Continuous Integration Pipeline using Jenkins and ONTAP Technologies with Zero Storage Touch

October 2017 | SL10322 Version 1.0

TABLE OF CONTENTS

1 Introduction

This lab introduces Continuous Integration through Jenkins running on NetApp ONTAP 9.1®.

Continuous Integration (CI) is a developer workflow wherein developers make small changes to code, test it, and then merge the changes to the main code base. This iterative process enables developers to:

- Identify bugs at an early stage of the development cycle.
- Improve code quality through iterative testing.

Both of these items result in improved business efficiency, as smaller code changes are easier to understand, implement, and test. In addition, the earlier a bug is detected, the smaller the cost and productivty impact required to fix it.

Running a CI process on NetApp storage provides the following additional efficiency advantages to a business:

1. Improves developer productivity. In a CI environment, each user requires a user workspace in which to develop his or her code. Provisioning a workspace is time consuming; the administrator must first create it, and then the developer needs to customize it with the necessary source code, libraries, compilers, tools, and config files needed to start code development. With NetApp, businesses can apply this customization to a master workspace template, and then use NetApp FlexClones to rapidly clone this pre-customized workspace for as many developers as needed. This reduces the time to provision a workspace from hours and days to just seconds.
2. Reduces build time. Version control systems like Git host source code, not object and binary files. So, developers who check out code into their workspace must perform a full build in order to test their code changes. This is expensive because full builds are time consuming and resource intensive (compute, network, and storage space). With NetApp that full build can be done once and incorporated into the master workspace template, providing the developer with a pre-generated full build in any deployed (FlexCloned) workspace. Any code changes the developer makes will only require an incremental build in his personal workspace to test, while protecting the master build tree from pollution by untested code changes.
3. Reduces infrastructure costs. NetApp data volumes (FlexVols), Snapshots, and Flexclones are thin provisioned, which allows businesses to scale up build workloads with incrementally smaller storage footprint growth and to optimize compute and network resources for parallel builds. Other NetApp storage efficiency features like De-duplication, Data Compaction, and Data Compression reduce storage costs even further. The result is an improved Return of Investment (ROI) by doing “more with less” in the entire build process.

2 Lab Objectives

In this lab you will operate a Continuous Integration Workflow using Jenkins running with NetApp storage. In addition to Jenkins and NetApp products, the CI workflow also leverages Gitlab, Artifactory, and Docker.

In this lab you will perform the following activities.

1. Check containers for SCM, CI, Jenkins Master.
2. Check Jenkins Master Context for Jenkins Home files.
    - Check the volume on storage.
3. Check Gitlab SCM Context and NFS mounts.
    - Check Gitlab on browser for source code.
    - Check the volume on the storage.
    - Check snapshot from Jenkins UI.
4. Check CI container context.
    - Check Source Code and check the location of source files to be changed.
5. Create a new Developer workspace from Jenkins UI.
    - Check docker container context of the workspace.
    - Check the code and permissions of files and directories.
    - Check the volumes on storage.
    - Make a change to the source code.
    - Commit the changes.
    - Run maven tests.
    - Push the changes to SCM.
6. Check Gitlab on browser for the pushed change.
    - Check snapshot from the Jenkins UI.
    - Verify Snapshot from storage.
7. Check Automatic Build on CI Volume.
    - Check Build Snapshots from Jenkins UI.
    - Verify Snapshots from Storage.
8. Start the Build Artifact Management Pipeline from Jenkins UI.
    - Check docker container context at /tmp/vol1.
    - Check time stamped zip file.

3 Prerequisites

This lab assumes that you are familiar with the Docker fundamentals introduced in the “Introduction to Containers” lab. The Containers lab also explores NetApp's Docker integration, which is conceptually helpful for this lab, but that knowledge is not critical to completing this lab.

This lab also assumes that you know how to use PuTTY, and how to launch and log in to System Manager to manage a cluster. If you are unfamiliar with any of those procedures then please review *Appendix 1* of this lab guide.

4 Lab Environment

The following diagram illustrates the environment for this lab.

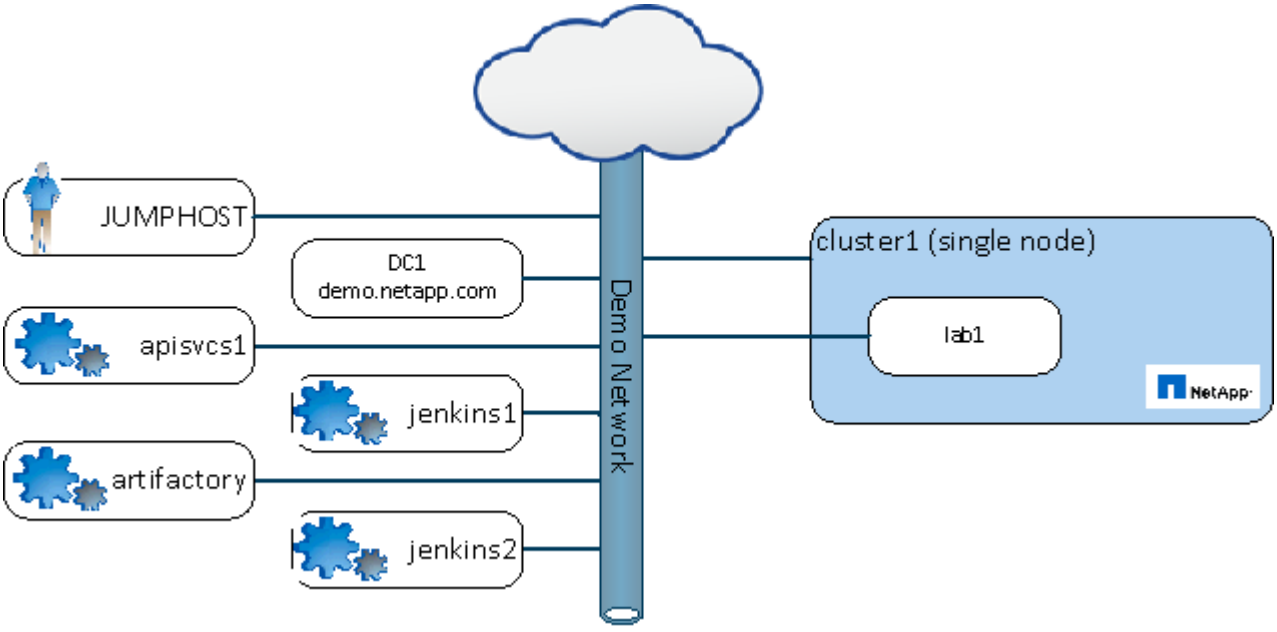


Figure 4-1:

All of the servers and storage controllers presented in this lab are virtual devices, and the networks that interconnect them are exclusive to your lab session. While we encourage you to follow the demonstration steps outlined in this lab guide, you are free to deviate from this guide and experiment with other ONTAP features that interest you. While the virtual storage controllers (vsims) used in this lab offer nearly all of the same functionality as physical storage controllers, they are not capable of providing the same performance as a physical controller, which is why these labs are not suitable for performance testing.

The *Lab Host Credentials* table provides a list of the servers and storage controller nodes in the lab, along with their IP address.

Table 1: Lab Host Credentials

Hostname	Description	IP Address(es)	Username	Password
apisvcs1	RHEL 7.3 x64 NetApp API Services	192.168.0.65	root	Netapp1!
artifactory	RHEL7.3 x64 Docker/Artifactory host	192.168.0.63	root	Netapp1!
dc1	Active Directory Server	192.168.0.253	Demo \Administrator	Netapp1!
jumphost	Windows 20012R2 Remote Access host	192.168.0.5	Demo \Administrator	Netapp1!
jenkins1	RHEL 7.3 x64 Docker/Jenkins host	192.168.0.61	root	Netapp1!
jenkins2	RHEL 7.3 x64 Docker/Jenkins host	192.168.0.62	root	Netapp1!
cluster1	ONTAP cluster	192.168.0.101	admin	Netapp1!

The *Preinstalled NetApp Software* table lists the NetApp software that is pre-installed on the various hosts in this lab.

Table 2: Preinstalled NetApp Software

Hostname	Description
jumphost	Data ONTAP DSM v4.1 for Windows MPIO, Windows Unified Host Utility Kit v7.0.0, NetApp PowerShell Toolkit v4.3.0
jenkins1	NetApp-Jenkins Framework, NetApp Docker Volume Plugin
jenkins2	NetApp Docker Volume Plugin
apisvcs	NetApp Service Level Manager v1.0

5 Lab Activities

In this lab you will explore how the NetApp-Jenkins framework can help you implement a more efficient Continuous Integration workflow. Specifically, you will:

- Examine the basic components of the Continuous Integration pipeline.
- Configure a Continuous Integration environment.
- Create a new developer workspace.
- Check the automatic build on a Continuous Integration volume.

This lab leverages a number of different NetApp and 3rd party products to create the CI environment, as follows:

NetApp-Jenkins framework: NetApp has developed a framework that integrates the Continuous Integration pipeline using Jenkins and ONTAP technologies, which enables businesses to accelerate the development process and time to market. The framework is a collection of python scripts that uses Docker, NetApp Docker Volume framework, and NetApp Service Level Manager (AKA ONTAP APIs) . The framework encapsulates 4 major modules in the CI pipeline – Source Code Management (SCM), CI environment, binary artifact manager, and User workspaces. The Jenkins master Docker image automatically install the Jenkins slave and Gitlab and configures the rest of the CI workflow on demand. The ONTAP APIs are used by the framework to provision the persistent data storage (FlexVols) for the Docker containers that can load balance and scale across storage nodes in a cluster. The framework uses nDVP is to mount the persistent data storage on a Docker container. All storage layer interactions are transparent, and provide a zero-touch to the developer. Developers do not need storage expertise to understand and manage the continuous integration workflow because the NetApp-Jenkins framework automates the entire CI pipeline with Jenkins. These scripts can be customized to support any customer environment.

GitLab: GitLab is a Source Code Management (SCM) system, also known as a version control system, and manages the different code changes in the source code repository. Git is one of the most common, popular, and highly distributed open source code management tools. Different flavors of Git include Gitlab, Git BitBucket, and Helix4Git.

Jenkins: Jenkins is a commonly used Open source Continuous Integration (CI) tool. This tool allows developers to build and test code to identify and resolve bugs quickly in an automated manner. Jenkins follows a distributed architecture with a master and slave configuration, where the Jenkins master is responsible for scheduling, managing, dispatching and monitoring build jobs. Each of these jobs represents a slave, and the slaves run different jobs (as requested by the master) in a distributed manner.

Artifactory: Artifactory is a build artifact repository. Artifacts are the binary files that software development projects depend upon and/or produce. Examples of artifacts include libraries, compilers, tools you use during development, and the product executables and packages (JAR, rpm, ISO, etc.) that you produce for product delivery and installation.

Docker: Jenkins master and slaves run as Docker services in a Docker Swarm multi-host cluster. A Docker service is a combination of a running Docker container image and the commands and/or application you wish to run inside that container instance to provide a specific application service. In Docker environments, solution providers commonly deploy applications as microservices, which are small modular services designed to work together and that can easily scale through the provisioning of additional container instances. In this lab the Jenkins master spawns off slaves for Gitlab, CI and developer workspace environments

NetApp Service Level Manager (NSLM) 1.0 : NLSM provides the ONTAP RESTful API integration that Jenkins leverages to create volumes (FlexVols), take snapshots, and provision file and volume clones (FlexClones) on NetApp storage. These ONTAP APIs automatically enable load balancing and scalabity of the FlexVols and FlexClones in a cluster namespace based on a controller headroom.

NetApp Docker Volume framework (nDVP): nDVP mounts the data volumes created by the ONTAP APIs on the Docker containers over NFS. nDVP allows the containers to mount the data volumes from ONTAP in a persistent manner. The “home directory” of the Jenkins master on ONTAP over NFS provides the resiliency and scalability of the Jenkins master and slave Docker containers.

5.1 Start the Lab Containers

The Docker containers that provide the Jenkins, Gitlab, and Artifactory services are not running when the lab starts. In this exercise you will start those containers.

1. On the taskbar of jumphost, launch **PuTTY**.

Figure 5-1:

The PuTTY Configuration window opens.

2. If the right pane does not contain the header “Basic options for your PuTTY session”, then in the left pane select the **Session** category.
3. In the right pane, in the “Saved Sessions” list, double-click **jenkins1**.

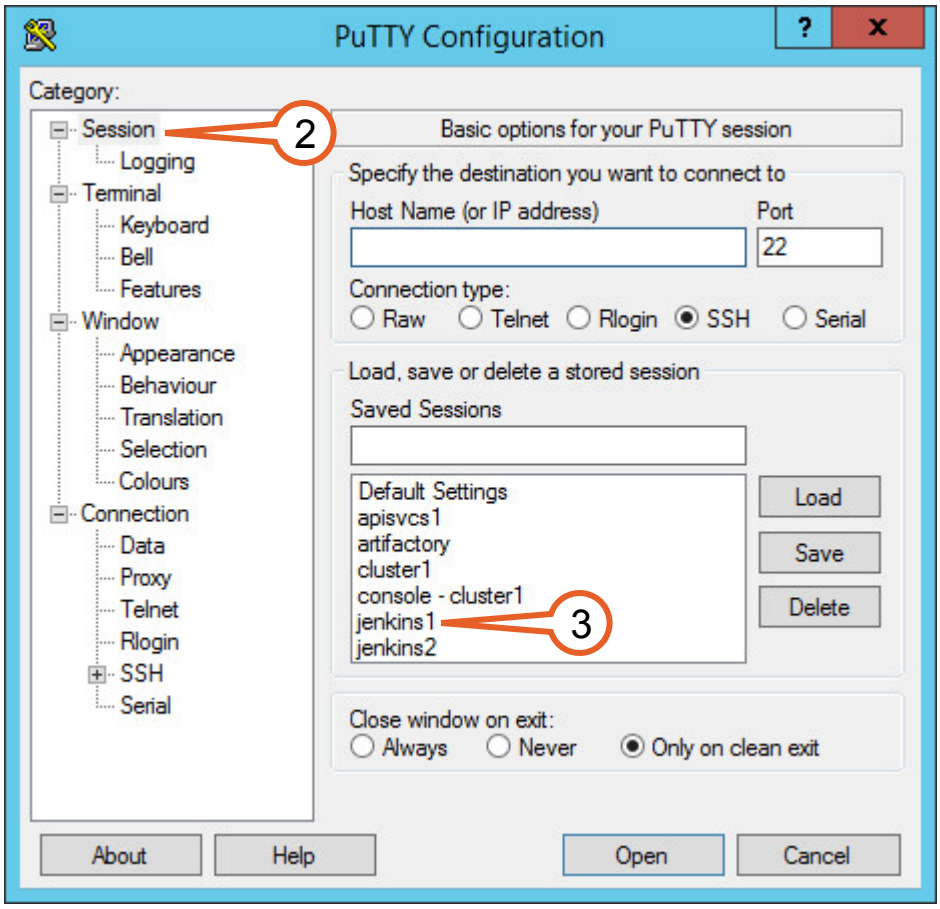


Figure 5-2:

A PuTTY teminal session window opens.

4. Log in using the username **`root`** and the password **`Netapp1!`**.

```
login as: root
root@jenkins1.demo.netapp.com's password:
Last login: Sat Jul 22 17:59:20 2017
[root@jenkins1 ~]#
```

5. View the list of nodes that comprise the Docker Swarm cluster.

```
[root@jenkins1 ~]# docker node ls
ID                           HOSTNAME  STATUS  AVAILABILITY  MANAGER STATUS
8lz1myn61kc7nt4kqfqrzj2ty    jenkins2  Ready   Active
s5f0prriswmjaime52nquom2q *  jenkins1  Ready   Active        Leader
[root@jenkins1 ~]#
```

The output shows that this swarm cluster contains two nodes, which is a minimal swarm environment, just large enough to provide high availability for the Jenkin master in case of a node failure. Swarm clusters typically have more than two nodes, but two is more than sufficient for the activities contained in this lab.

