the container 3)Build Docker Image of the container 4)Push the image to a private repo	dockerimagecreate.py
CreateBuildCheckpoints	1)Create a NetApp Snapshot 2)Tag build name and number to snapshot name 3)Write snapshot name to properties file so that extensible choice parameter plugin can read and display it in a dropdown menu	snapshot_create_write.py
SCMCheckpoints	This job is supposed to run by a Git WebHook for successful push 1)Tag a SHA number to name of a Snapshot 2)Create a netapp snapshot	scmcheckpoint_create.py
List_Build_Checkpoints	1)Display Snapshots for CI Build Volume	snap_show.py
List_SCM_Checkpoints	1)Display Snapshots for SCM Volume	snap_show.py
Purge Policy	This job is supposed to run on a cron schedule 1. Find Free and Busy Snapshots 2. Delete Free snapshots above a predefined number 3. If Number of busy snapshots exceed a certain pre-	purge.py

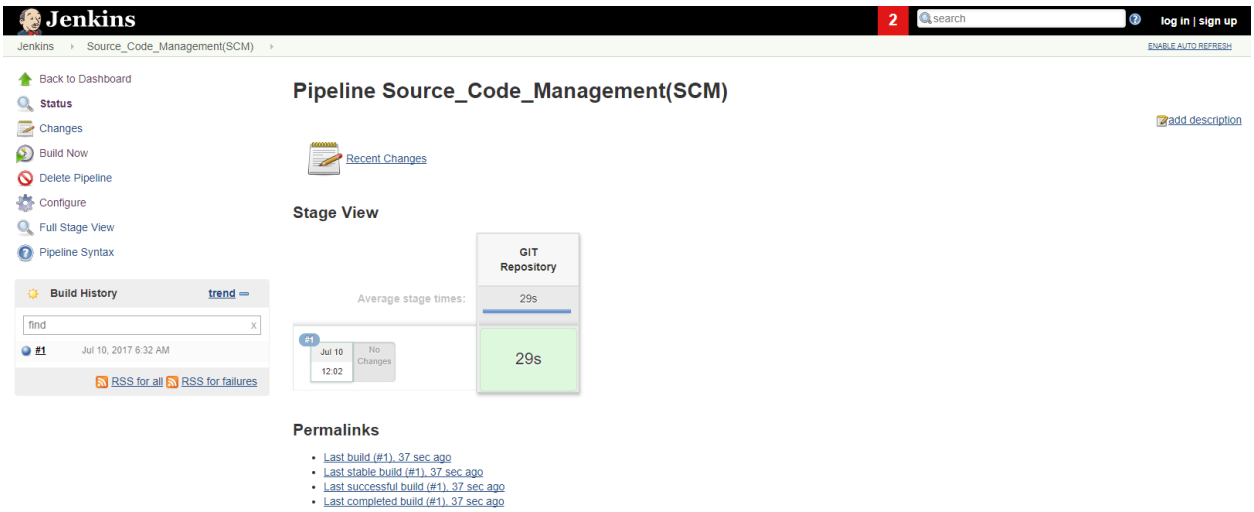
1. Source Code Management Setup

This pipeline will setup a GitLab container which acts as a SCM tool in this solution.

1.1) Click on the *Source_Code_Management* > *Build Now*



1.2) A Jenkins Stage named GIT Repository is started when the Source_Code_Management pipeline is built.



1.3) The GitLab Container can be seen on the Linux Host by running a “docker ps” command.:

```
>> docker ps
```

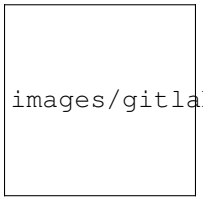
```
[root@linux-node13 ~]# docker ps
CONTAINER ID        IMAGE                                                COMMAND                  CREATED             STATUS
          PORTS                                                                                  NAMES
3f4928d55a1c        devopsnetapponaws/netapp-jenkins_gitlab              "/usr/bin/startcont.s"   About a minute ago   Up About a minute (health:
starting)   0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 0.0.0.0:10022->22/tcp   JFrog_OSS_Repo
40c862ef47c5        devopsnetapponaws/netapp-jenkins-plugin-2.0:oss      "/bin/tini -- /usr/lo"   About an hour ago    Up About an hour
          8080/tcp, 50000/tcp                                                                    jenkins.1.d4rcpiwgqhyuvk0wkf3fdeekm
[root@linux-node13 ~]#
```

Note: It takes about 2-5 minutes for GitLab to start.

1.4) Configuring Gitlab Navigate to http://<<RHEL-VM-IP>>/ in your browser and and set a root password for GitLab

1.5) After you set a password, Login into Gitlab using the recently set password on the same page.

1.6) Once logged into GitLab, Create a New Project



1.7) Import code repo into the local GitLab instance. For this validation Sample Hello world codes from JFrog's GitHub Repo are used:

```
https://github.com/JFrogDev/project-examples.git
```

1.8) When the code import is complete, Note the Local Git URL

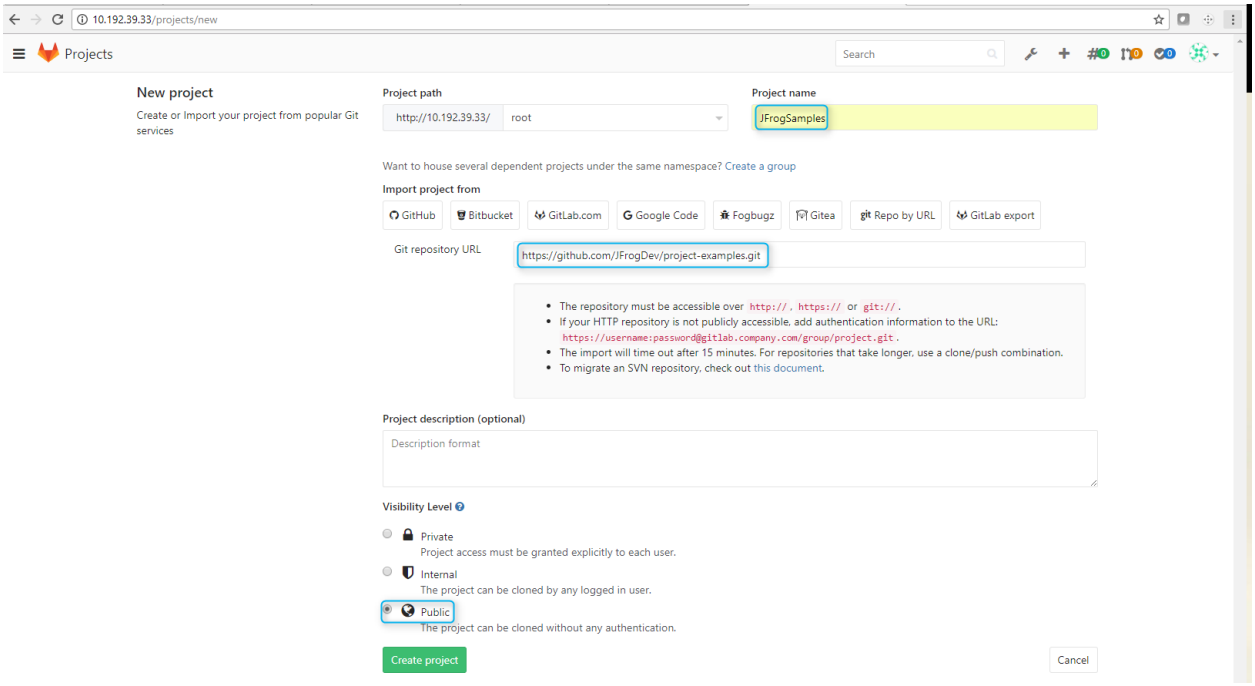
1.9) Adding a WebHook for automatic SCM Checkpoint Creation.

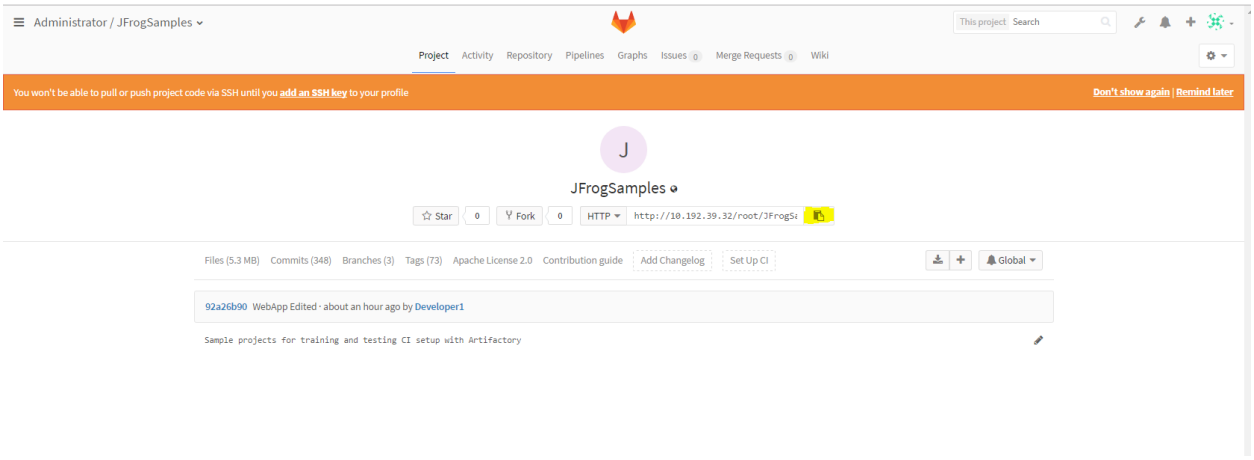
- To add a webhook in Jenkins, Click the Settings button at right corner of the GitLab project and select Integrations
- Add a the following Jenkins Job WebHook URL