6. Display a list of the running containers.

```
[root@jenkins1 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
      PORTS               NAMES
[root@jenkins1 ~]#
```

There are no containers running yet.

7. Start all the docker containers in the CI pipeline containers. In this exercise these containers will all start on the jenkins1 node.

```
[root@jenkins1 ~]# docker start $(docker ps -q -f status=exited)
22edf3dd1cdc
[root@jenkins1 ~]#
```

> **Note:** The portion of this command after “docker start” launches a subshell that queries for containers that have an exited status, meaning the returned list of containers were shut down gracefully and are not currently running. The “docker start” portion of the command will then start the list of returned containers.

8. Display the status of the docker containers running on the jenkins1 node. The output should eventually show two running containers, with the JFrog_OSS_Repo showing as “(healthy)”. It takes about a minute for the containers to settle into this state, so monitor them by issuing the `docker ps` command every 10 seconds or so. Do not proceed to the next step until the JFrog_OSS_Repo container reports as healthy.

```
[root@jenkins1 ~]# docker ps
CONTAINER ID        IMAGE                                                  COMMAND
 CREATED             STATUS                                 PORTS
                    NAMES
3b9fe3ba4585        devopsnetapponaws/netapp-jenkins-plugin-2.0:lod   "/bin/tini -- /usr..."
 5 seconds ago       Up 3 seconds                           8080/tcp, 50000/tcp
                     jenkins.1.jdf6vqyf2k1ggenjjamju1oqo
22edf3dd1cdc        devopsnetapponaws/netapp-jenkins_gitlab            "/usr/bin/startcon..."
  2 months ago        Up 22 seconds (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
[root@jenkins1 ~]# docker ps
CONTAINER ID        IMAGE                                                  COMMAND
 CREATED             STATUS                          PORTS
                       NAMES
3b9fe3ba4585        devopsnetapponaws/netapp-jenkins-plugin-2.0:lod   "/bin/tini -- /usr..."
 54 seconds ago      Up 51 seconds                   8080/tcp, 50000/tcp
                       jenkins.1.jdf6vqyf2k1ggenjjamju1oqo
22edf3dd1cdc        devopsnetapponaws/netapp-jenkins_gitlab            "/usr/bin/startcon..."
  2 months ago        Up About a minute (healthy)   0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp,
 0.0.0.0:10022->22/tcp   JFrog_OSS_Repo
[root@jenkins1 ~]#
```

9. On the taskbar of Jumphost, launch the **Chrome** browser.

Figure 5-3:

Chrome will open and display several tabs by default.

10. Select the **Pipelines [Jenkins]** tab, which is the tab for the Jenkins UI. This page may take a minute or so to load.

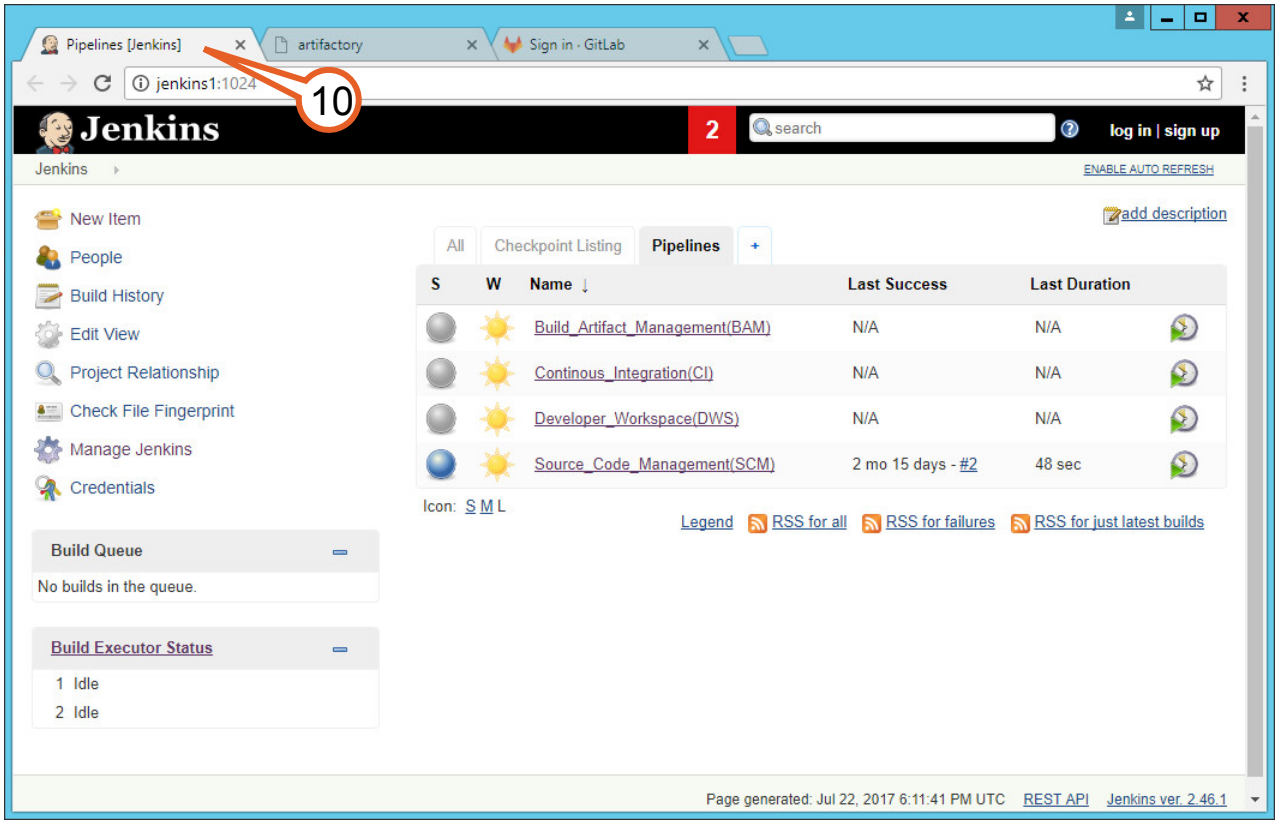


Figure 5-4:

If the page loads successfully then the Jenkins master is running properly.

> **Note:** If you do not see the list of pipelines on the right side of your page then the browser window is likely not wide enough, try widening it.
>
> **Note:** Ignore the “log in” link at the top of the Jenkins UI page. In order to streamline the lab experience, we have disabled the requirement to authenticate in Jenkins. You do not need to log in to Jenkins while running this lab.

Leave your browser and the PuTTY session to rhel1 running. You will need to use them again later.

11. On the taskbar on Jumphost, right click on the **PuTTY** icon.
12. Select **PuTTY** from the context menu.

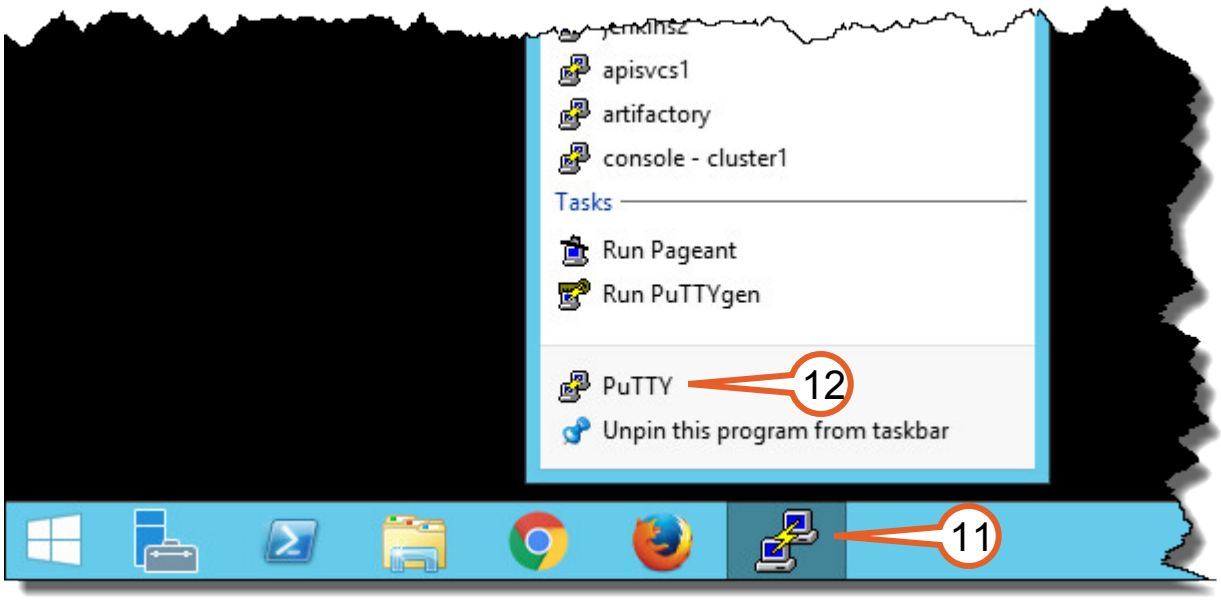


Figure 5-5:

13. In the saved sessions list, double click on **artifactory**, and log in as the user **`root`** with the password **`Netapp1!`**.

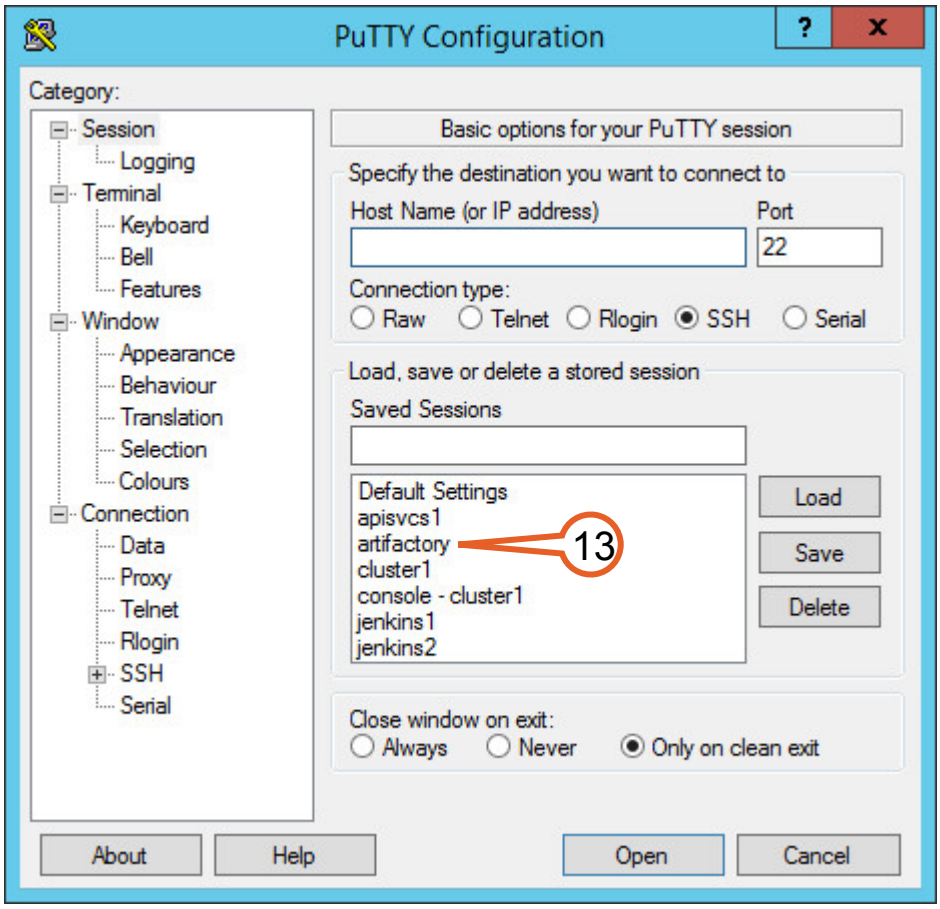


Figure 5-6:

14. Display a list of the running containers.

```
[root@artifactory ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
       PORTS               NAMES
[root@artifactory ~]#
```

No containers are currently running.

15. Start the artifactory-oss container.

```
[root@artifactory ~]# docker start artifactory-oss
artifactory-oss
[root@artifactory ~]#
```

16. Verify the container is running.

```
[root@artifactory ~]# docker ps
CONTAINER ID        IMAGE                                                COMMAND
 CREATED             STATUS              PORTS                     NAMES
4805e605caac        docker.bintray.io/jfrog/artifactory-oss:latest   "/bin/sh -c /entry..."
 5 days ago          Up 5 seconds        0.0.0.0:8081->8081/tcp    artifactory-oss
[root@artifactory ~]#
```

17. In Chrome, select the **artifactory** tab, and refresh the page to verify that you can access the Artifactory user interface.

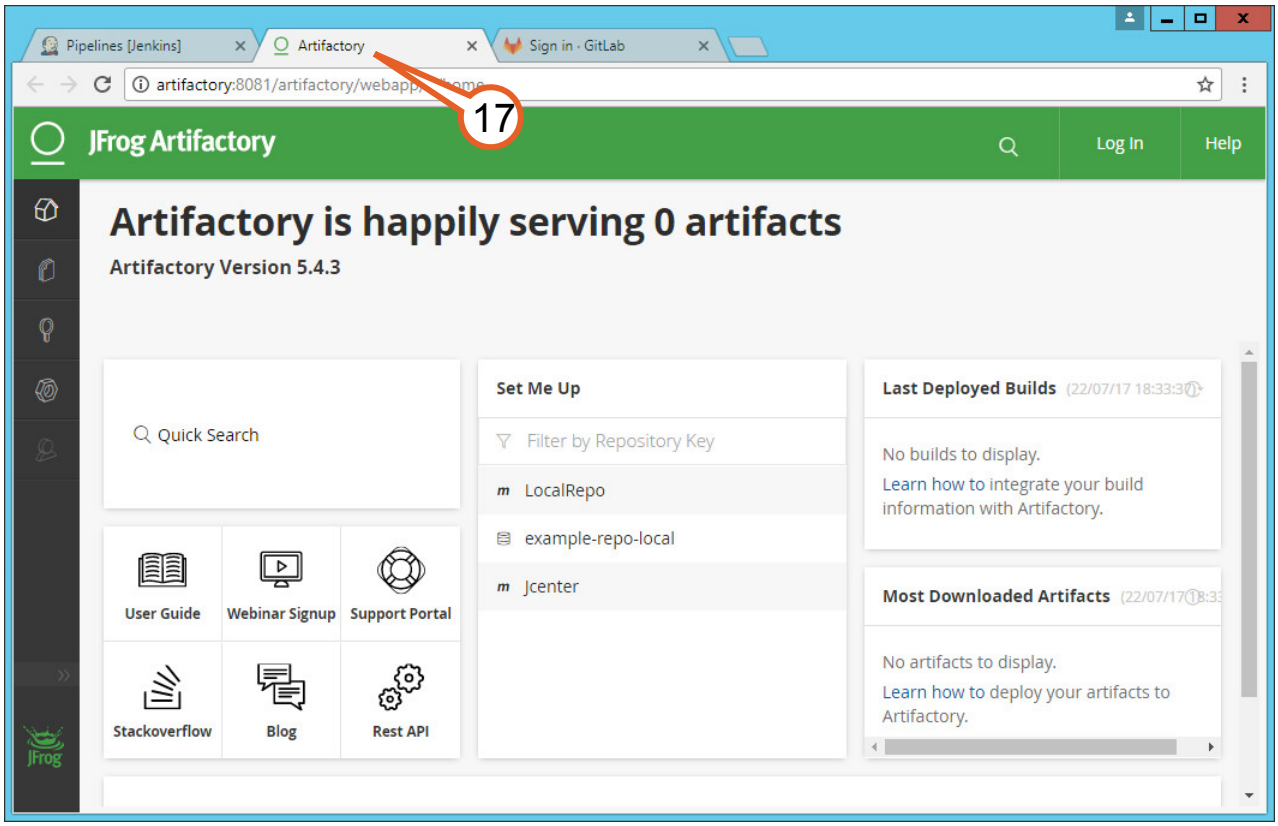


Figure 5-7:

18. Leave your browser and PuTTY sessions open, as you will be using them again later.

Note: If the jenkins1 or artifactory VMs restart for any reason (for example, a crash or reboot), you will need to repeat these steps to manually start these containers.

5.2 Review the Lab Environment

In this exercise you take a closer look at the infrastructure components to see how they are utilizing NetApp storage.

1. In your PuTTY session to jenkins1, identify the container that is serving as the jenkins master. The Jenkins master is responsible for deploying and managing the build jobs that run in slave containers.

```
[root@jenkins1 ~]# docker ps
CONTAINER ID        IMAGE                                                COMMAND
 CREATED             STATUS                PORTS
              NAMES
3b9fe3ba4585        devopsnetapponaws/netapp-jenkins-plugin-2.0:lod   "/bin/tini -- /usr..."
 2 hours ago         Up 2 hours            8080/tcp, 50000/tcp
              jenkins.1.jdf6vqyf2k1ggenjjamju1oqo
22edf3dd1cdc        devopsnetapponaws/netapp-jenkins_gitlab           "/usr/bin/startcon..."
   2 months ago        Up 2 hours (healthy)   0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp,
 0.0.0.0:10022->22/tcp   JFrog_OSS_Repo
[root@jenkins1 ~]#
```

   Note: You can identify the container for the jenkins master as the one whose “Name” value matches the pattern “jenkins.1.<long string of random characters>”. The container's full name in your lab will likely be different than what is shown in this example.

2. Open a bash session inside the Jenkins master container. Replace the container name value in this command with the value from your lab by using copy/paste.

```
[root@jenkins1 ~]# docker exec -it jenkins.1.jdf6vqyf2k1ggenjjamju1oqo /bin/bash
root@3b9fe3ba4585:/#
```

3. Display the list of mounted volumes.

```
root@3b9fe3ba4585:/# df
Filesystem
     1K-blocks    Used Available Use% Mounted on
/dev/mapper/docker-253:0-68215161-
e50f36baf8a837c8da0462c1a885d92233e0dd43ba431e59a9c885b8a6adea36  10474496 1187880   9286616
 12% /
tmpfs
        941996       0    941996   0% /dev
tmpfs
        941996       0    941996   0% /sys/fs/cgroup
192.168.0.132:/jenkins_home
       9961472  270144   9691328   3% /var/jenkins_home
/dev/mapper/rhel-root
      37202180 8887204  28314976  24% /etc/hosts
shm
         65536       0     65536   0% /dev/shm
tmpfs
        941996    8912    933084   1% /run/docker.sock
tmpfs
        941996       0    941996   0% /sys/firmware
root@3b9fe3ba4585:/#
```

   The /var/jenkins_home directory is NFS mounted from the NetApp storage controller. This volume, like all the NFS mounted volumes in this lab, is a persistent storage volume for the container, meaning the container can be destroyed and redeployed without any impact to the contents of this volume. Because these persistent volumes are hosted on NetApp, the volume contents are protected with NetApp Snapshots, allowing users to quickly self-restore files without the need for storage administrator involvement. In addition, these volumes can take advantage of NetApp storage efficiency features like volume thin provisioning and deduplication. They can potentially also leverage NetApp data replication and protection features like SnapMirror and SnapVault (if the customer has purchased these products).

4. Change directory to the jenkins_home directory, and display the directory's contents.

```
root@3b9fe3ba4585:/# cd /var/jenkins_home
root@3b9fe3ba4585:/var/jenkins_home# ls -l
total 200
-rw-r--r--  1 root root  4544 Jul 17 06:13 config.xml
-rw-r--r--  1 root root  4237 Jul 22 18:01 copy_reference_file.log
```

```
drwxr-xr-x  2 root root  4096 Jul 17 10:37 fingerprints
-rw-r--r--  1 root root   228 May  8 07:13 hudson.ivy.IvyBuildTrigger.xml
-rw-r--r--  1 root root   101 May  8 07:13 hudson.ivy.IvyModuleSet.xml
-rw-r--r--  1 root root   810 May  8 07:13 hudson.maven.MavenModuleSet.xml
-rw-r--r--  1 root root   159 Jul 22 18:02 hudson.model.UpdateCenter.xml
-rw-r--r--  1 root root   320 May  8 07:13 hudson.plugins.git.GitSCM.xml
-rw-r--r--  1 root root   370 May  8 07:11 hudson.plugins.git.GitTool.xml
-rw-r--r--  1 root root   173 May  8 07:11 hudson.plugins.gradle.Gradle.xml
-rw-r--r--  1 root root   145 May  8 07:11 hudson.tasks.Ant.xml
-rw-r--r--  1 root root   188 May  8 07:13 hudson.tasks.Mailer.xml
-rw-r--r--  1 root root   320 May  8 07:11 hudson.tasks.Maven.xml
-rw-r--r--  1 root root    76 May  8 07:13 hudson.tasks.Shell.xml
-rw-r--r--  1 root root   215 May  8 07:13 hudson.triggers.SCMTrigger.xml
-rw-------  1 root root  1712 May  8 06:51 identity.key.enc
drwxr-xr-x  2 root root  4096 May  8 06:51 init.groovy.d
-rw-r--r--  1 root root     6 Jul 22 18:02 jenkins.install.InstallUtil.lastExecVersion
-rw-r--r--  1 root root     4 May  8 06:51 jenkins.install.UpgradeWizard.state
-rw-r--r--  1 root root   159 May  8 07:13 jenkins.model.ArtifactManagerConfiguration.xml
-rw-r--r--  1 root root   138 May 11 04:57 jenkins.model.DownloadSettings.xml
-rw-r--r--  1 root root   259 May  8 07:13 jenkins.model.JenkinsLocationConfiguration.xml
-rw-r--r--  1 root root   247 May  8 07:11 jenkins.mvn.GlobalMavenConfig.xml
-rw-r--r--  1 root root   169 May 11 04:57
 jenkins.security.QueueItemAuthenticatorConfiguration.xml
-rw-r--r--  1 root root   240 May 11 04:57
 jenkins.security.UpdateSiteWarningsConfiguration.xml
drwxr-xr-x 16 root root  4096 Jul 18 12:32 jobs
drwxr-xr-x  4 root root  4096 May  8 09:02 logs
-rw-r--r--  1 root root   901 Jul 22 18:02 nodeMonitors.xml
drwxr-xr-x  2 root root  4096 May  8 06:52 nodes
-rw-r--r--  1 root root   418 May  8 07:13
 org.jenkinsci.plugins.conditionalbuildstep.singlestep.SingleConditionalBuilder.xml
-rw-r--r--  1 root root   298 May  8 07:11
 org.jenkinsci.plugins.docker.commons.tools.DockerTool.xml
-rw-r--r--  1 root root   255 May  8 07:11 org.jenkinsci.plugins.gitclient.JGitApacheTool.xml
-rw-r--r--  1 root root   243 May  8 07:11 org.jenkinsci.plugins.gitclient.JGitTool.xml
-rw-r--r--  1 root root   290 May  8 07:13
 org.jenkinsci.plugins.pipeline.modeldefinition.config.GlobalConfig.xml
-rw-r--r--  1 root root    46 May  8 09:31
 org.jenkinsci.plugins.workflow.flow.FlowExecutionList.xml
-rw-r--r--  1 root root   218 May  8 07:13
 org.jenkinsci.plugins.workflow.libs.GlobalLibraries.xml
-rw-r--r--  1 root root   600 May  8 07:13 org.jfrog.hudson.ArtifactoryBuilder.xml
drwxr-xr-x 64 root root 24576 May  8 06:51 plugins
-rw-r--r--  1 root root   130 Jul 18 18:21 queue.xml.bak
-rw-r--r--  1 root root   679 May  8 09:25 scriptApproval.xml
-rw-r--r--  1 root root    64 May  8 06:51 secret.key
-rw-r--r--  1 root root     0 May  8 06:51 secret.key.not-so-secret
drwx------  4 root root  4096 May  8 09:02 secrets
-rw-r--r--  1 root root    52 May  8 09:09 snaps2.properties
drwxr-xr-x  2 root root  4096 Jul 22 18:02 updates
drwxr-xr-x  2 root root  4096 May  8 06:52 userContent
drwxr-xr-x 10 root root  4096 May 12 05:10 war
drwxr-xr-x  2 root root  4096 May  8 06:52 workflow-libs
root@3b9fe3ba4585:/var/jenkins_home#
```

The /var/jenkins_home directory stores all the details of the Jenkins server configuration, much of it in the form of XML files.

5. Terminate the bash session to the Jenkins Master container.

```
root@3b9fe3ba4585:/var/jenkins_home# exit
exit
[root@jenkins1 ~]#
```

6. Display the list of mounted volumes inside the JFrog_OSS_Repo container, which is the container instance running the Gitlab SCM server. The SCM only hosts a project's source code files; no build activity is performed in this container..

```
[root@jenkins1 ~]# docker exec -it JFrog_OSS_Repo df
Filesystem
     1K-blocks    Used Available Use% Mounted on
/dev/mapper/docker-253:0-68215161-
c80c3ad8345635a7464b1c34d6b0fe9c5437a5e883a13356901d5fe79f709089  10474496 1967308   8507188
 19% /
tmpfs
        941996       0    941996   0% /dev
tmpfs
        941996       0    941996   0% /sys/fs/cgroup
```

```
192.168.0.132:/JFrog_OSS_Repo_config
       5412608     768   5411840   1% /etc/gitlab
/dev/mapper/rhel-root
      37202180 8887532  28314648  24% /etc/hosts
shm
         65536       4     65532   1% /dev/shm
192.168.0.132:/JFrog_OSS_Repo_logs
       5412608    5120   5407488   1% /var/log/gitlab
192.168.0.132:/JFrog_OSS_Repo
       2166208  481792   1684416  23% /var/opt/gitlab
tmpfs
        941996       0    941996   0% /sys/firmware
[root@jenkins1 ~]#
```

The NFS volumes mounted on /etc/gitlab and /var/log/gitlab host the configuration files for the Gitlab instance, and the NFS volume mounted on /var/opt/gitlab volume hosts the source code files that Gitlab is managing. These are once again persistent storage volumes that are hosted on the NetApp storage, and that can take advantage of NetApp data management features.

7. In your PuTTY session to artifactory, display the list of mounted volumes inside the artifactory-oss container, which hosts the Artifactory artifact repository.

```
[root@artifactory ~]# docker ps
CONTAINER ID        IMAGE                                          COMMAND
 CREATED             STATUS              PORTS                     NAMES
4805e605caac        docker.bintray.io/jfrog/artifactory-oss:latest   "/bin/sh -c /entry..."
 5 days ago          Up About an hour    0.0.0.0:8081->8081/tcp   artifactory-oss
[root@artifactory ~]# docker exec -it artifactory-oss df
Filesystem
     1K-blocks    Used Available Use% Mounted on
/dev/mapper/
docker-253:0-33746439-56e1114f1693cb32c49ed1c7526dcbde45e7a776a81853c3a3afebfcd4cac55c
 10474496  907924   9566572   9% /
tmpfs
       4005536       0   4005536   0% /dev
tmpfs
       4005536       0   4005536   0% /sys/fs/cgroup
/dev/mapper/rhel-root
      37202180 4878416  32323764  14% /etc/hosts
shm
         65536       0     65536   0% /dev/shm
192.168.0.132:/artifactory
        996160    5312    990848   1% /var/opt/jfrog/artifactory
tmpfs
       4005536       0   4005536   0% /sys/firmware
[root@artifactory ~]#
```

The NFS volume mounted on /var/opt/jfrog/artifactory hosts the artifacts that are managed by the Artifactory server.

8. Exit your PuTTY session to artifactory, as you will not need it again in this lab.

```
[root@artifactory ~]# exit
logout
```

5.3 Add Artifactory to Jenkins

In this exercise you will configure Jenkins to utilize Artifactory.

As mentioned in the introduction to the lab activites, Artifactory is a binary artifact manager, and artifacts are the binary files (libraries, compilers, tools, etc) that are required to build a software development project. They can also be the binaries produced by running a build (like product executables). Continuous Integration environments need a place to store these binaries, and that is the job of the binary artifact manager. In short, a binary artifact manager is much like a source code management system (such as GitLab), but rather than storing source code in the form of text files, the binary artifact manager stores the binary inputs and outputs required for and produced by software builds.

Gitlab and Artifactory communicate with Jenkins pipelines using different plugins. These plugins are already part of the NetApp-Jenkins framework that is deployed in this lab. The following steps configure Jenkins to discover the lab's Artifactory instance.

As you saw in a previous exercise, Artifactory is mounting a flexvol (persistent data store) from ONTAP, meaning you can leverage ONTAP data protection and storage efficiency features. For example, ONTAP De-duplication and compression often results in high degree of storage efficiency, as different versions of artifacts usually have a lot of blocks in common. This delivers a higher ROI since you do not have to buy as much storage to accomodate your artifact repository. Similarly, ONTAP Snapshots and SnapRestore provide faster recovery from data corruption and data loss.

1. In Chrome, select the **Pipelines [Jenkins]** tab.
2. In the left pane, click **Manage Jenkins**.

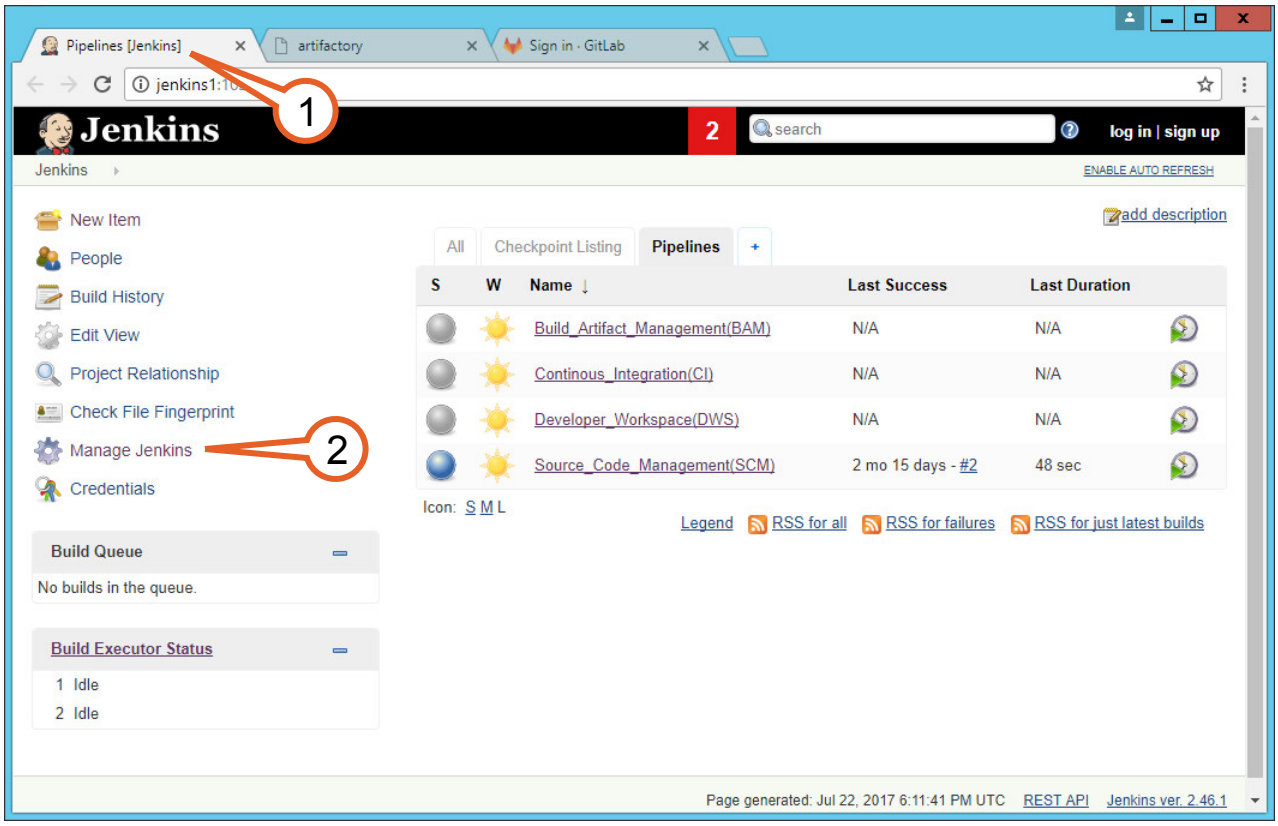


Figure 5-8:

The browser displays the “Manage Jenkins” page.