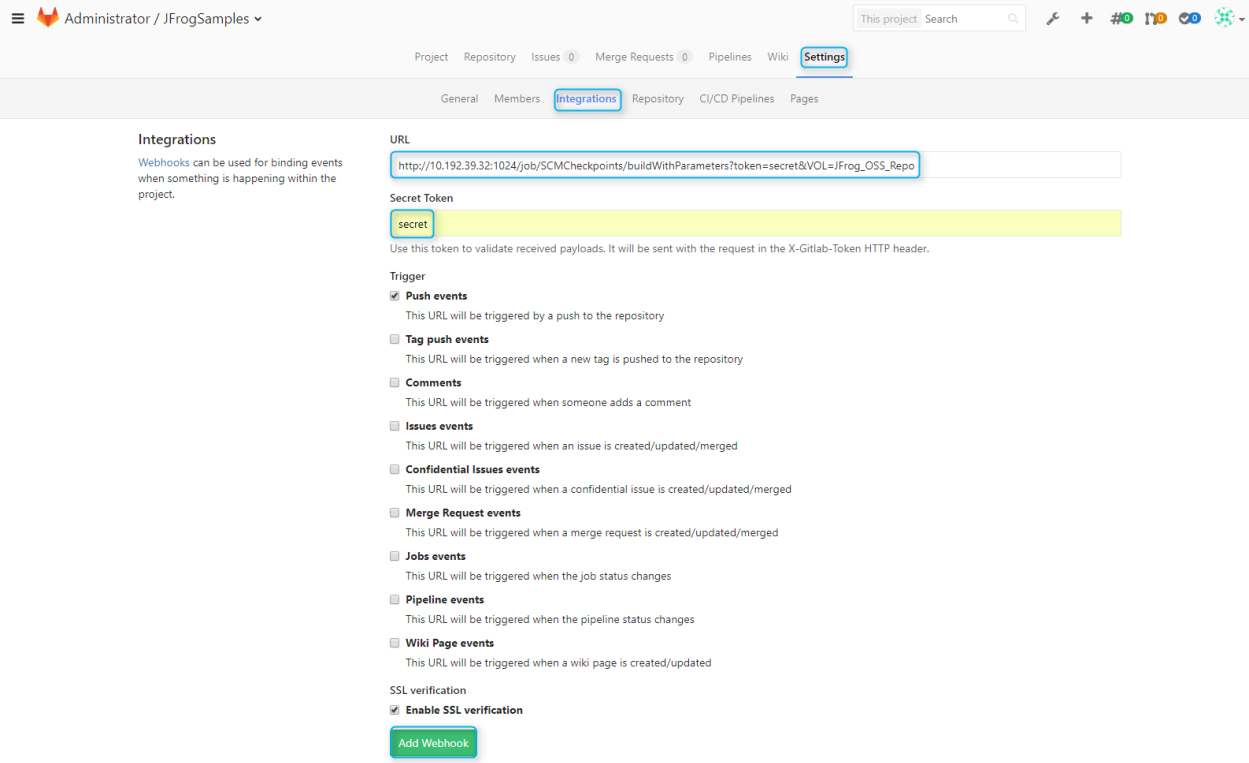
```
http://<<Jenkins-Master-IP>>:1024/job/SCMCheckpoints/
↪buildWithParameters?token=secret&VOL=<<SCM_Project_Name>>
```

- Add the Secret Token as "secret"
- Click Add WebHook
- This WebHook will automatically trigger a Jenkins Job(SCMCheckpoints) to create snapshot for every git push made in the SCM

Note: If you change your SCM Job name in Jenkins , use the same job name in this WebHook, as this will trigger a snapshot of your SCM volume.


10.192.39.33/projects/new
Projects
Search
New project
Create or Import your project from popular Git services
Project path
http://10.192.39.33/
root
Project name
JFrogSamples
Want to house several dependent projects under the same namespace? Create a group
Import project from
GitHub
Bitbucket
GitLab.com
Google Code
Fogbugz
Gitea
Repo by URL
GitLab export
Git repository URL
https://github.com/JFrogDev/project-examples.git
The repository must be accessible over http://, https:// or git://.
If your HTTP repository is not publicly accessible, add authentication information to the URL: https://username:password@gitlab.company.com/group/project.git.
The import will time out after 15 minutes. For repositories that take longer, use a clone/push combination.
To migrate an SVN repository, check out this document.
Project description (optional)
Description format
Visibility Level
Private
Project access must be granted explicitly to each user.
Internal
The project can be cloned by any logged in user.
Public
The project can be cloned without any authentication.
Create project
Cancel


Administrator / JFrogSamples
This project
Search
Project
Activity
Repository
Pipelines
Graphs
Issues 0
Merge Requests 0
Wiki
You won't be able to pull or push project code via SSH until you add an SSH key to your profile
Don't show again | Remind later
J
JFrogSamples
Star 0
Fork 0
HTTP
http://10.192.39.32/root/JFrogS
Files (5.3 MB)
Commits (348)
Branches (3)
Tags (73)
Apache License 2.0
Contribution guide
Add Changelog
Set Up CI
Global
92a26b90 WebApp Edited · about an hour ago by Developer1
Sample projects for training and testing CI setup with Artifactory



Navigate to SCMCheckpoints job on Jenkins UI . (http://xx.xx.xx.xx:1024/view/All/job/SCMCheckpoints/configure) , where xx.xx.xx.xx is the IP of your RHEL VM running the Jenkins OSS Container.

The build trigger in Authentication Token field should correspond to secret key set in GitLab Web-Hook

Make sure the SCM Section has the Git URL of the project as it collects the SHA ID from here.

---

2. CI Environment Setup

This pipeline will build a Continuous Integration Environment where a CI Build will run for the project

2.1) The Continuous Integration (Integrated Builds) stage will build a Docker Container named JFrog_2017_1 which acts as our CI-Environment with a NetApp Volume named JFrog_2017_1 mounted on it.

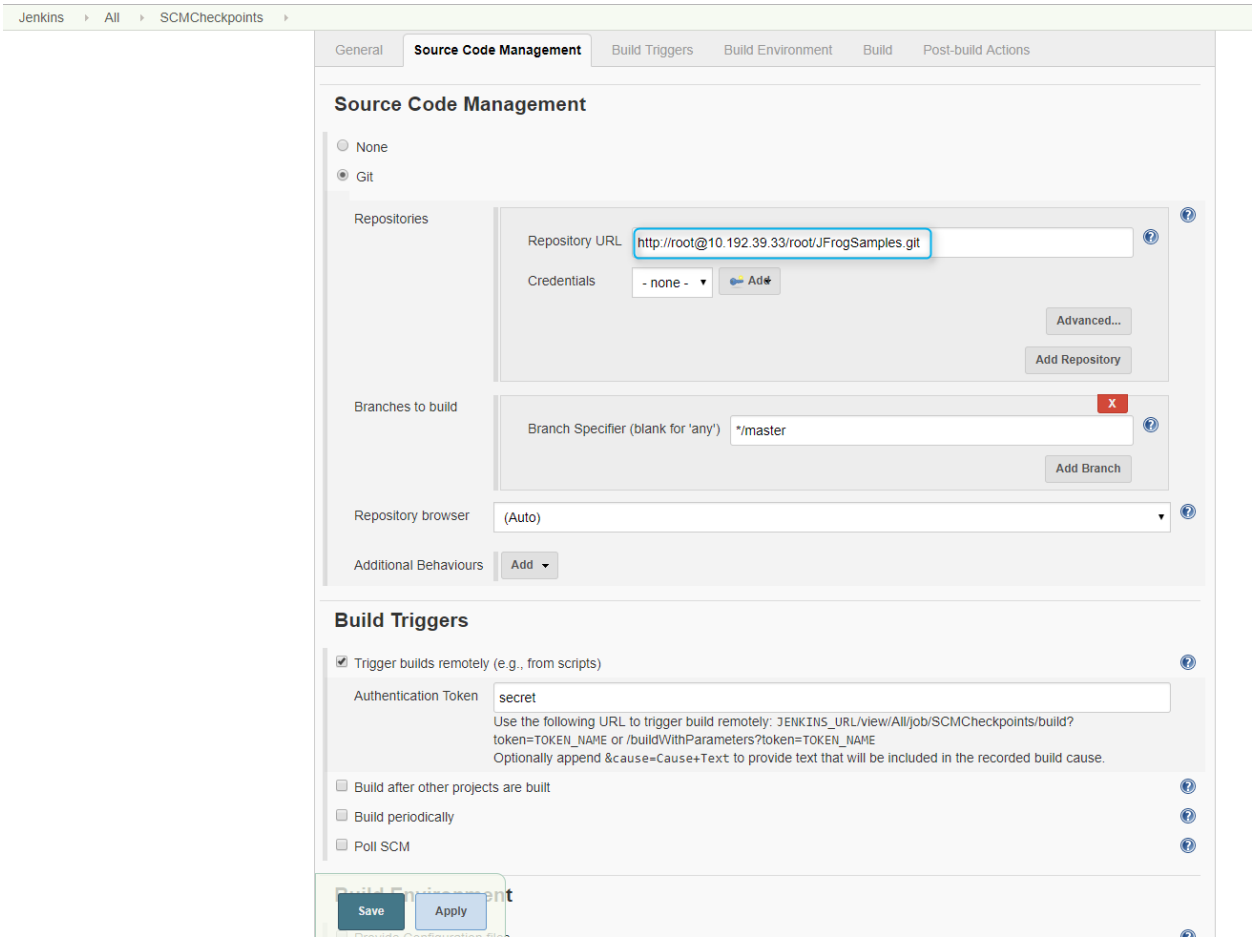
2.2) The CI Environment runs as a docker service on one of the swarm node. This can be verified by listing all docekr services on Swarm Manager node.:

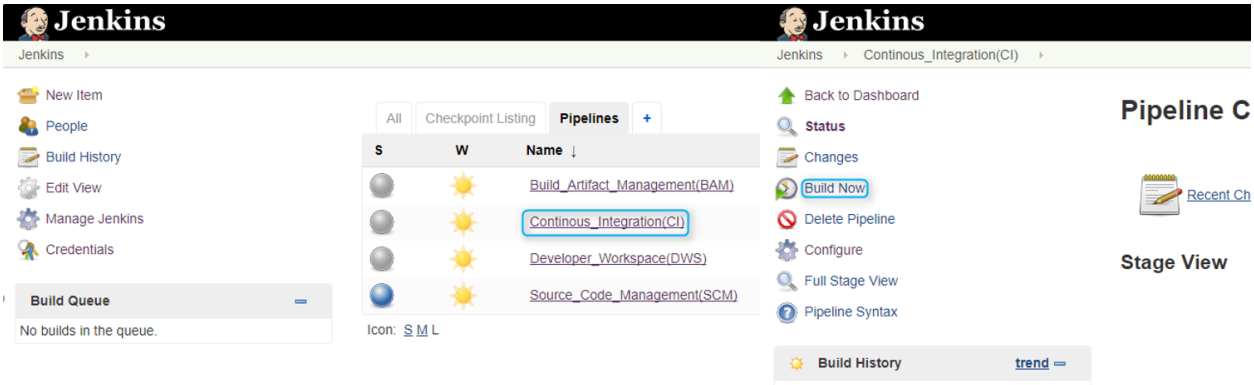
```
>>docker service ls
```

2.3) To check where the service is running , use the command

```
>>docker service ps <<CI-Environment-Job-Name>
```