3. In the right pane click **Configure System**.

> **Note:** You may see a warning stating that a new version of Jenkins is available. Do not upgrade, because if you do then you may not be able to complete this lab!

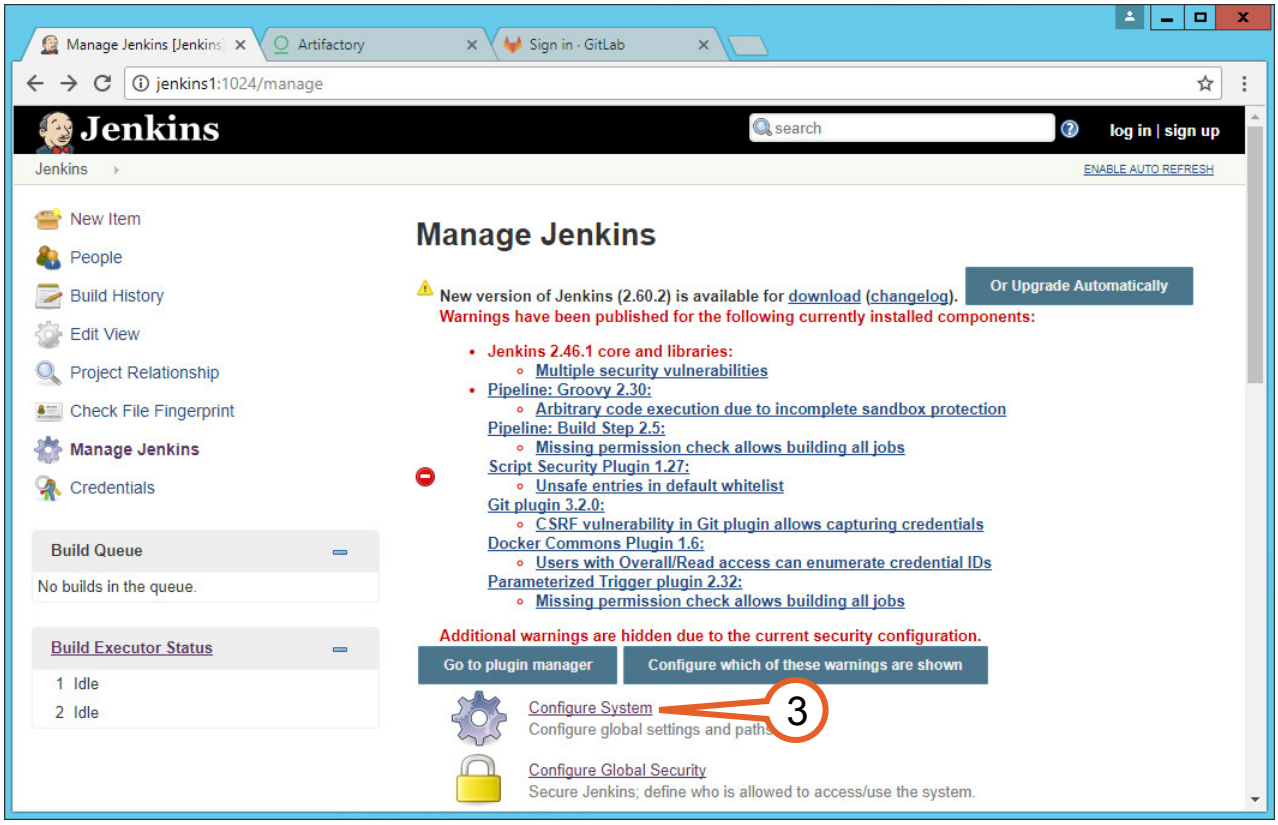


Figure 5-9:

The browser displays the Jenkins configuration page.

4. Scroll down in the browser window and locate the Artifactory section.
5. Click **Add Artifactory Server**.

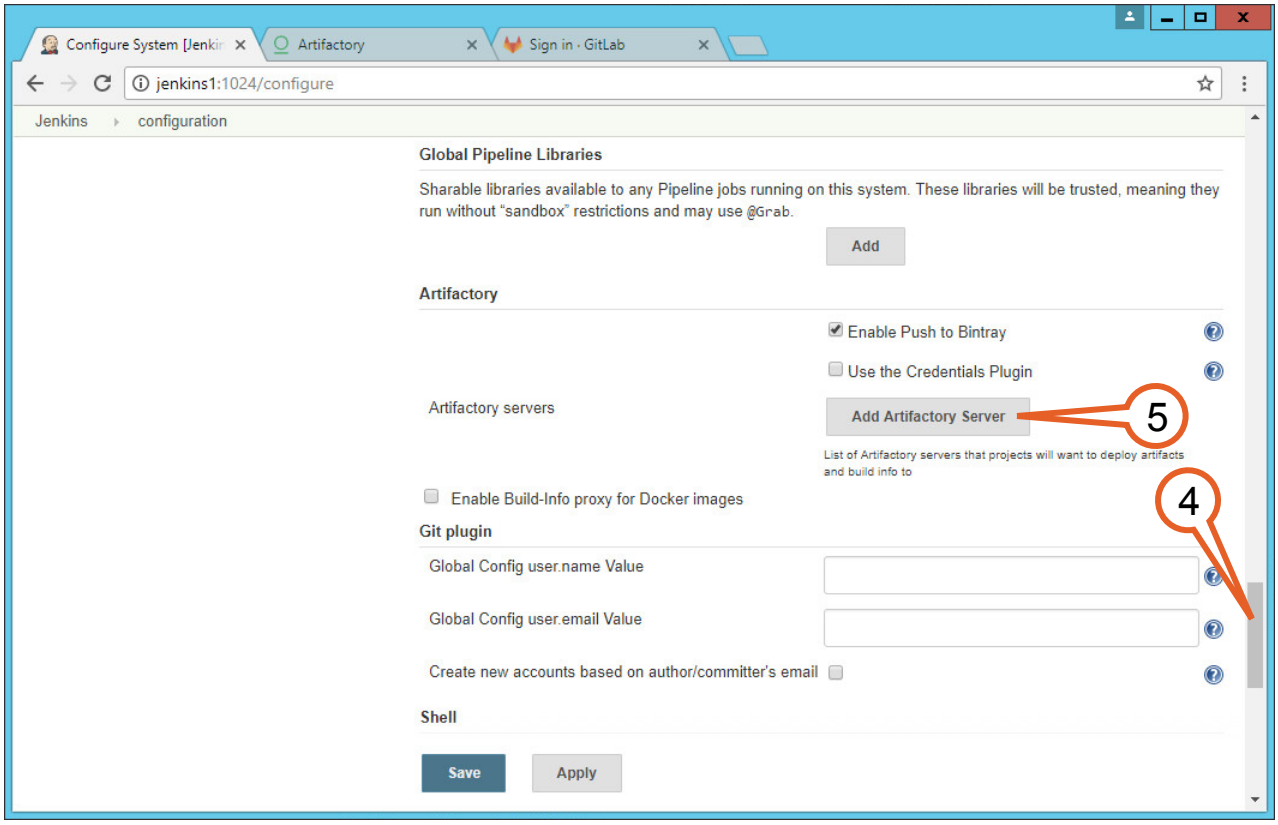


Figure 5-10:

Some new fields appear in the web page's Artifactory section.

6. Set the fields in the Artifactory section as follows:
    - Server ID: **`1`**
    - URL: **`http://artifactory:8081/artifactory`**

    Leave all the other fields at their default values.

    Note: The URL value you entered here (http://artifactory:8081/artifactory) is the same URL specified for the Artifactory tab in your Chrome browser.
7. Click **Test Connection**.

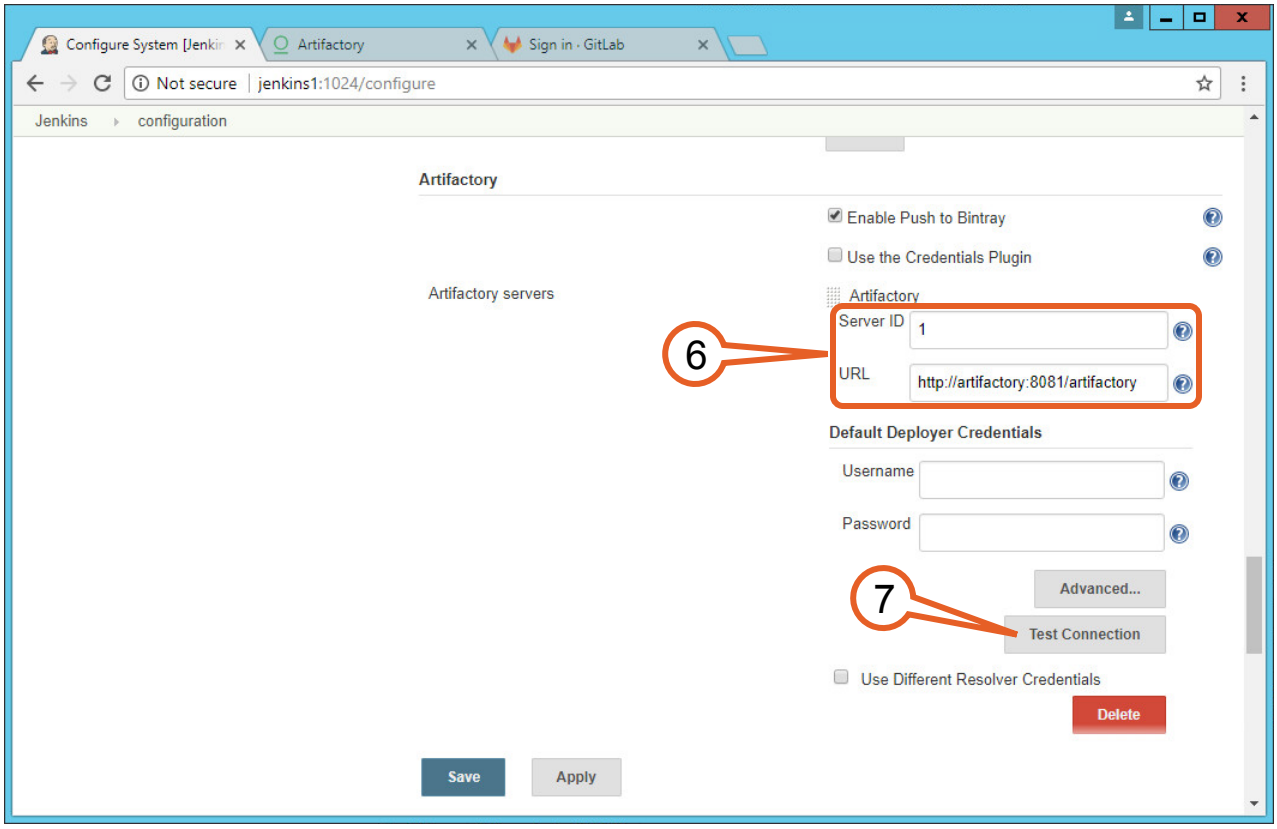


Figure 5-11:

8. The browser begins the test cycle, and after approximately 10 seconds you should see the text “Found Artifactory 5.4.3” display to the left of the **Test Connection** button.
9. Click **Save**.

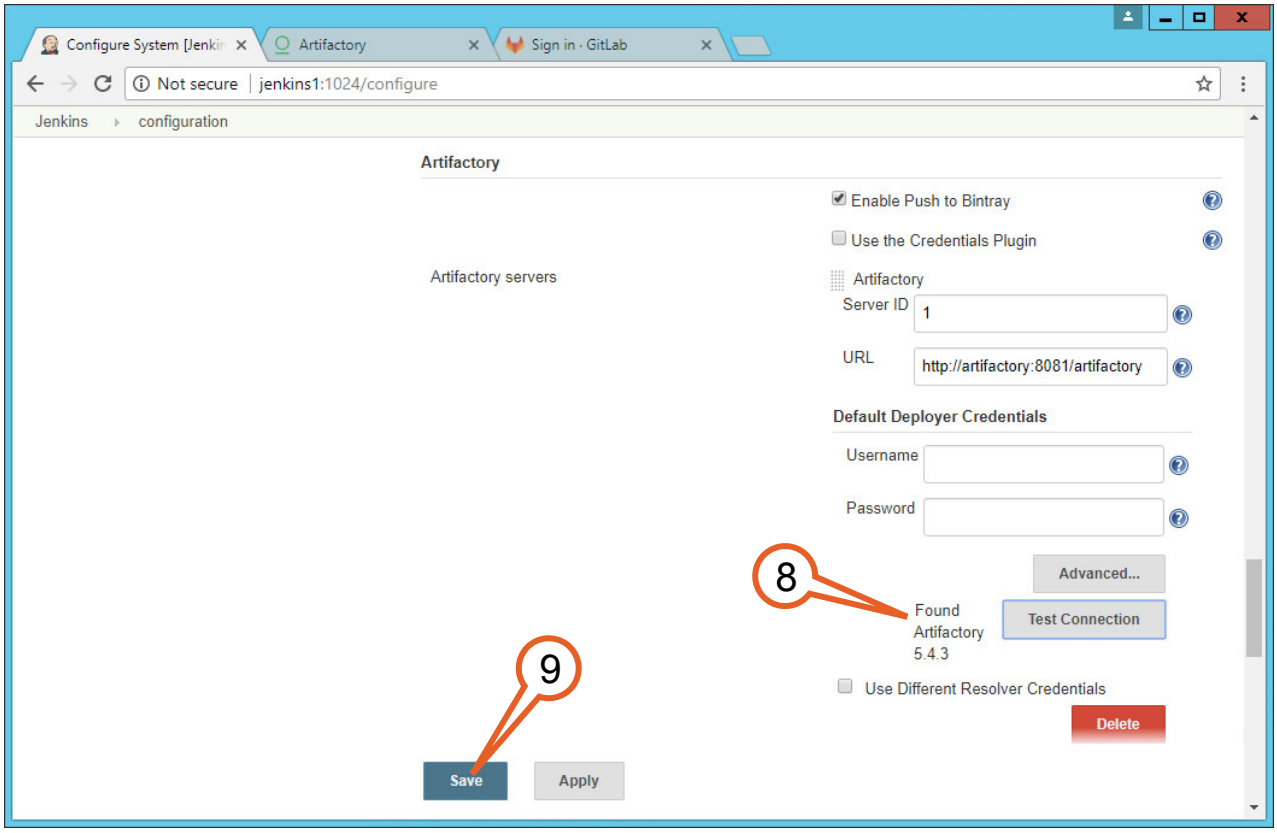
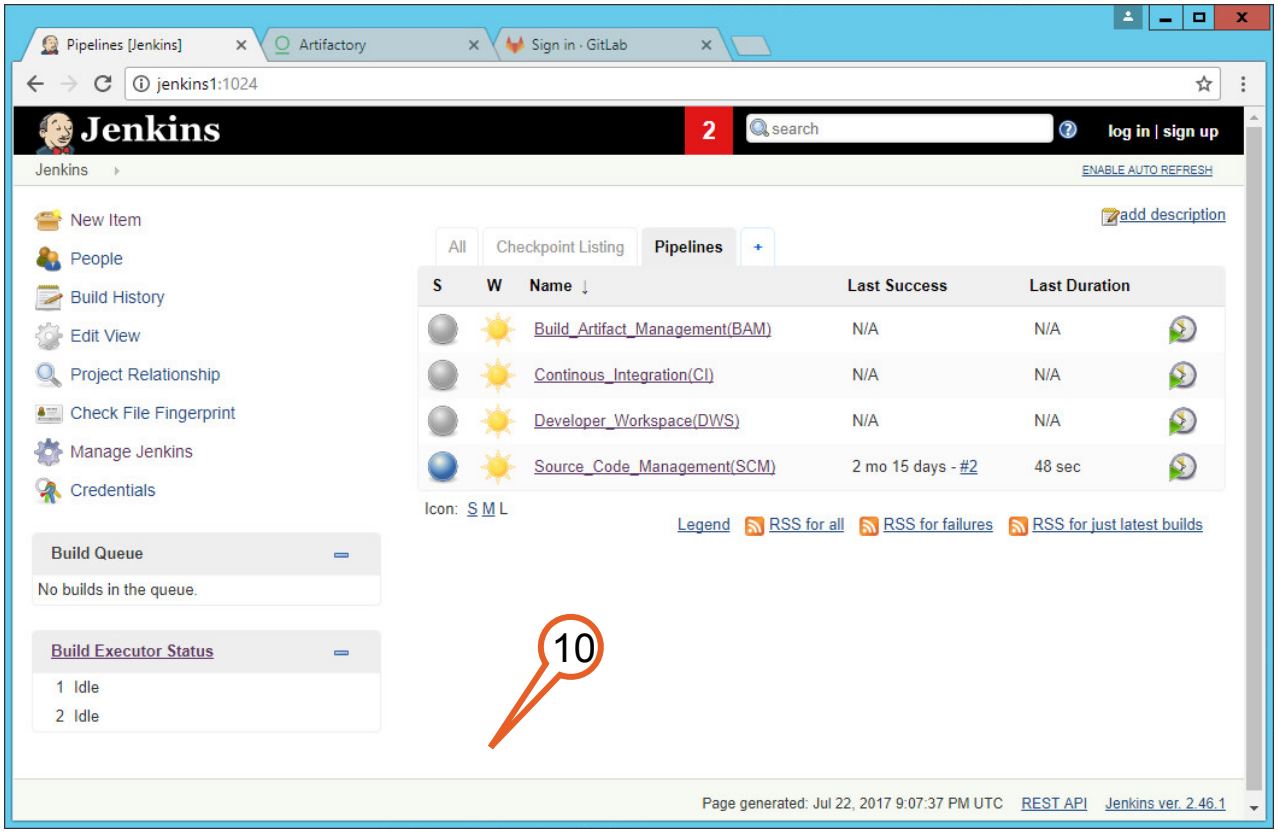