2.4) A NetApp volume mount can be verified on the linux host by going inside the context of the container






```
[root@linux-node13 ~]# docker service ps JFrog_2017_1
ID                  NAME            IMAGE                                                   NODE          DESIRED STATE  CURRENT STATE
     ERROR
4nlh5ufsloquyd0aea8z59ays  JFrog_2017_1.1  devopsnetapponaws/netapp-jenkins_slave:autodiscover  linux-node12  Running        Running 3 minutes
ago
```

```
>>docker ps
>>docker exec -it <<Container-ID>> bash
>>df
```

```
[root@linux-node12 changes]# docker ps
CONTAINER ID        IMAGE                                                 COMMAND                  CREATED             STATUS              PORT
S               NAMES
bd614598d7df        devopsnetapponaws/netapp-jenkins_slave:autodiscover   "/bin/sh -c 'exec jav"   4 minutes ago       Up 4 minutes
                JFrog_2017_1.1.4nlh5ufsloquyd0aea8z59ays
[root@linux-node12 changes]# docker exec -it bd614598d7df bash
root@bd614598d7df:/#
root@bd614598d7df:/#
root@bd614598d7df:/# df -h
Filesystem                                                                                      Size  Used Avail Use% Mounted on
/dev/mapper/docker-253:0-134380772-bda58a8158aac6668178ac2940f9a93ca5707015b3617bf1490306000edae2e9   10G 1009M  9.1G  10% /
tmpfs                                                                                            32G     0   32G   0% /dev
tmpfs                                                                                            32G     0   32G   0% /sys/fs/cgroup
10.192.39.76:/JFrog_2017_1                                                                      4.2G  256K  4.2G   1% /workspace/JFrog_201
7_1
/dev/mapper/rhel-root                                                                            50G   36G   14G  72% /etc/hosts
shm                                                                                              64M     0   64M   0% /dev/shm
```

3. CI Build Setup

3.1) Create a Maven Job named "JFrog_CI_Build" in Jenkins by going to New Item > Create Maven Job

```
[root@linux-node12 changes]# docker ps
CONTAINER ID        IMAGE                                                 COMMAND                  CREATED             STATUS              PORT
S               NAMES
bd614598d7df        devopsnetapponaws/netapp-jenkins_slave:autodiscover   "/bin/sh -c 'exec jav"   4 minutes ago       Up 4 minutes
                JFrog_2017_1.1.4nlh5ufsloquyd0aea8z59ays
[root@linux-node12 changes]# docker exec -it bd614598d7df bash
root@bd614598d7df:/#
root@bd614598d7df:/#
root@bd614598d7df:/# df -h
Filesystem                                                                                      Size  Used Avail Use% Mounted on
/dev/mapper/docker-253:0-134380772-bda58a8158aac6668178ac2940f9a93ca5707015b3617bf1490306000edae2e9   10G 1009M  9.1G  10% /
tmpfs                                                                                            32G     0   32G   0% /dev
tmpfs                                                                                            32G     0   32G   0% /sys/fs/cgroup
10.192.39.76:/JFrog_2017_1                                                                      4.2G  256K  4.2G   1% /workspace/JFrog_201
7_1
/dev/mapper/rhel-root                                                                            50G   36G   14G  72% /etc/hosts
shm                                                                                              64M     0   64M   0% /dev/shm
```

Note: For this validation a sample Hello-World maven code from JFrog_Repo is used.

3.2) Configuring the JFrog_CI_Build Maven Job

Following sections need to be configured in the Maven Job:

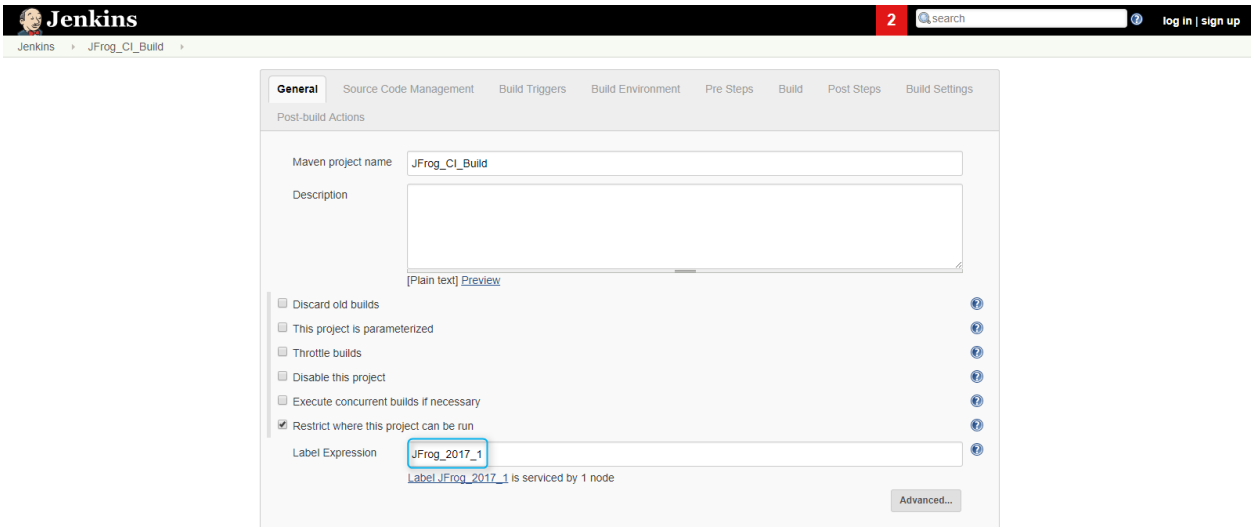
- Restrict the project to run in the previously created CI-Environment Enter the label : "JFrog_2017_1"
- Add Local Gitlab URL in the SCM section
- In Build Triggers select the option POLL SCM and set per minute polling schedule:

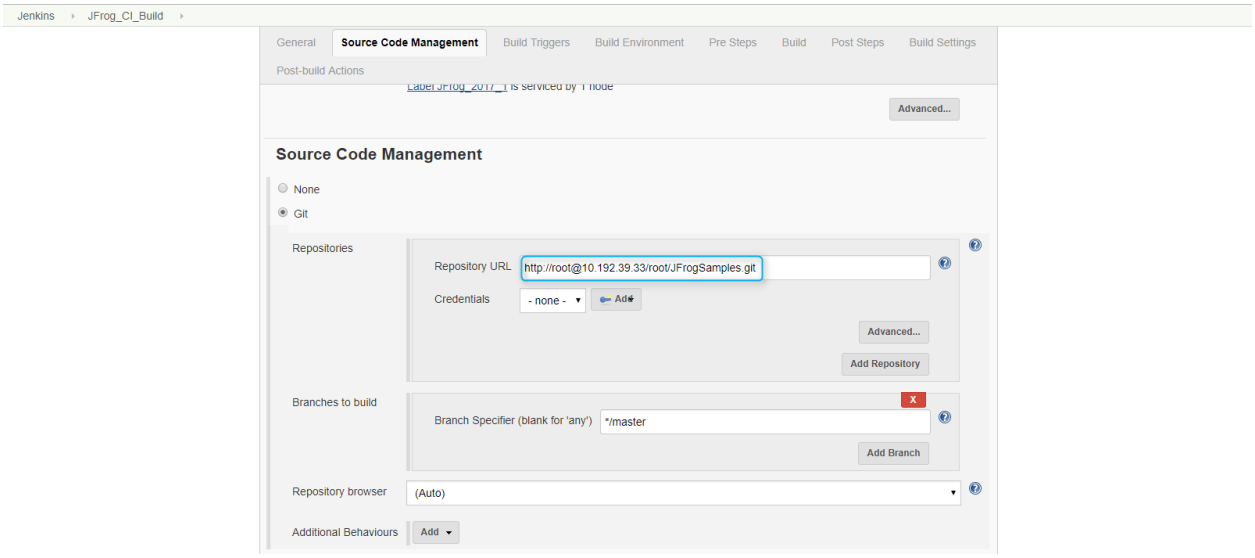
```
* * * * *
```

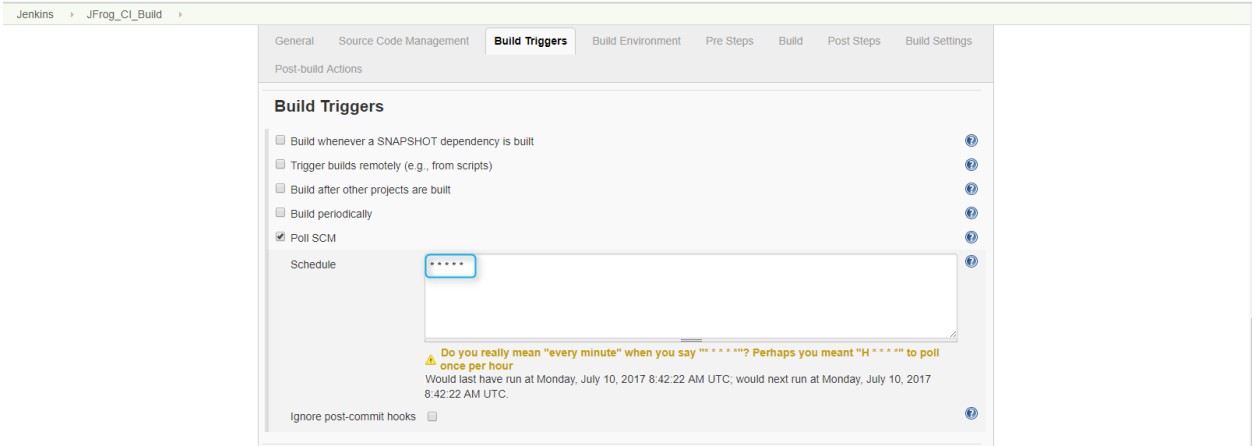
- In Build Environment Section, select Resolve Artifacts from Artifactory. Click Refresh Repositories Select the repositories to resolve the artifacts from Artifactory

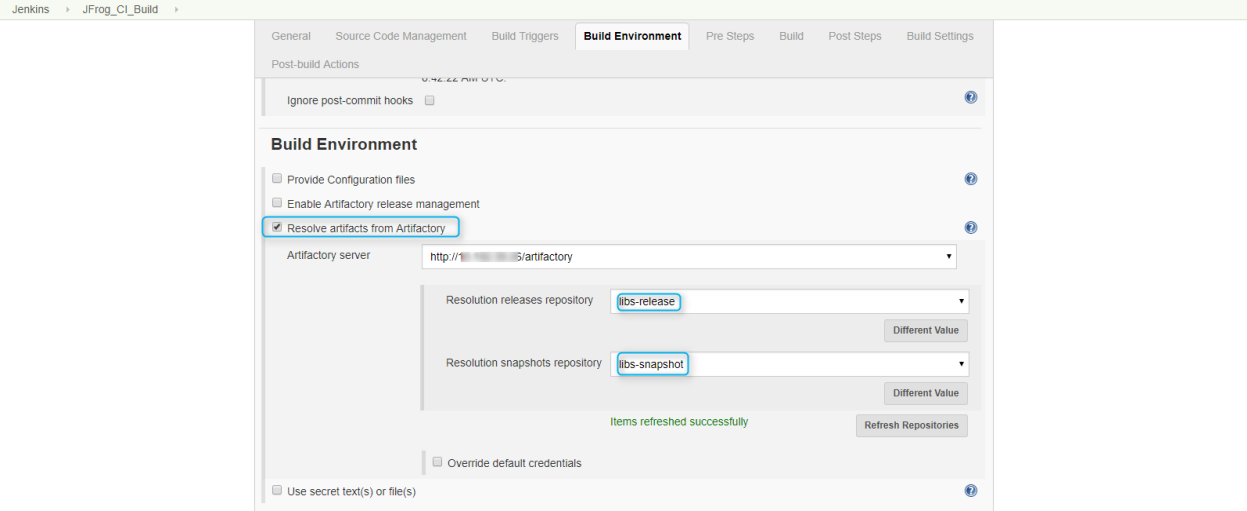
- In the Build Section, enter relative path of the pom.xml file and set the install goal

```
ROOT POM:       "maven-example/pom.xml"
Goals:          "install -DskipTests"
```


Jenkins
2
search
log in | sign up
Jenkins › JFrog_CI_Build ›
General
Source Code Management
Build Triggers
Build Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
Maven project name
JFrog_CI_Build
Description
[Plain text] Preview
Discard old builds
This project is parameterized
Throttle builds
Disable this project
Execute concurrent builds if necessary
Restrict where this project can be run
Label Expression
JFrog_2017_1
Label JFrog_2017_1 is serviced by 1 node
Advanced...