Figure 5-12:

10. When Jenkins finishes saving the configuration, the “Save” and “Apply” buttons disappear from the bottom of the browser window, and the browser URL reverts to display the “Pipelines [Jenkins]” page.

    Figure 5-13:



Jenkins is now configured to use Artifactory.

5.4 Create the Continuous Integration Environment

So far you have examined the infrastructure components that will support your Jenkins continuous integration project, and configured Jenkins to use Artifactory as its artifact repository, but you have not created created a continuous integration instance in which to run your master builds.

In the exercise you will use the Jenkins UI to create that instance.

1. Chrome should still be open to the “Pipelines [Jenkins]' tab (http://jenkins1:1024) from the preceding exercise. If not, re-open it.
2. In the right pane, click the **Continuous_Integration(CI)** link.

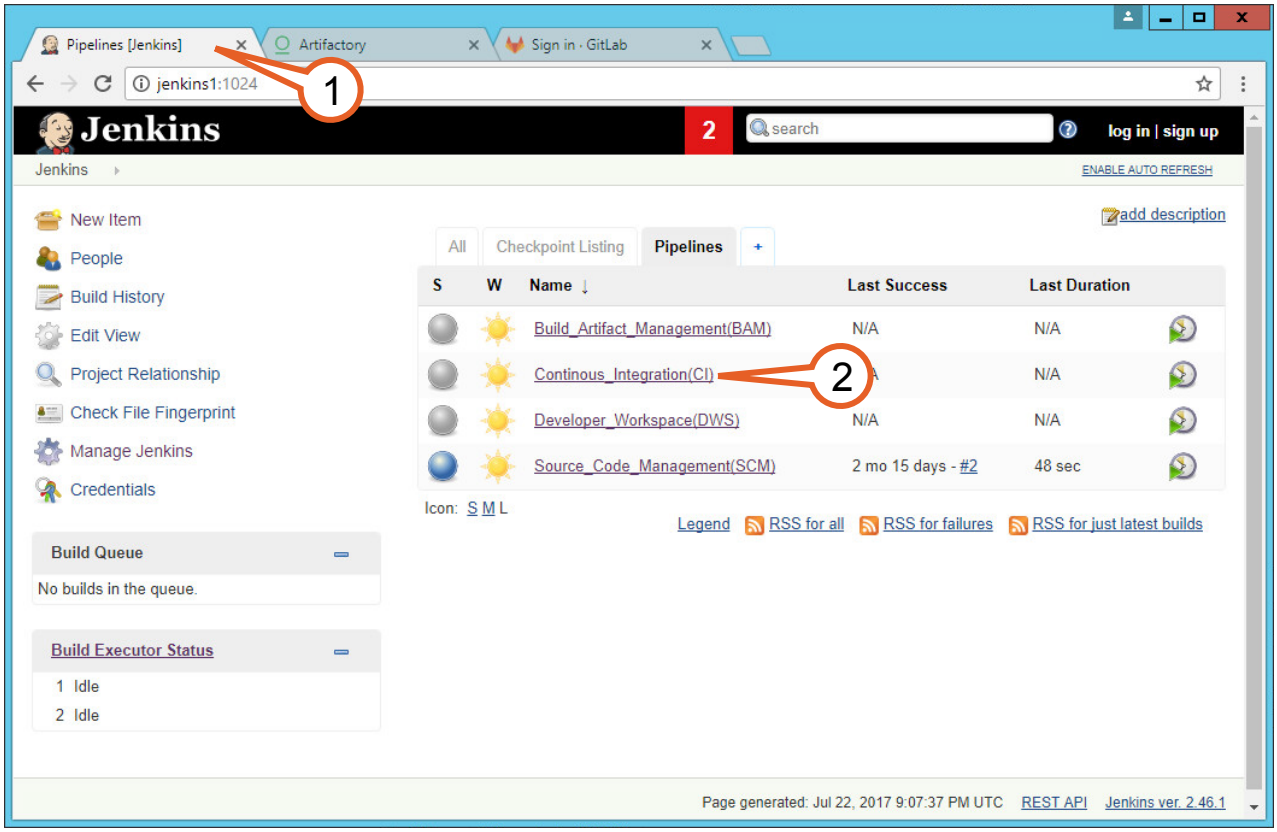


Figure 5-14:

The browser displays the “Pipeline Continuous_Integration(CI)” page.

3. In the left pane click **Build Now**.

> **Note:** Do not get confused by the term “build”, which in a software development context often means to compile source code. In Jenkins a “build” simply means to execute a workflow, which in this case is the pipeline to create a new continuous integration instance.

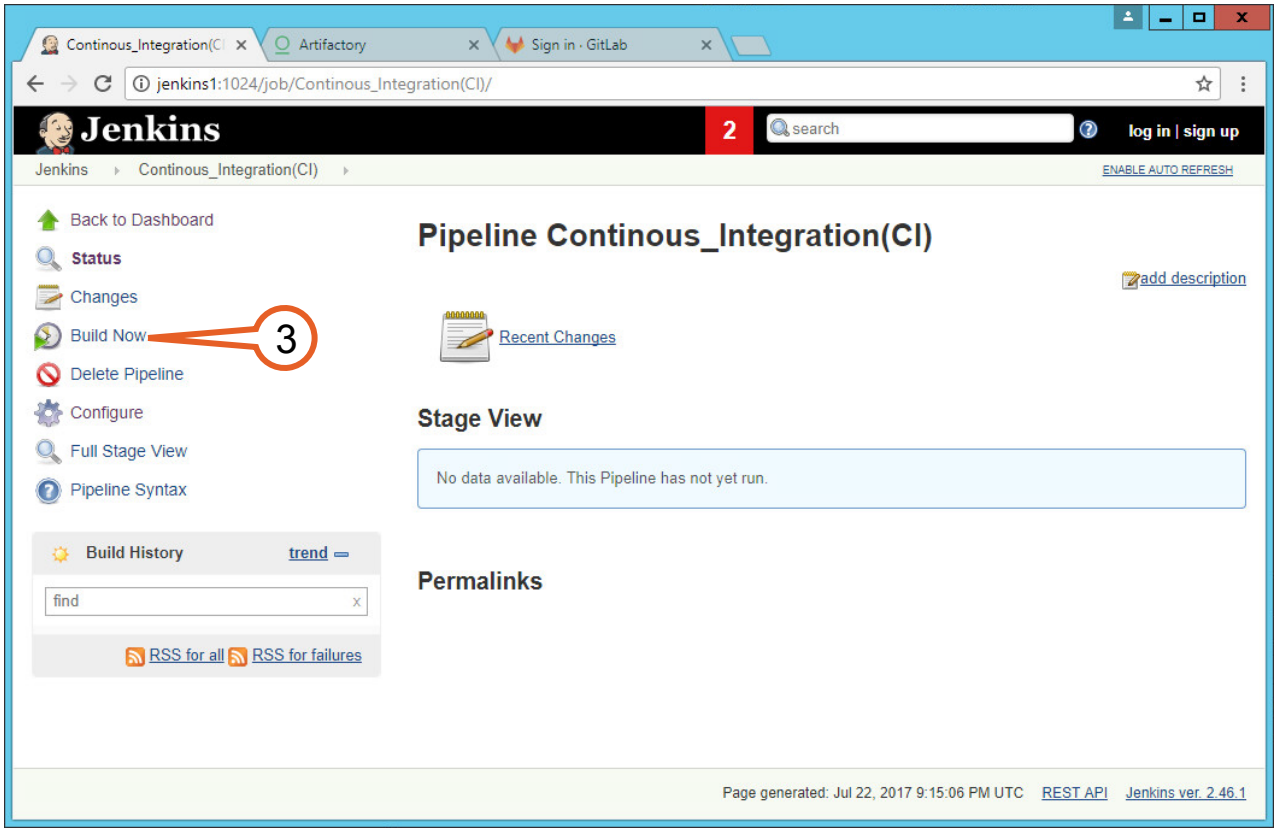


Figure 5-15:

4. A build job is added in the left pane under the “Build History” section.
5. A build pipeline graphic appears in the right pane under the “Stage View” section. Hover your mouse cursor over the pipeline stage (the blue rectangle, that will turn green once the pipeline completes). A “Logs” window pops up.

   Note: If you mouse out of the blue/green box the pop-up window will disappear.
6. Click on the **Logs** button in the window.

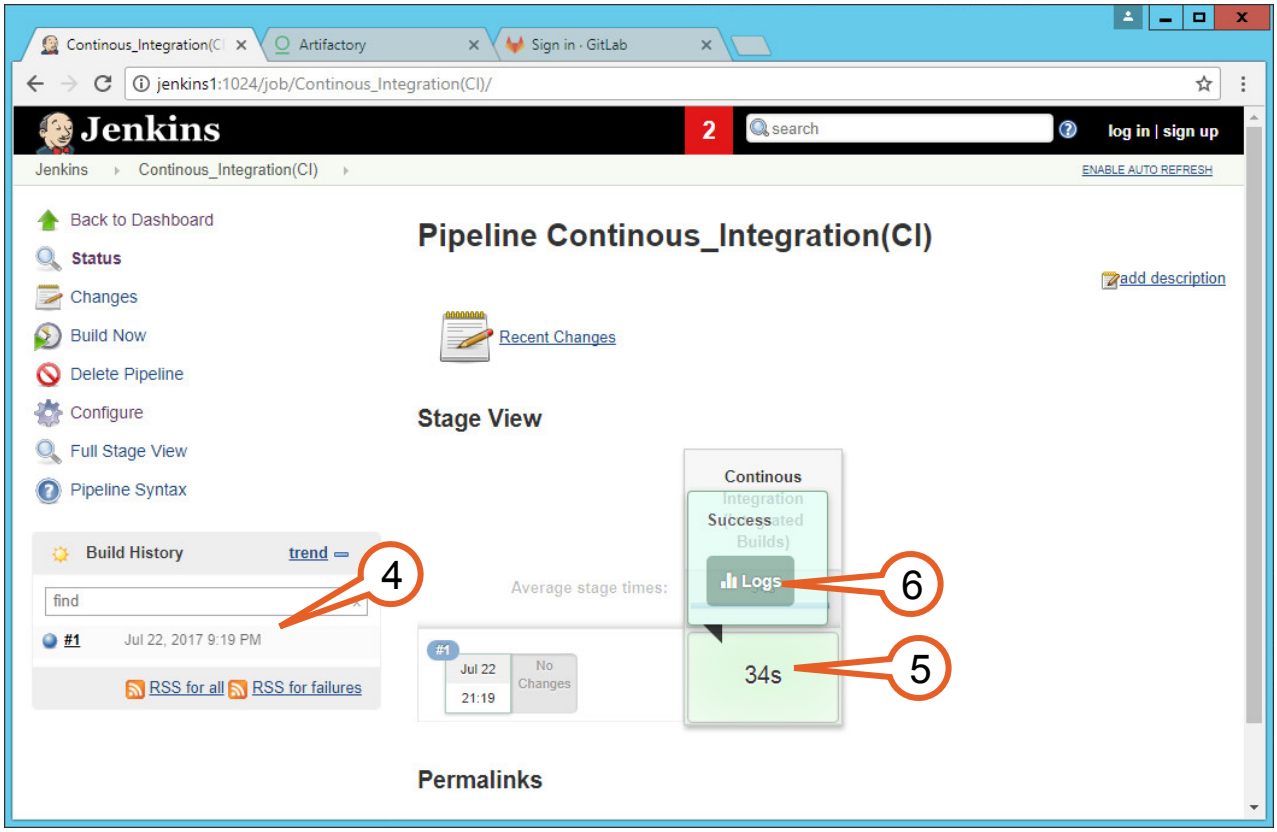


Figure 5-16:

A “Stage Logs (Continuous Integration (Integrated Builds))” window opens.

7. On the “Starting building:” line, click **JFrog_2017_1_#2**.

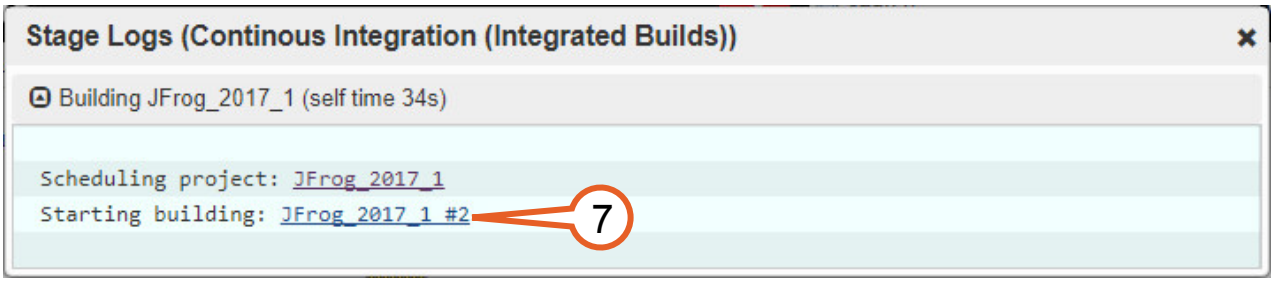


Figure 5-17:

The job stage log window opens.

8. If you are fast enough to get here while the pipeline is still running then you will see a progress bar near the top of this page, and the progress bar should continue to update in real time as the pipeline continues to execute.
9. In the left pane click **Console Output**.

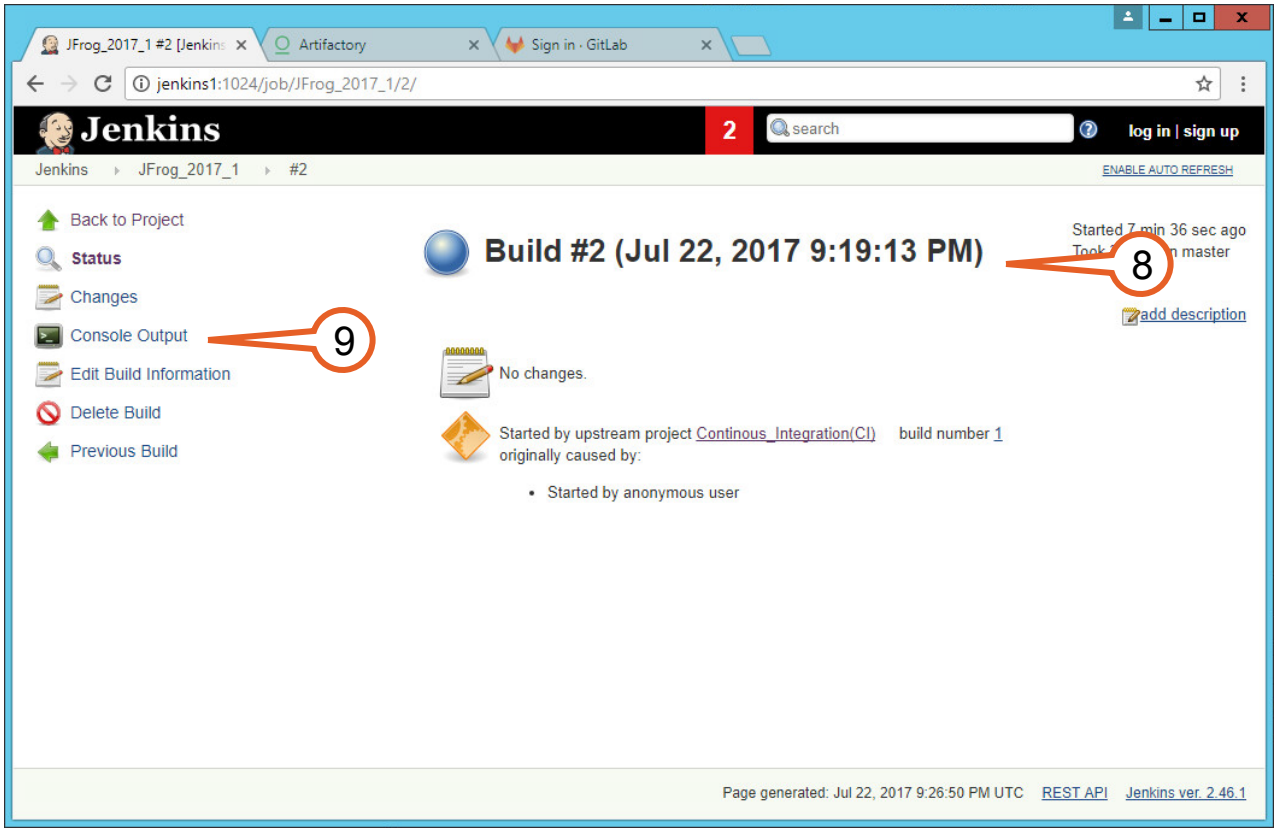


Figure 5-18:

The browser displays the Console Output page for the pipeline stage.

10. This page displays the console output from the pipeline execution. The console output on this page will update in real time as the pipeline continues to execute.
11. Once the pipeline completes you will see the completion status displayed at the bottom of the output.

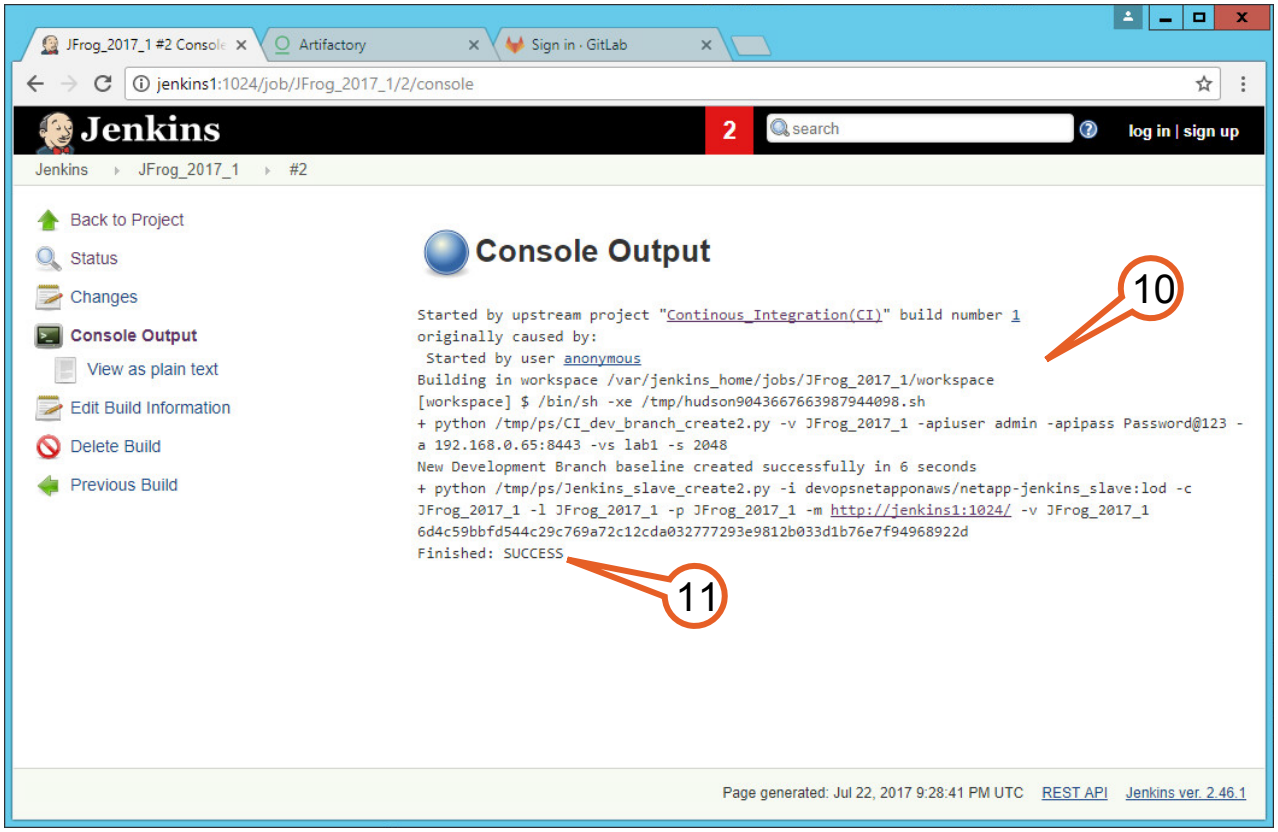


Figure 5-19:

If this pipeline completes successfully (and it should), then you now have a new CI container instance running within your docker cluster. Verify that is the case by examining your docker instances.

12. Switch to your PuTTY session for jenkins1.
13. Display a list of the running containers.

```
[root@jenkins1 ~]# docker ps
CONTAINER ID        IMAGE                                                 COMMAND
  CREATED             STATUS                  PORTS
                  NAMES
6d4c59bbfd54        devopsnetapponaws/netapp-jenkins_slave:lod        "/bin/sh -c 'exec ..."
  11 minutes ago      Up 11 minutes
                  JFrog_2017_1
3b9fe3ba4585        devopsnetapponaws/netapp-jenkins-plugin-2.0:lod   "/bin/tini -- /usr..."
  3 hours ago         Up 3 hours              8080/tcp, 50000/tcp
                  jenkins.1.jdf6vqyf2k1ggenjjamju1oqo
22edf3dd1cdc        devopsnetapponaws/netapp-jenkins_gitlab           "/usr/bin/startcon..."
   2 months ago        Up 3 hours (healthy)   0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp,
 0.0.0.0:10022->22/tcp   JFrog_OSS_Repo
[root@jenkins1 ~]#
```

Prevously you observed that two containers were running on this host. Now there are three. The container named “JFrog_2017_1” is your new continuous integration build instance.

14. Open a bash session inside the JFrog_2017_1 container.

```
[root@jenkins1 ~]# docker exec -it JFrog_2017_1 /bin/bash
root@6d4c59bbfd54:/#
```

15. Display the container's list of mounted volumes.

```
root@6d4c59bbfd54:/# df
Filesystem
      1K-blocks    Used Available Use% Mounted on
```

```
/dev/mapper/docker-253:0-68215161-
e84f18bf0facc6e980e23c183b1d11196c58ba7229854761b388046486cfda39  10474496  872756   9601740
   9% /
tmpfs
         941996       0    941996   0% /dev
tmpfs
         941996       0    941996   0% /sys/fs/cgroup
192.168.0.132:/JFrog_2017_1
        2166208     192   2166016   1% /workspace/JFrog_2017_1
/dev/mapper/rhel-root
       37202180 8905024  28297156  24% /etc/hosts
shm
          65536       0     65536   0% /dev/shm
tmpfs
         941996       0    941996   0% /sys/firmware
root@6d4c59bbfd54:/#
```

The volume NFS mounted as /workspace/JFrog_2017_1 contains the source code build tree for the master builds. Once again, this is a persistent volume hosted on NetApp storage.

16. Change directory into the CI build directory and view its contents.

```
root@6d4c59bbfd54:/# cd /workspace/JFrog_2017_1
root@6d4c59bbfd54:/workspace/JFrog_2017_1# ls -l
total 0
[root@jenkins1 ~]#
```

The directory is empty! That's because, while this pipeline handles creating the CI instance for your master build, it does not handle populating it with the source code for your project. You will handle that in the next exercise.

17. Terminate the bash session to the CI container.

```
[root@jenkins1 ~]# exit
logout
```

5.5 Set Up a Continuous Integration Build

In this exercise you will set up a master continuous integration build within your newly created CI environment. This build will utilize a Maven “Hello World” project from JFrog's set of sample source code, which has been pre-loaded for you into GitLab.

1. In Chrome, navigate to the **Sign in - Gitlab** tab.
2. Log in as the user **`root`** with the password **`Netapp1!`**.

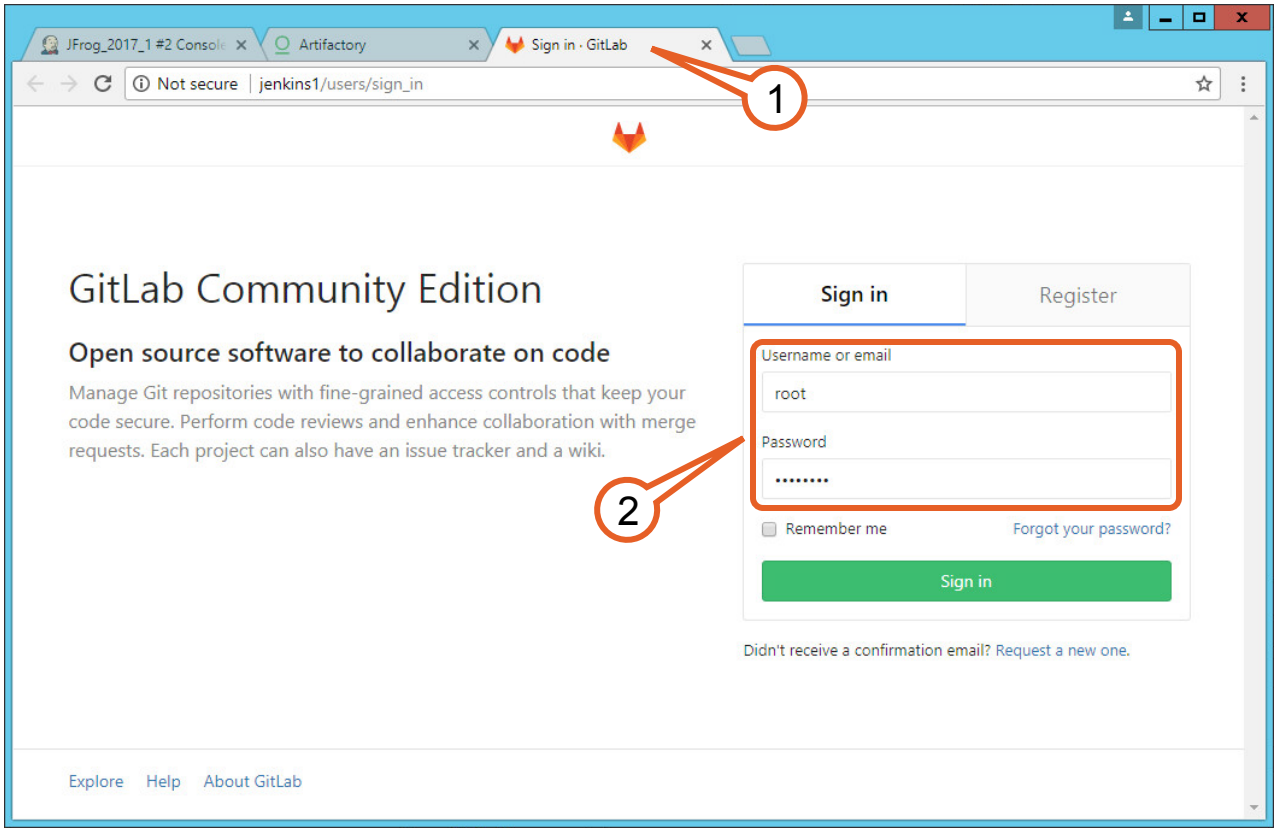


Figure 5-20:

The browser displays the Gitlab “Projects” page.