Jenkins › JFrog_CI_Build ›
General
Source Code Management
Build Triggers
Build Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
Advanced...
Source Code Management
None
Git
Repositories
Repository URL
http://root@10.192.39.33/root/JFrogSamples.git
Credentials
- none -
Add
Advanced...
Add Repository
Branches to build
Branch Specifier (blank for 'any')
*/master
Add Branch
Repository browser
(Auto)
Additional Behaviours
Add


Jenkins › JFrog_CI_Build ›
General
Source Code Management
Build Triggers
Build Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
Build Triggers
Build whenever a SNAPSHOT dependency is built
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Build periodically
Poll SCM
Schedule
* * * * *
Do you really mean "every minute" when you say "* * * * *"? Perhaps you meant "H * * * *" to poll once per hour
Would last have run at Monday, July 10, 2017 8:42:22 AM UTC; would next run at Monday, July 10, 2017 8:42:22 AM UTC.
Ignore post-commit hooks



- Select Use private Maven Repository checkbox and point it to use Local to Workspace. This ensures all the artifacts stay on a NetApp workspace.
- Click the Advanced tab and select use Custom Workspace, Enter the workspace as:

```
/workspace/<<CI-Environment-Name>>
```

  This ensures that the project is being built on a NetApp workspace.

Note:

- Fill in these values as per your environment/code structure
- The custom workspace field should be in the format "/workspace/ <<Your-CI-Environment-Job-Name>>

3.3) Configuring automatic creation of snapshots of successful builds.

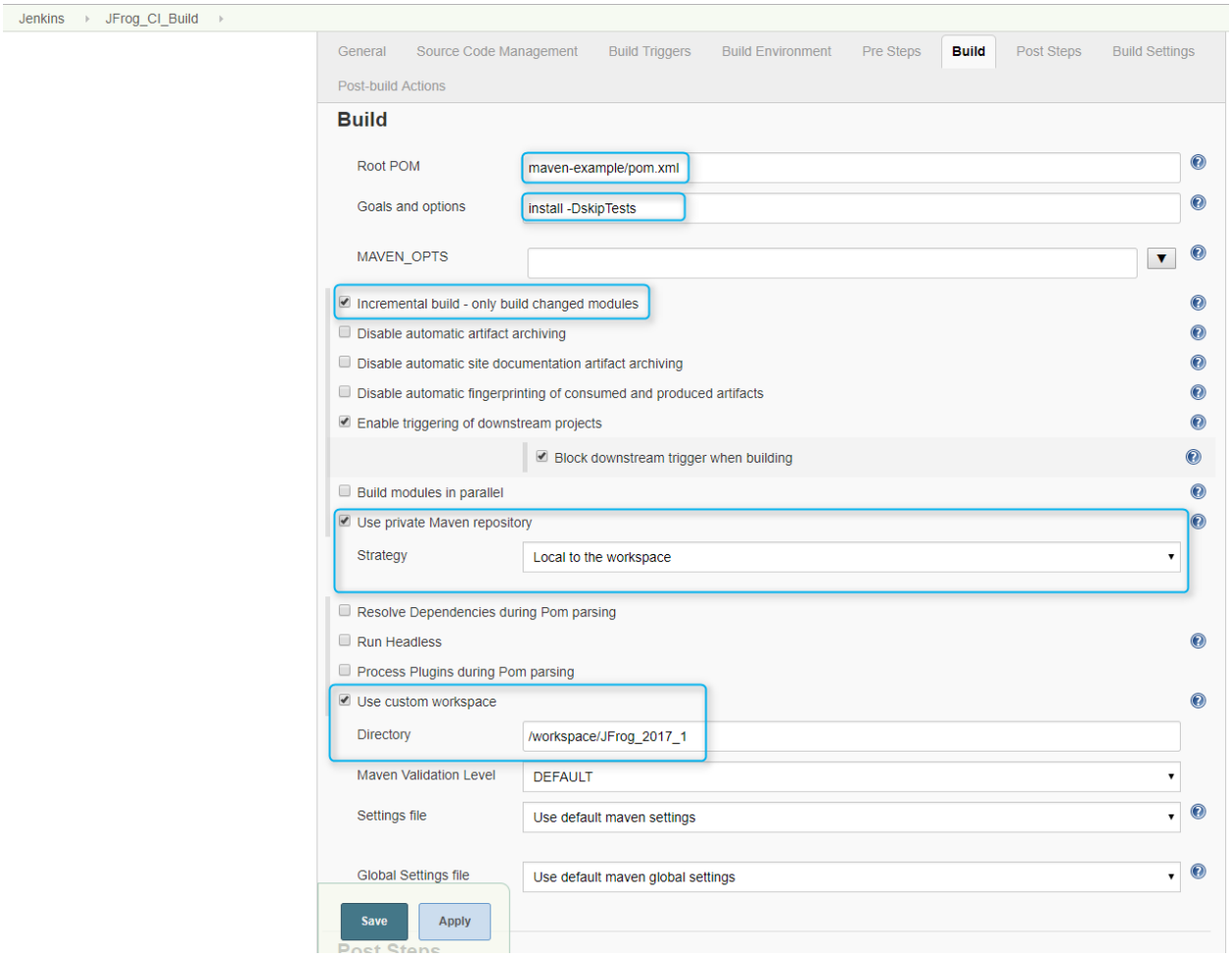
- Navigate to All Jenkins Jobs
- Select the job "Create_Build_Checkpoints"
- Select Configure
- Configure the folloeing sections of CreateBuildCheckpoints job:
  - In the general section, Select this project is parameterized option and set the default value as your <<CI-Environment-Name>>
  - In the Build Trigger section, Enter the name of CI Build Name. Select trigger only if build is stable.

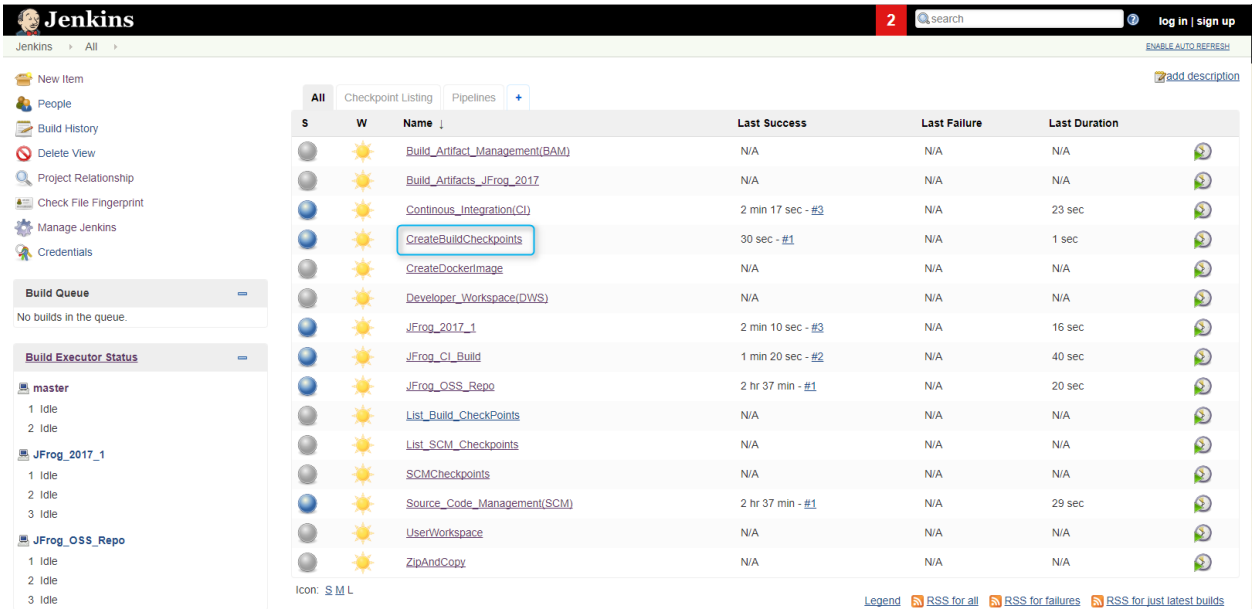
Note: Note on Multiple CI Builds

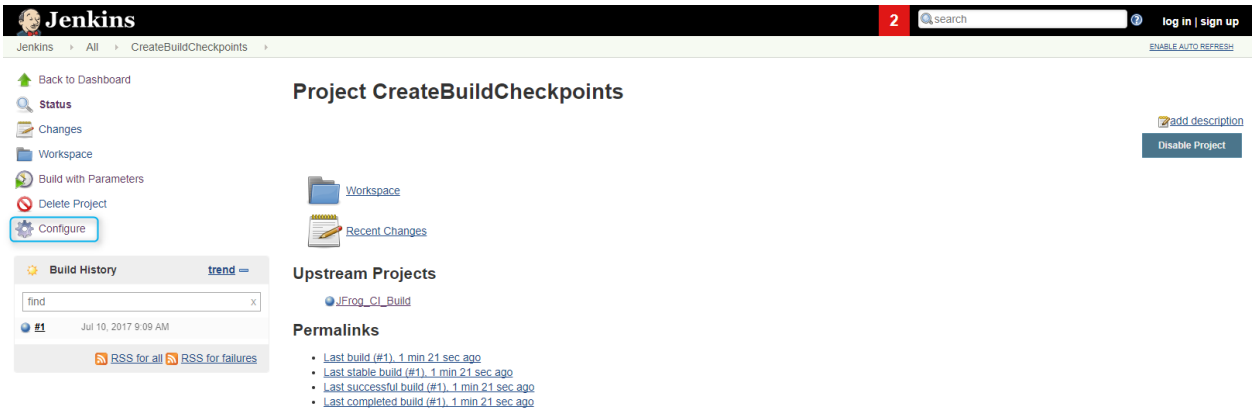
---

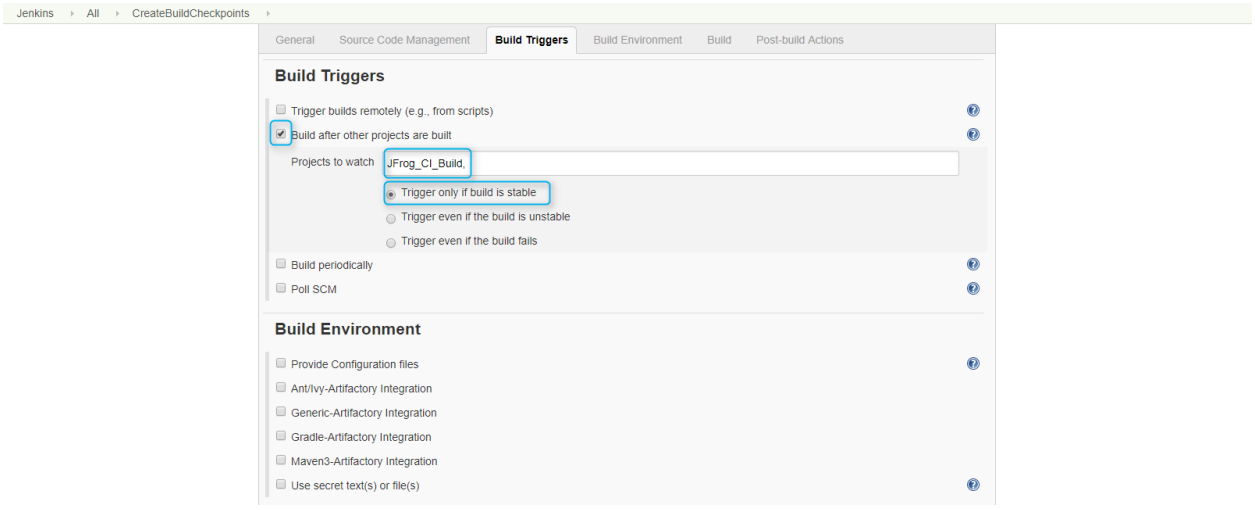
4. Developer Workspaces

This Pipeline Job will Create User Workspaces for Private Builds on NetApp FlexClones

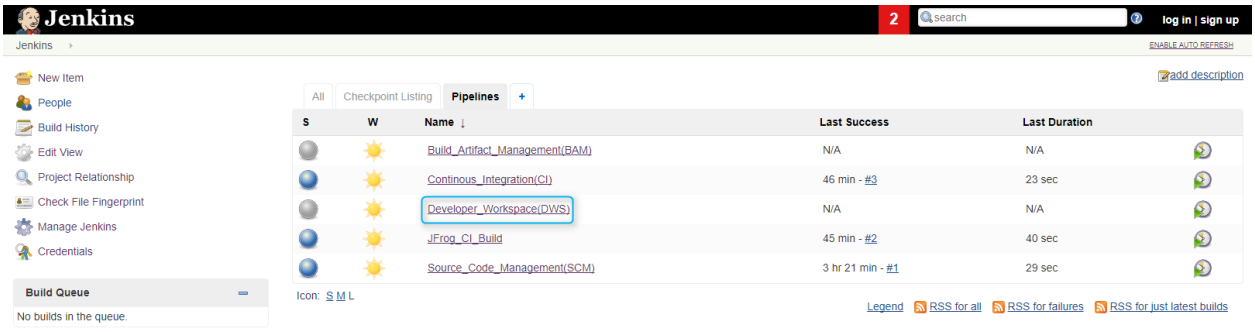

Jenkins ▸ JFrog_CI_Build ▸
General
Source Code Management
Build Triggers
Build Environment
Pre Steps
Build
Post Steps
Build Settings
Post-build Actions
Build
Root POM
maven-example/pom.xml
Goals and options
install -DskipTests
MAVEN_OPTS
Incremental build - only build changed modules
Disable automatic artifact archiving
Disable automatic site documentation artifact archiving
Disable automatic fingerprinting of consumed and produced artifacts
Enable triggering of downstream projects
Block downstream trigger when building
Build modules in parallel
Use private Maven repository
Strategy
Local to the workspace
Resolve Dependencies during Pom parsing
Run Headless
Process Plugins during Pom parsing
Use custom workspace
Directory
/workspace/JFrog_2017_1
Maven Validation Level
DEFAULT
Settings file
Use default maven settings
Global Settings file
Use default maven global settings
Save
Apply

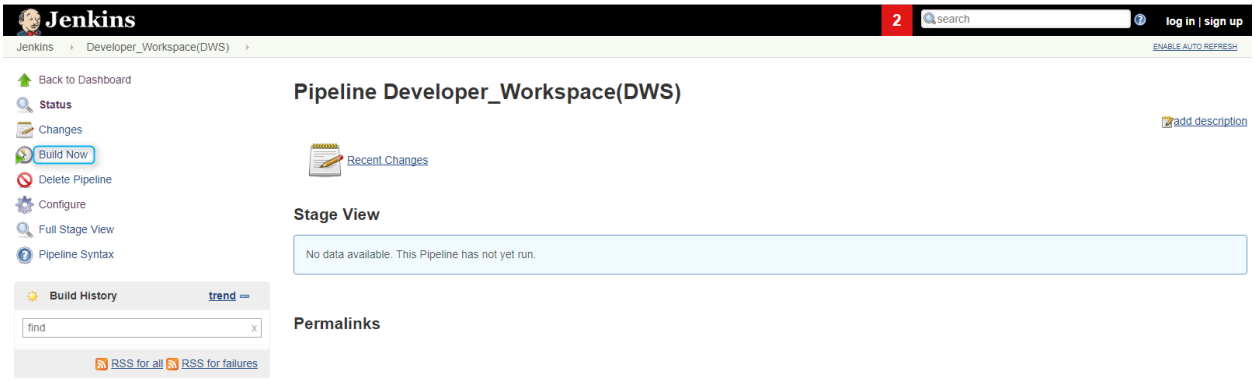

Jenkins
2
search
log in | sign up
Jenkins ▸ All ▸
ENABLE AUTO REFRESH
New Item
People
Build History
Delete View
Project Relationship
Check File Fingerprint
Manage Jenkins
Credentials
Build Queue
No builds in the queue.
Build Executor Status
master
1 Idle
2 Idle
JFrog_2017_1
1 Idle
2 Idle
3 Idle
JFrog_OSS_Repo
1 Idle
2 Idle
3 Idle
add description
All
Checkpoint Listing
Pipelines
S
W
Name ↓
Last Success
Last Failure
Last Duration
Build_Artifact_Management(BAM) N/A N/A N/A
Build_Artifacts_JFrog_2017 N/A N/A N/A
Continous_Integration(CI) 2 min 17 sec - #3 N/A 23 sec
CreateBuildCheckpoints 30 sec - #1 N/A 1 sec
CreateDockerImage N/A N/A N/A
Developer_Workspace(DWS) N/A N/A N/A
JFrog_2017_1 2 min 10 sec - #3 N/A 16 sec
JFrog_CI_Build 1 min 20 sec - #2 N/A 40 sec
JFrog_OSS_Repo 2 hr 37 min - #1 N/A 20 sec
List_Build_CheckPoints N/A N/A N/A
List_SCM_Checkpoints N/A N/A N/A
SCMCheckpoints N/A N/A N/A
Source_Code_Management(SCM) 2 hr 37 min - #1 N/A 29 sec
UserWorkspace N/A N/A N/A
ZipAndCopy N/A N/A N/A
Icon: S M L
Legend
RSS for all
RSS for failures
RSS for just latest builds


Jenkins
2
search
log in | sign up
Jenkins › All › CreateBuildCheckpoints ›
ENABLE AUTO REFRESH
Back to Dashboard
Status
Changes
Workspace
Build with Parameters
Delete Project
Configure
Build History
trend
find
#1
Jul 10, 2017 9:09 AM
RSS for all
RSS for failures
Project CreateBuildCheckpoints
add description
Disable Project
Workspace
Recent Changes
Upstream Projects
JFrog_CI_Build
Permalinks
Last build (#1), 1 min 21 sec ago
Last stable build (#1), 1 min 21 sec ago
Last successful build (#1), 1 min 21 sec ago
Last completed build (#1), 1 min 21 sec ago


Jenkins › All › CreateBuildCheckpoints ›
General
Source Code Management
Build Triggers
Build Environment
Build
Post-build Actions
Build Triggers
Trigger builds remotely (e.g., from scripts)
Build after other projects are built
Projects to watch
JFrog_CI_Build,
Trigger only if build is stable
Trigger even if the build is unstable
Trigger even if the build fails
Build periodically
Poll SCM
Build Environment
Provide Configuration files
Ant/Ivy-Artifactory Integration
Generic-Artifactory Integration
Gradle-Artifactory Integration
Maven3-Artifactory Integration
Use secret text(s) or file(s)

4.1) Select Developer Workspace Pipeline and Click Build Now





4.2) This pipeline requires following inputs:

Input	Default	What it does
UID	301	UID for the Developer
GID	300	GID for the Developer
CI Dev Branch Name	JFrog_2017_1	Name of CI Environment
Workspace Name	Dev1	Name of workspace to create

4.3) Click Proceed and Hover over the Pipeline stage to select build checkpoint to create a workspace and click Proceed.

4.4) A new docker service with a Private workspace for the Developer will be available on the Linux host. This can be checked by running following command on the host.

```
>>docker service ls
>>docker service ps <<Service_Name>>
```

4.5) The contents of this docker service can be verified by going inside the context of the container.