3. Under the “Your Projects” section of the page, click **JFrogSamples**.

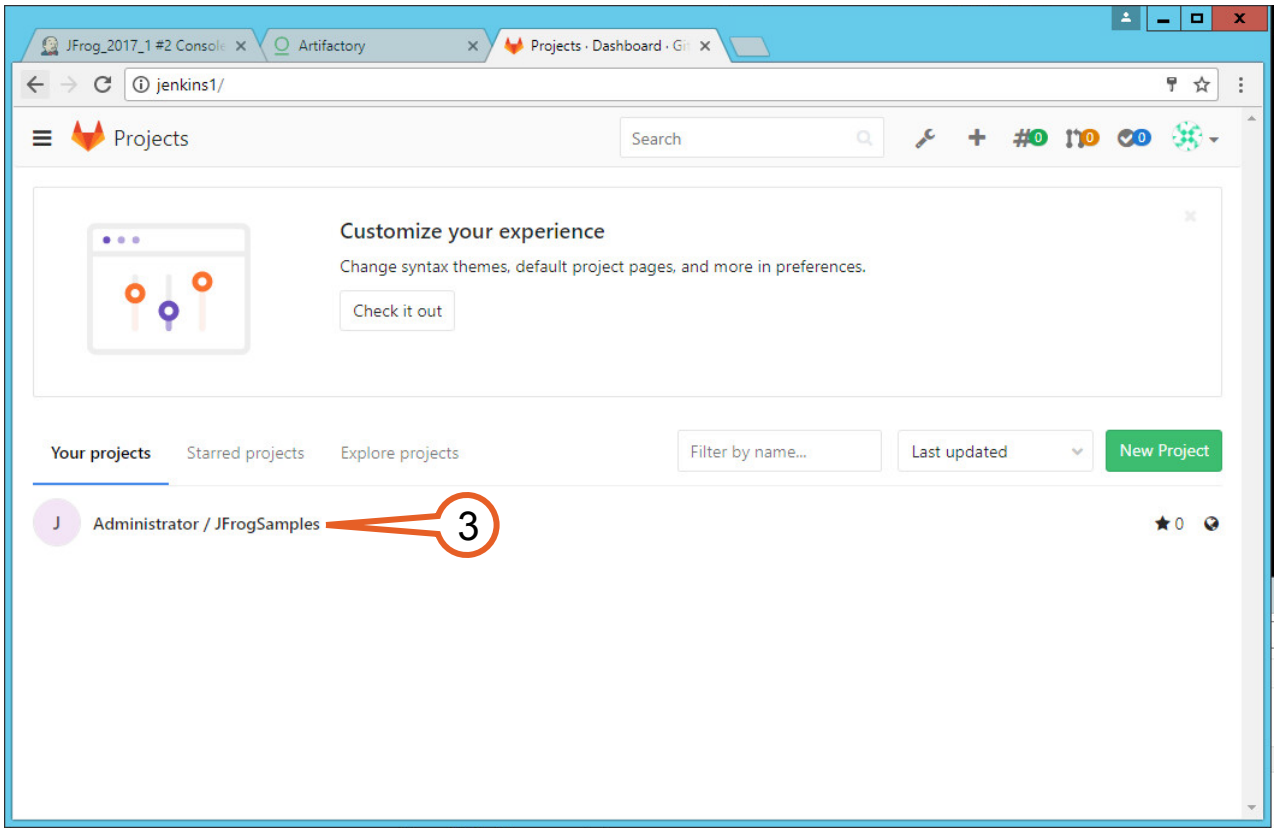


Figure 5-21:

The browser displays the “JFrogSamples” project page.

4. Note the contents of the HTTP box at the top of the page. This is the URL used to access the project from git (http://root@jenkins1/root/JFrogSamples.git).
5. Click the button just to the right of the HTTP box to copy the URL to the clipboard.

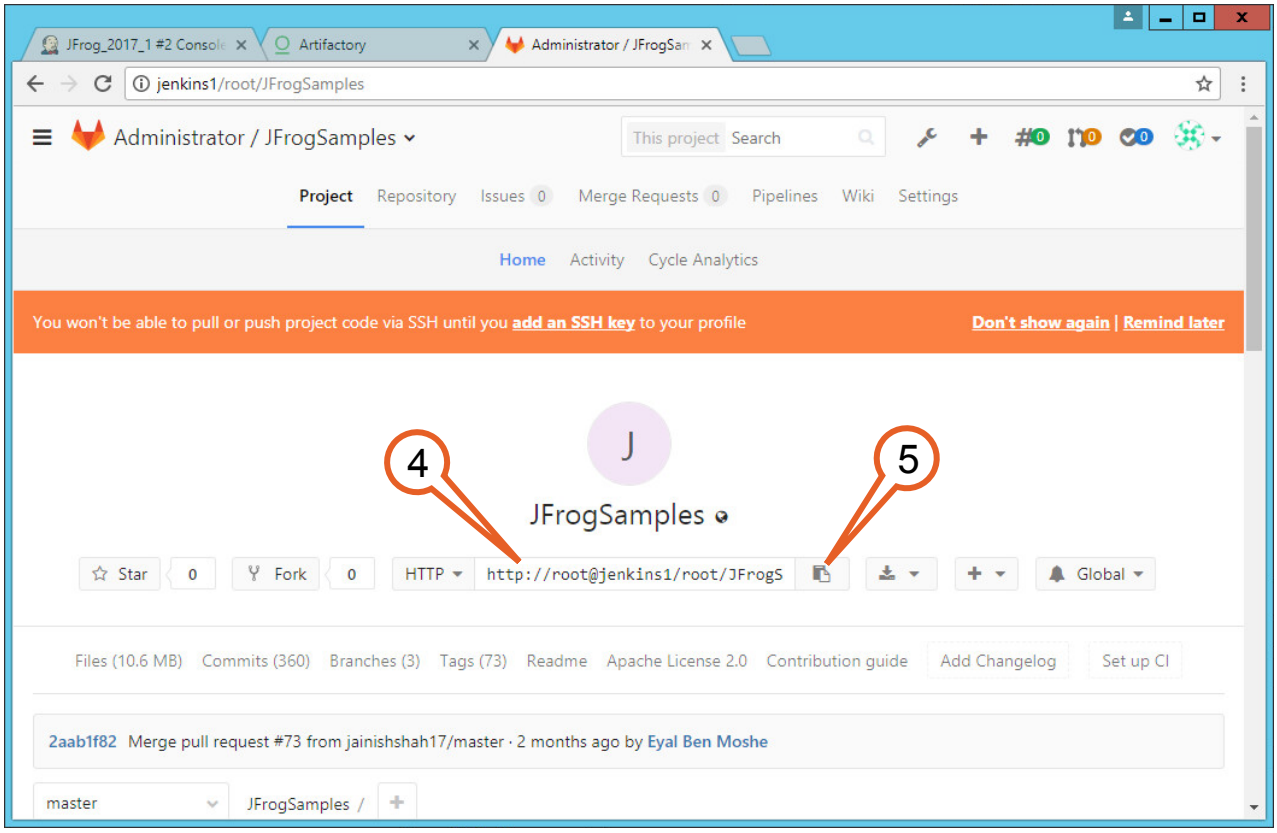


Figure 5-22:

6. Scroll down the page and locate the maven-example folder that contains the subproject that your new CI build will be building.
7. Click **maven-example**.

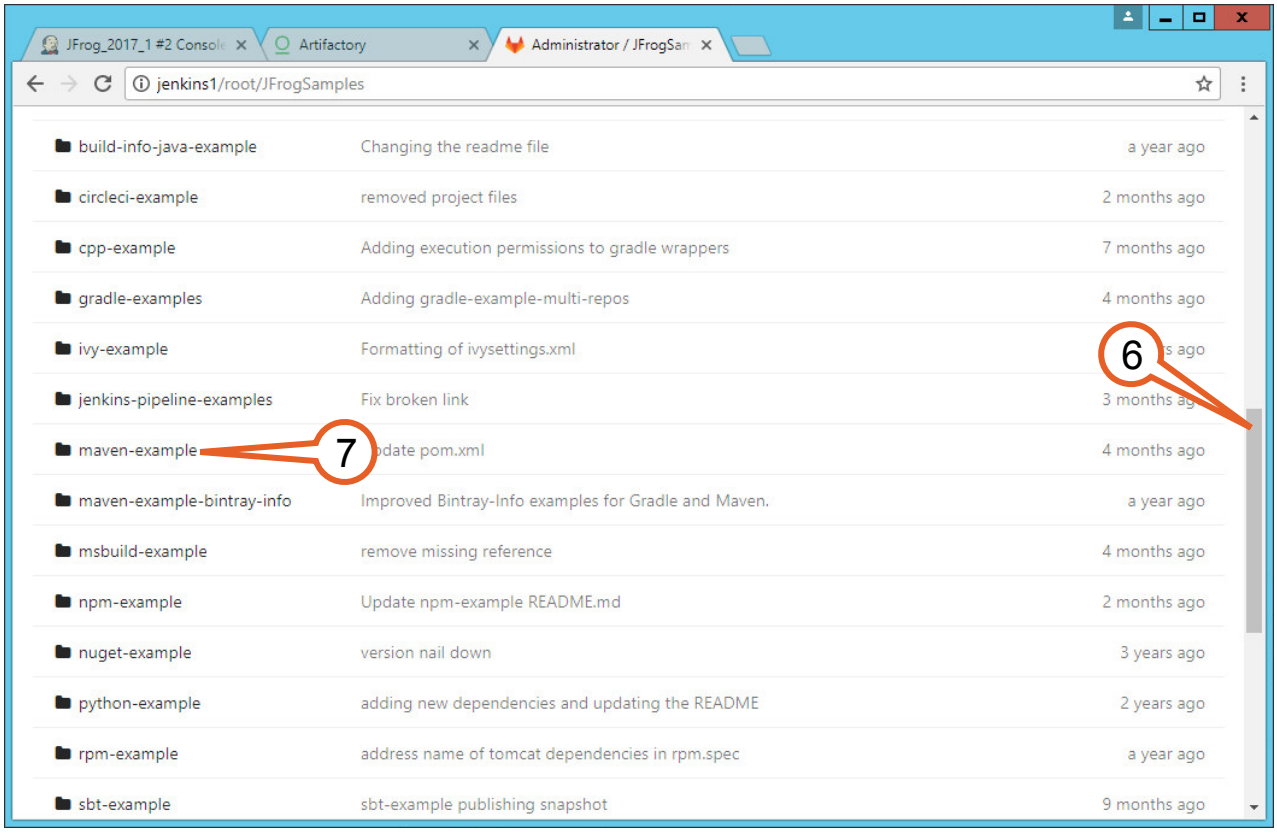


Figure 5-23:

8. Verify the presence of the pom.xml file in this folder. Maven uses a Project Object Model (POM) to store the information it needs to build the project. The POM is implemented as an XML file, and resides in the project's base directory. You will specify the path to this subproject's POM file when you configure the build in a later step of this exercise.

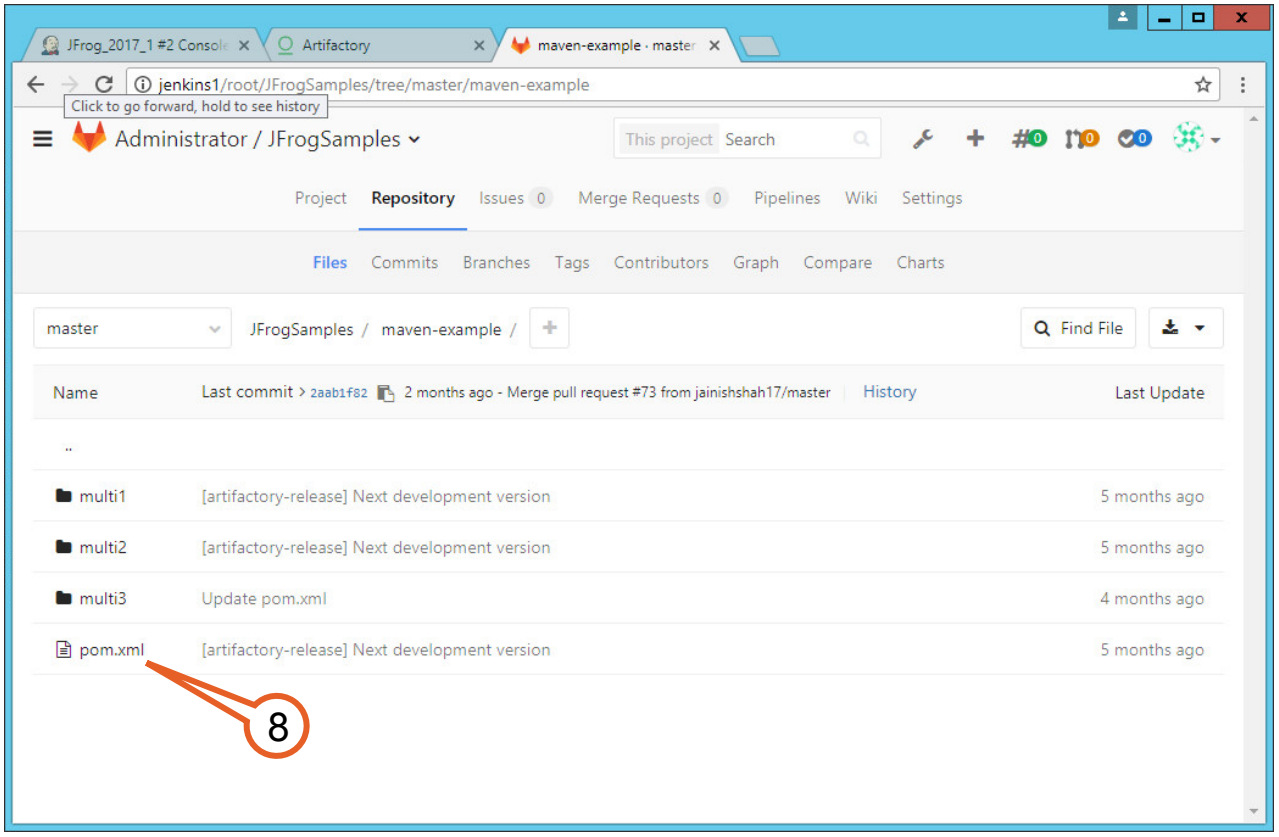


Figure 5-24:

9. In Chrome, select the tab for the Jenkins UI (http://jenkins1:1024/). At this point that tab is likely named "JFrog_2017_1 #2 Console" as a result of a previous exercise.
10. Click on the **Jenkins** graphic in the upper left-hand corner of the window.

Figure 5-25:

The browser returns to the Jenkins home page, which is the "Pipelines [Jenkins]" page. The tab title changes to that value too.

11. Click **New Item**.

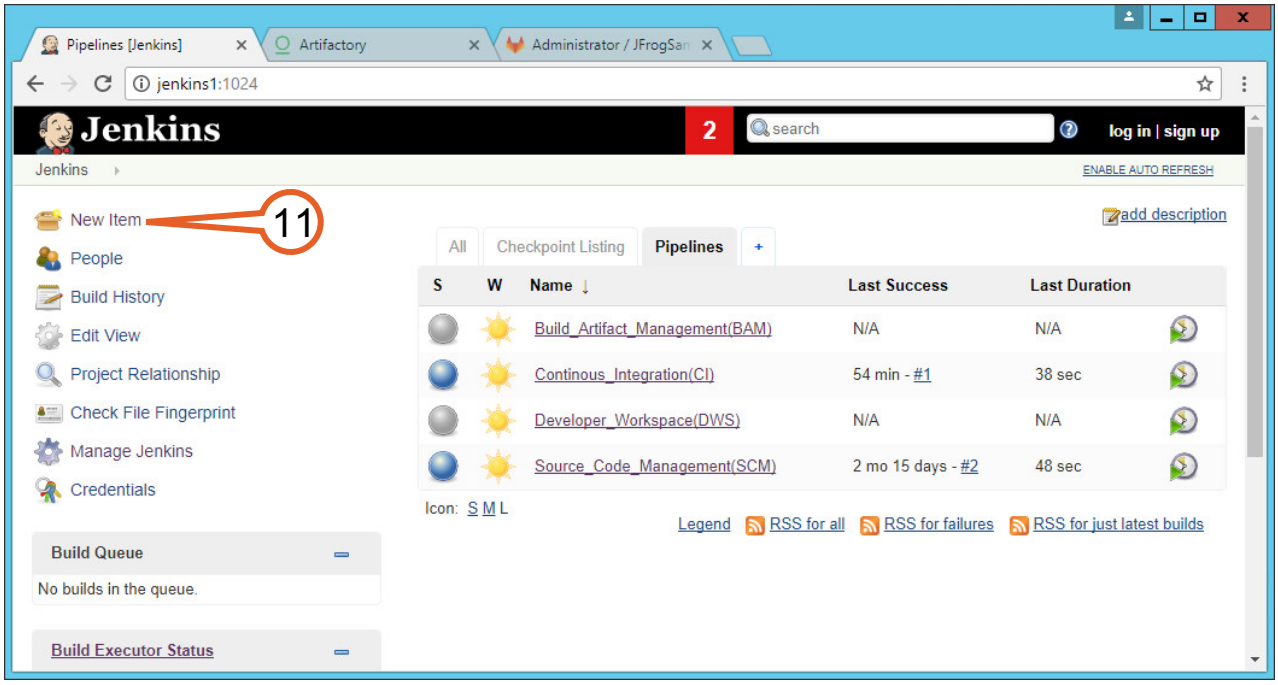


Figure 5-26:

The browser display the new item page.