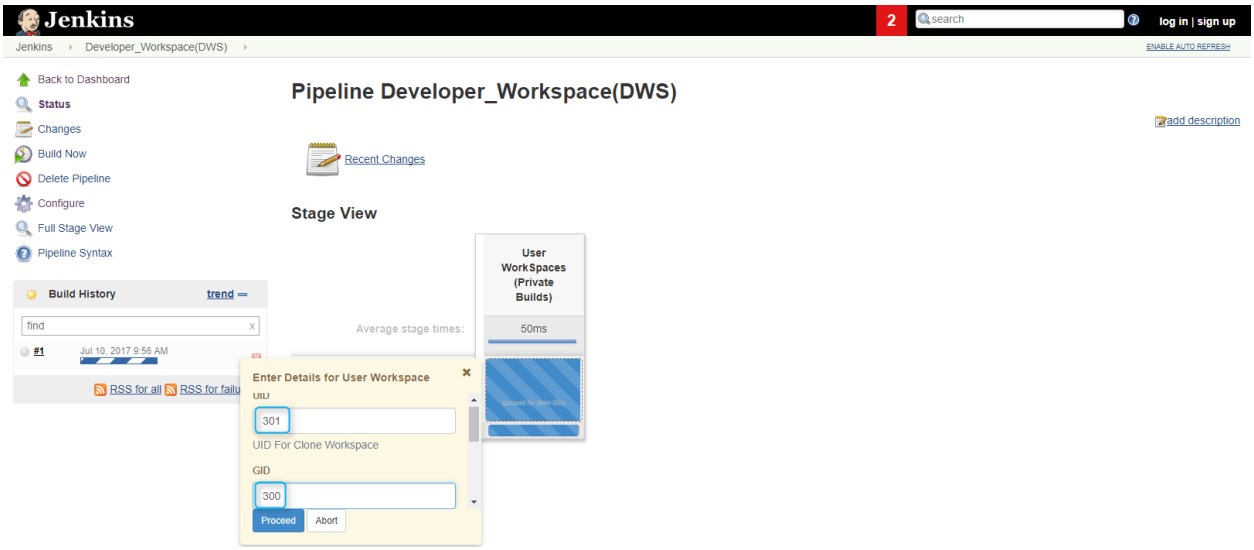
```
>> docker exec -it <<Service Container ID>> bash
```

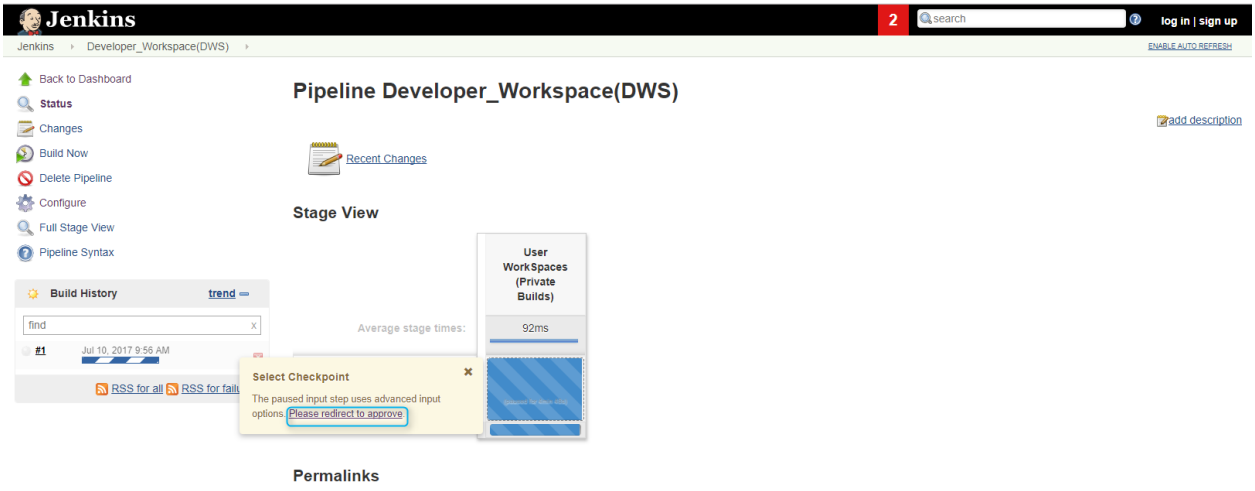
4.6) All the source files and artifacts will be present at

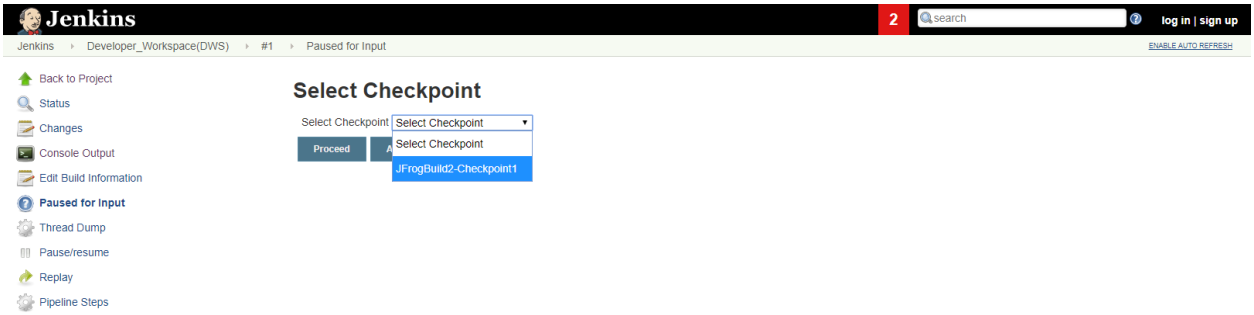
```
/workspace/<<Developer-Workspace-Name>>
```

4.7) Developer can make changes to the code and then commit them.

- Set the git username and email for the Developer.








```
[root@linux-node13 ~]# docker service ls
ID            NAME             REPLICAS  IMAGE                                                COMMAND
983fppyl2qmq  JFrog_2017_1     1/1       devopsnetapponaws/netapp-jenkins_slave:autodiscover
c21t35otkuiu  Dev1_JFrog_2017_1  1/1       devopsnetapponaws/netapp-jenkins_slave:autodiscover
dbku5dbb8pwd  jenkins          1/1       devopsnetapponaws/netapp-jenkins-plugin-2.0:oss
[root@linux-node13 ~]# docker service ps Dev1_JFrog_2017_1
ID                  NAME                IMAGE                                                NODE          DESIRED STATE  CURRENT STATE
          ERROR
f3doj33wmgiax3pyearooexx0  Dev1_JFrog_2017_1.1  devopsnetapponaws/netapp-jenkins_slave:autodiscover  linux-node12  Running        Running 4 mi
utes ago
[root@linux-node13 ~]#
```

```
[root@linux-node12 changes]# docker exec -it 7abea287b1b8 bash
root@7abea287b1b8:/#
root@7abea287b1b8:/#
root@7abea287b1b8:/#
root@7abea287b1b8:/# cd workspace/Dev1_JFrog_2017_1/
root@7abea287b1b8:/workspace/Dev1_JFrog_2017_1#
root@7abea287b1b8:/workspace/Dev1_JFrog_2017_1#
root@7abea287b1b8:/workspace/Dev1_JFrog_2017_1# ls -ltra
total 76
-rw-r--r--  1  301  300  202 Jul 10 09:05 .gitignore
-rw-r--r--  1  301  300  469 Jul 10 09:05 CONTRIBUTING.md
-rw-r--r--  1  301  300   67 Jul 10 09:05 README
drwxr-xr-x  5  301  300 4096 Jul 10 09:05 artifactory-maven-plugin-example
drwxr-xr-x  2  301  300 4096 Jul 10 09:05 bash-example
drwxr-xr-x  3  301  300 4096 Jul 10 09:05 build-info-java-example
drwxr-xr-x  8  301  300 4096 Jul 10 09:05 circleci-example
drwxr-xr-x  5  301  300 4096 Jul 10 09:05 cpp-example
drwxr-xr-x  4  301  300 4096 Jul 10 09:05 gradle-examples
drwxr-xr-x  4  301  300 4096 Jul 10 09:05 ivy-example
drwxr-xr-x 14  301  300 4096 Jul 10 09:05 jenkins-pipeline-examples
drwxr-xr-x  6  301  300 4096 Jul 10 09:05 maven-example-bintray-info
drwxr-xr-x  5  301  300 4096 Jul 10 09:05 maven-example
drwxr-xr-x  8  301  300 4096 Jul 10 09:05 msbuild-example
drwxr-xr-x  5  301  300 4096 Jul 10 09:05 nuget-example
drwxr-xr-x  4  301  300 4096 Jul 10 09:05 python-example
drwxr-xr-x 17  301  300 4096 Jul 10 09:05 .
drwxr-xr-x  3  301  300 4096 Jul 10 09:05 sbt-example
drwxr-xr-x  8  301  300 4096 Jul 10 09:05 .git
drwxr-xr-x  3 root root   31 Jul 10 10:01 ..
root@7abea287b1b8:/workspace/Dev1_JFrog_2017_1#
```

```
>> git config -global user.name "Dev1"

>> git config -global user.email "Dev1@netapp.com"
```

4.8) For this validation we will make changes to the master branch

```
>> git checkout master
```

4.9) As a example, a small Hello World to Hello NetApp code change is shown as example.

```
>> vi maven-example/multi3/src/main/java/artifactory/test/Multi3.java
```

```
package artifactory.test;

/**
 * Hello world!
 */
public class Multi3 {
    public static void main(String[] args) {
        new Multi1();
        System.out.println("Hello World!");
    }
}
```

Before Codechange

```
package artifactory.test;

/**
 * Hello world!
 */
public class Multi3 {
    public static void main(String[] args) {
        new Multi1();
        System.out.println("Hello NetApp!");
    }
}
```

After Codechange

4.10) Commit the changes.:

```
>> git commit -all
```

```
root@7abea287b1b8:/workspace/Dev1_JFrog_2017_1/maven-example/multi3/src/main/java/artifactory/test# git commit --all
[detached HEAD 8d5d9b5] Hello World -> Hello NetAppp
 1 file changed, 1 insertion(+), 1 deletion(-)
```

4.11) If the changes pass the pre-push hook, then developer can go ahead and push the code to SCM.

```
root@7abea287b1b8:/workspace/Dev1_JFrog_2017_1/maven-example/multi3/src/main/java/artifactory/test# git push --all
Password for 'http://root@10.192.39.33':
Counting objects: 10, done.
Delta compression using up to 32 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 728 bytes | 0 bytes/s, done.
Total 10 (delta 3), reused 0 (delta 0)
To http://root@10.192.39.33/root/JFrogSamples.git
   36b1aef..a1ad944  master -> master
root@7abea287b1b8:/workspace/Dev1_JFrog_2017_1/maven-example/multi3/src/main/java/artifactory/test#
```

4.12) This recent push will reflect in GitLab UI.

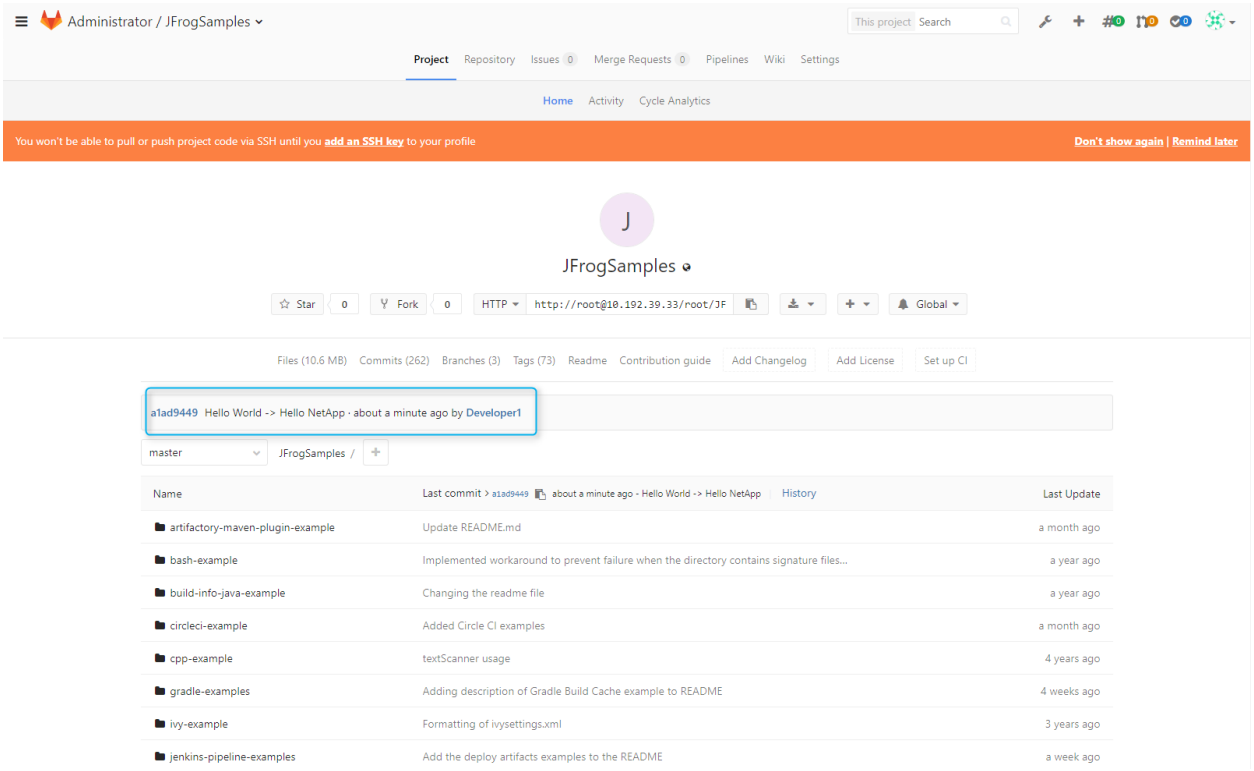
```
>> http://<<Jenkins-Host-IP/>>
```

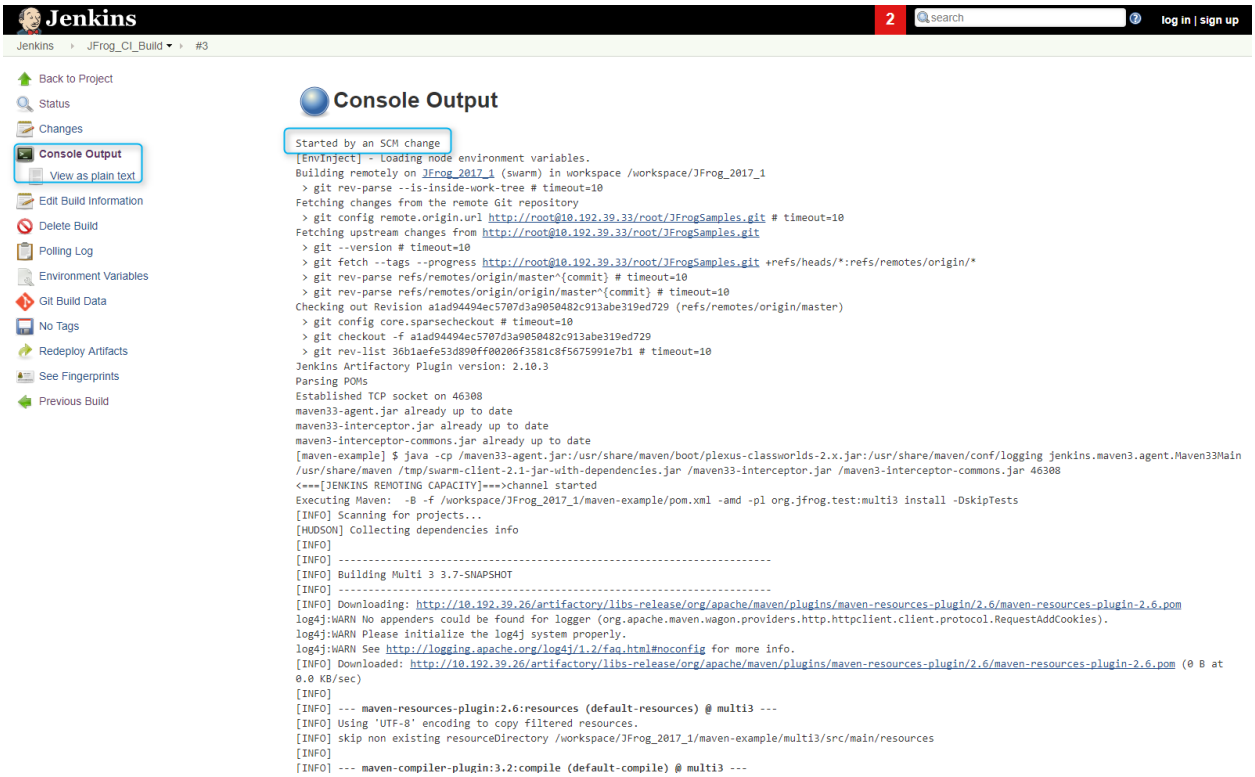
4.13) After the Git Push, the CI job will be triggered automatically as there was a change in SCM and the polling is done for every minute.

5. Build Artifact Management Pipeline

This pipeline will:

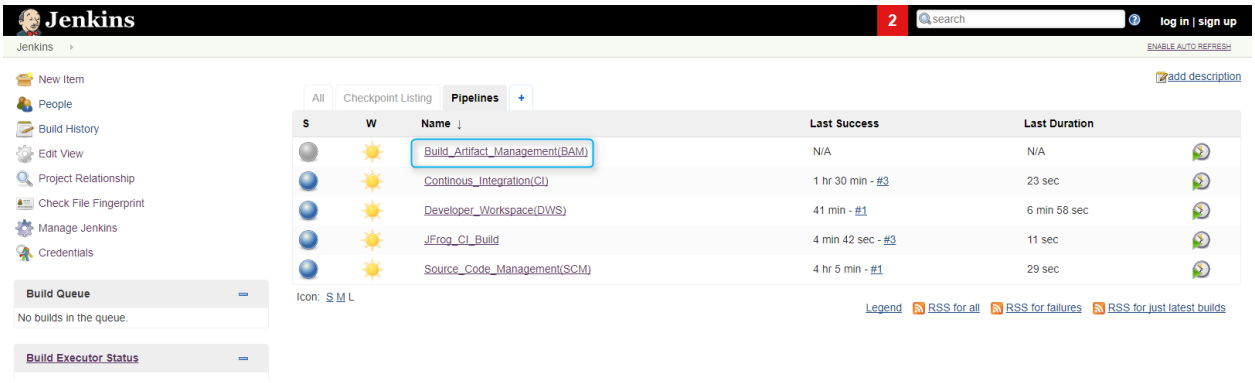
1. Create a Build Artifact Container
2. Create Zip of all the volume contents


Administrator / JFrogSamples
Project Repository Issues Merge Requests Pipelines Wiki Settings
Home Activity Cycle Analytics
You won't be able to pull or push project code via SSH until you add an SSH key to your profile
Don't show again | Remind later
JFrogSamples
Star Fork HTTP http://root@10.192.39.33/root/JF Global
Files (10.6 MB) Commits (262) Branches (3) Tags (73) Readme Contribution guide Add Changelog Add License Set up CI
a1ad9449 Hello World -> Hello NetApp · about a minute ago by Developer1
master JFrogSamples /
Name Last commit a1ad9449 about a minute ago - Hello World -> Hello NetApp History Last Update
artifactory-maven-plugin-example Update README.md a month ago
bash-example Implemented workaround to prevent failure when the directory contains signature files... a year ago
build-info-java-example Changing the readme file a year ago
circleci-example Added Circle CI examples a month ago
cpp-example textScanner usage 4 years ago
gradle-examples Adding description of Gradle Build Cache example to README 4 weeks ago
ivy-example Formatting of ivysettings.xml 3 years ago
jenkins-pipeline-examples Add the deploy artifacts examples to the README a week ago


Jenkins
2
search
log in | sign up
Jenkins JFrog_CI_Build #3
Back to Project
Status
Changes
Console Output
View as plain text
Edit Build Information
Delete Build
Polling Log
Environment Variables
Git Build Data
No Tags
Redeploy Artifacts
See Fingerprints
Previous Build
Console Output
Started by an SCM change
[EnvInject] - Loading node environment variables.
Building remotely on JFrog_2017_1 (swarm) in workspace /workspace/JFrog_2017_1
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url http://root@10.192.39.33/root/JFrogSamples.git # timeout=10
Fetching upstream changes from http://root@10.192.39.33/root/JFrogSamples.git
> git --version # timeout=10
> git fetch --tags --progress http://root@10.192.39.33/root/JFrogSamples.git +refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision a1ad94494ec5707d3a9050482c913abe319ed729 (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f a1ad94494ec5707d3a9050482c913abe319ed729
> git rev-list 36b1aefe53d890ff00206f3581c8f5675991e7b1 # timeout=10
Jenkins Artifactory Plugin version: 2.10.3
Parsing POMs
Established TCP socket on 46308
maven33-agent.jar already up to date
maven33-interceptor.jar already up to date
maven3-interceptor-commons.jar already up to date
[maven-example] $ java -cp /maven33-agent.jar:/usr/share/maven/boot/plexus-classworlds-2.x.jar:/usr/share/maven/conf/logging jenkins.maven3.agent.Maven33Main
/usr/share/maven /tmp/swarm-client-2.1-jar-with-dependencies.jar /maven33-interceptor.jar /maven3-interceptor-commons.jar 46308
<===[JENKINS REMOTING CAPACITY]===>channel started
Executing Maven: -B -f /workspace/JFrog_2017_1/maven-example/pom.xml -amd -pl org.jfrog.test:multi3 install -DskipTests
[INFO] Scanning for projects...
[HUDSON] Collecting dependencies info
[INFO]
[INFO] --
[INFO] Building Multi 3 3.7-SNAPSHOT
[INFO] --
[INFO] Downloading: http://10.192.39.26/artifactory/libs-release/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.pom
log4j:WARN No appenders could be found for logger (org.apache.maven.wagon.providers.http.httpclient.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
[INFO] Downloaded: http://10.192.39.26/artifactory/libs-release/org/apache/maven/plugins/maven-resources-plugin/2.6/maven-resources-plugin-2.6.pom (0 B at
0.0 KB/sec)
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ multi3 ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /workspace/JFrog_2017_1/maven-example/multi3/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.2:compile (default-compile) @ multi3 ---

3. Create Docker Image of running CI Environment

5.1) Select Build Artifact Management Pipeline and click build now



5.2) This pipeline requires following inputs:

Input	Default	What it does
CI Dev Branch to be Artifact	JFrog_2017_1	Name of CI Build Environment to Artifact
Build Artifact Container	Build_Artifact_JFrog_2017_1	This container will store
Name of Zip File to Create	JFrog_2017_1.zip	A Zip file containing all contents of volumes is created.
Docker Image Name to Create	<<Artifactory-IP:PORT>>/image1:version1	Creates docker image of build environment with this name.
Checkpoint	Select from a DropDown	Creates a temporary clone from this checkpoint.

5.3) Build the Pipeline

5.4) If the build is successful, a docker image will be pushed to the docker-dev repo in Artifactory.

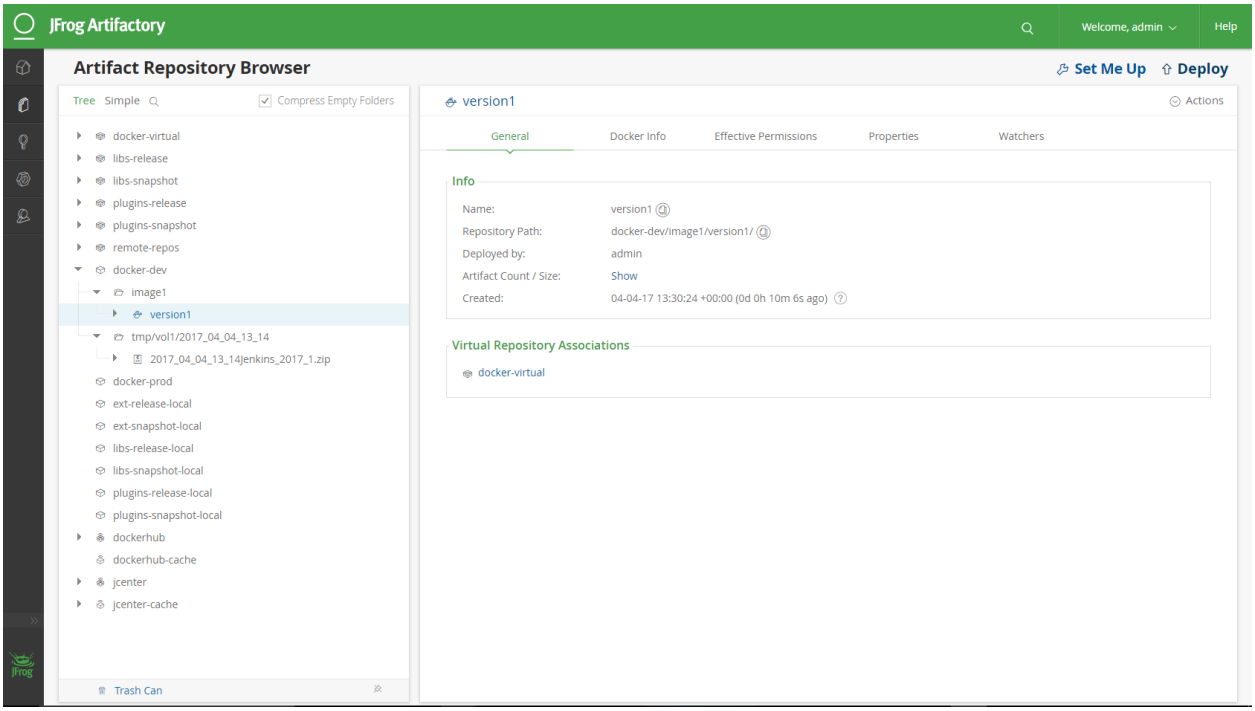
5.5) To restore your Build Environment using the docker image, login to any linux host and use the following command

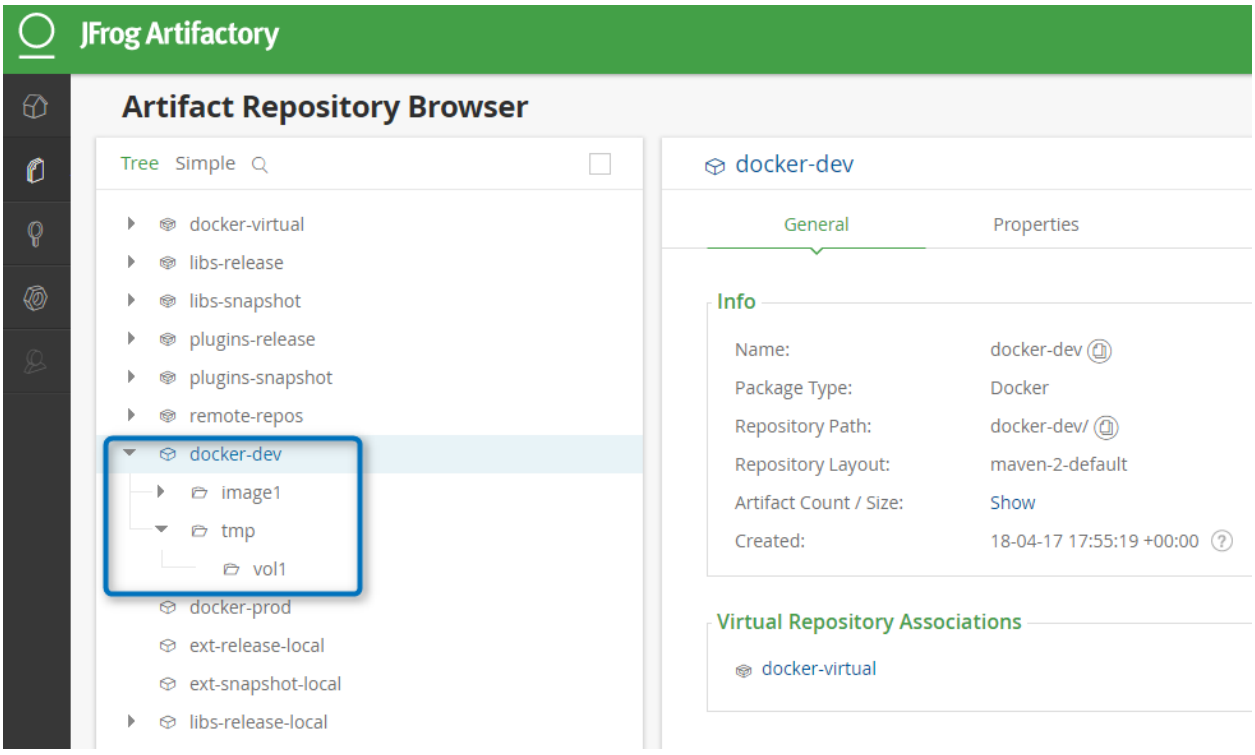
```
docker run  -t -d -e masterip=http://<<jenkins-master-ip>>:1024 -e␣
↪slavename=JFrog_2017_1_copy --name JFrog_2017_1_copy <<Artifactory-Server-
↪URL:Port>>/image1:version1
```

5.6) All the build data is stored as a timestamped zip file in Artifactory in the same docker repo. This zip can be downloaded/curl/wget to your backed up container.

Pre-Packaged Plugins

- The NetApp Jenkins Master Docker Image is pre-packaged with following Jenkins Plugins:
  1. **Swarm.** [2.1] This plugin enables slaves to auto-discover nearby Jenkins master and join it automatically
  2. **Pipeline.** [2.5] Pipeline plugin(workflow-aggregator) is a suite of plugins used create Pipeline Jobs in Jenkins
  3. **Git.** [2.2.0] Git plugin is used to conduct GIT operations with Jenkins


JFrog Artifactory
Welcome, admin
Help
Artifact Repository Browser
Set Me Up
Deploy
Tree
Simple
Compress Empty Folders
docker-virtual
libs-release
libs-snapshot
plugins-release
plugins-snapshot
remote-repos
docker-dev
image1
version1
tmp/vol1/2017_04_04_13_14
2017_04_04_13_14Jenkins_2017_1.zip
docker-prod
ext-release-local
ext-snapshot-local
libs-release-local
libs-snapshot-local
plugins-release-local
plugins-snapshot-local
dockerhub
dockerhub-cache
jcenter
jcenter-cache
Trash Can
version1
Actions
General
Docker Info
Effective Permissions
Properties
Watchers
Info
Name:
version1
Repository Path:
docker-dev/image1/version1/
Deployed by:
admin
Artifact Count / Size:
Show
Created:
04-04-17 13:30:24 +00:00 (0d 0h 10m 6s ago)
Virtual Repository Associations
docker-virtual


JFrog Artifactory
Artifact Repository Browser
Tree
Simple
docker-virtual
libs-release
libs-snapshot
plugins-release
plugins-snapshot
remote-repos
docker-dev
image1
tmp
vol1
docker-prod
ext-release-local
ext-snapshot-local
libs-release-local
docker-dev
General
Properties
Info
Name:
docker-dev
Package Type:
Docker
Repository Path:
docker-dev/
Repository Layout:
maven-2-default
Artifact Count / Size:
Show
Created:
18-04-17 17:55:19 +00:00
Virtual Repository Associations
docker-virtual

4. **Extended Choice Parameter Plugin. :0.76** This plugin provides an option of having DropDown input sections in pipelines
5. **Artifactory. :2.12.1** Artifactory plugin resolves the build artifacts from local instance of JFrog Artifactory
6. **MultiJob Plugin. :1.24** MultiJob plugin lets you have multiple types of job configurations in a single job

- **To Bundle more plugins in the Jenkins-Master Docker image :-**
    1. Open the Dockerfile in any text editor
    2. Find the line with plugin install script

```
RUN /usr/local/bin/install-plugins.sh workflow-aggregator:2.5
```

    3. Append your plugin-id:plugin-version to the the above line, e.g if you wish to package the *blueocean* plugin in Jenkins Master

```
RUN /usr/local/bin/install-plugins.sh workflow-aggregator:2.5 blueocean:1.
↪0
```

    4. Save the Dockerfile
    5. Build a new Docker image.

Support

All the support for this plugin is provided via Slack in the #ci-cd channel.