12. In the “Enter an item name” field, enter **`JFrog_CI_Build`**.
13. Click **Maven project**.
14. Click **OK**.

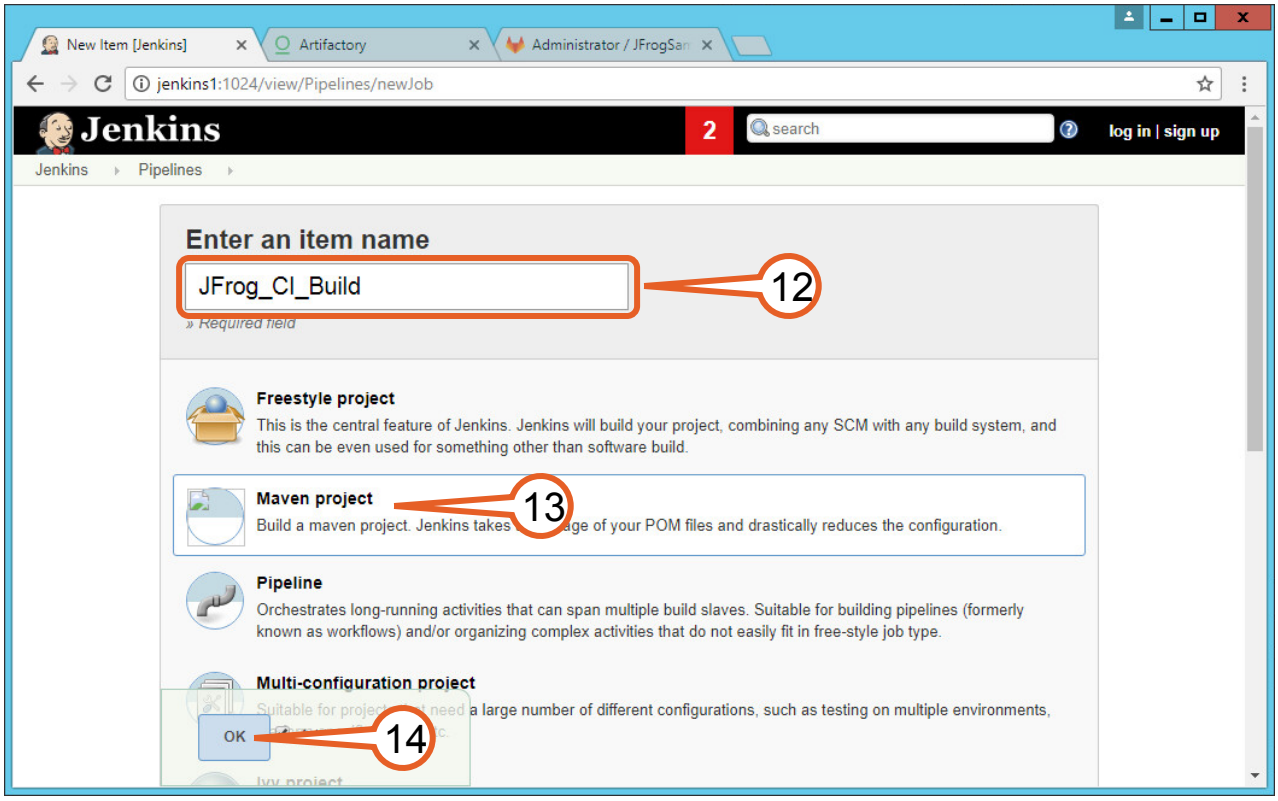


Figure 5-27:

The Jenkins UI displays properties page for the JFrog_CI_Build

15. Select the **General** tab.
16. Under the “Description” section, check the **Restrict where this project can be run** checkbox, which in turn causes a new field to appear.
17. Set the “Label Expression” field to **`JFrog_2017_1`**.

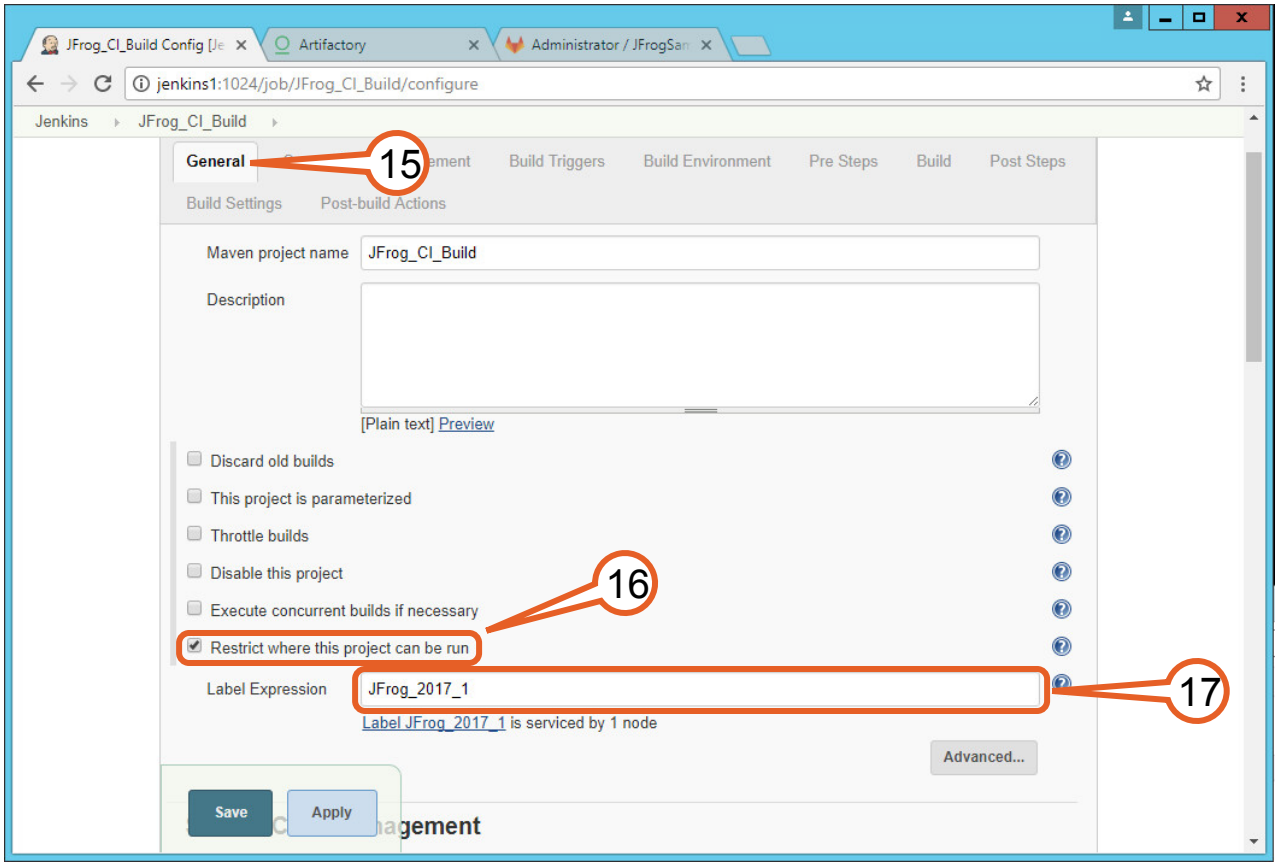


Figure 5-28:

18. Select the **Source Code Management** tab.
19. In the “Source Code Management” section, check the **Git** checkbox, which causes some new fields to appear.
20. Set the “Repository URL” field to **`http://root@jenkins1/root/JFrogSamples.git`** by pasting in the URL value you copied to the clipboard noted earlier in this exercise. You may need to click outside the value box after pasting in order for the text to be accepted.

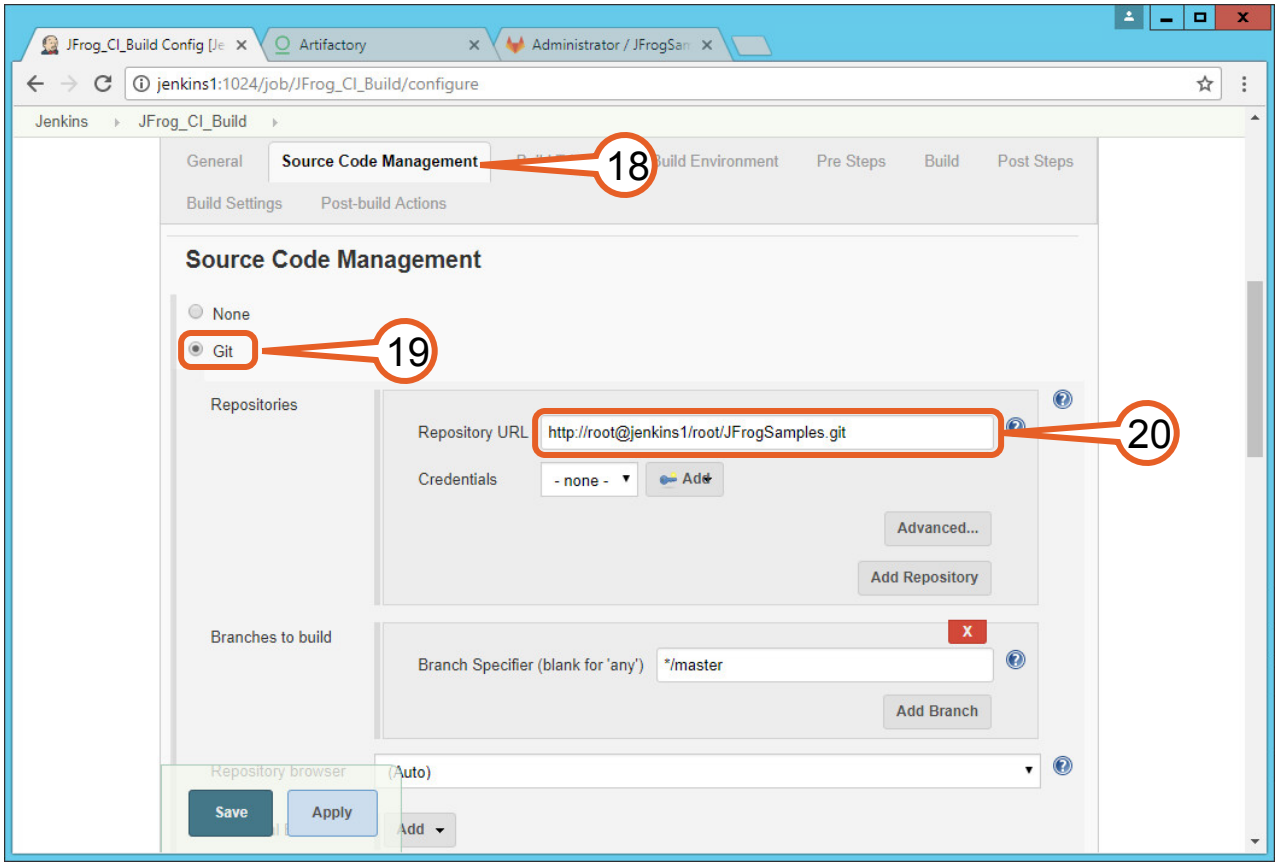


Figure 5-29:

21. Select the **Build Environment** tab.
22. Check the **Resolve artifacts from Artifactory** checkbox.
23. Click **Refresh Repositories**.
24. When the refresh completes, which should only take a few seconds, an “Items refreshed successfully” message appears to the left of the Refresh Repositories button.
25. The “Resolution releases repository” and “Resolution snapshots repository” fields are now populated with the value “LocalRepo”.

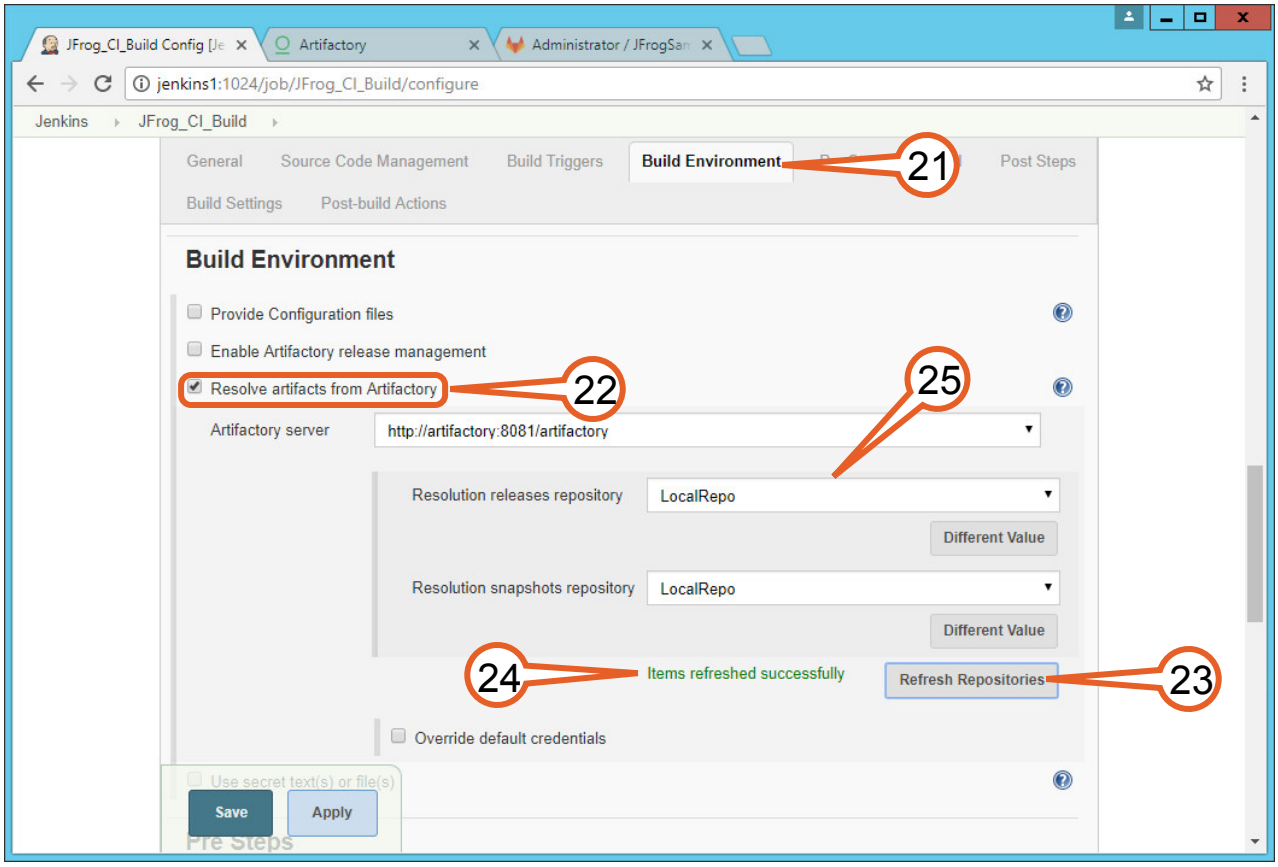


Figure 5-30:

26. Select the **Build** tab.
27. Change the “Root POM” field to the value **`maven-example/pom.xml`**, which is the path to the subproject POM file you looked at earlier in this exercise.
28. Set the “Goals and options” field to **`install -DskipTests`** so your CI build will skip the execution of any tests in the Maven SureFire test framework.
29. Click **Advanced**.

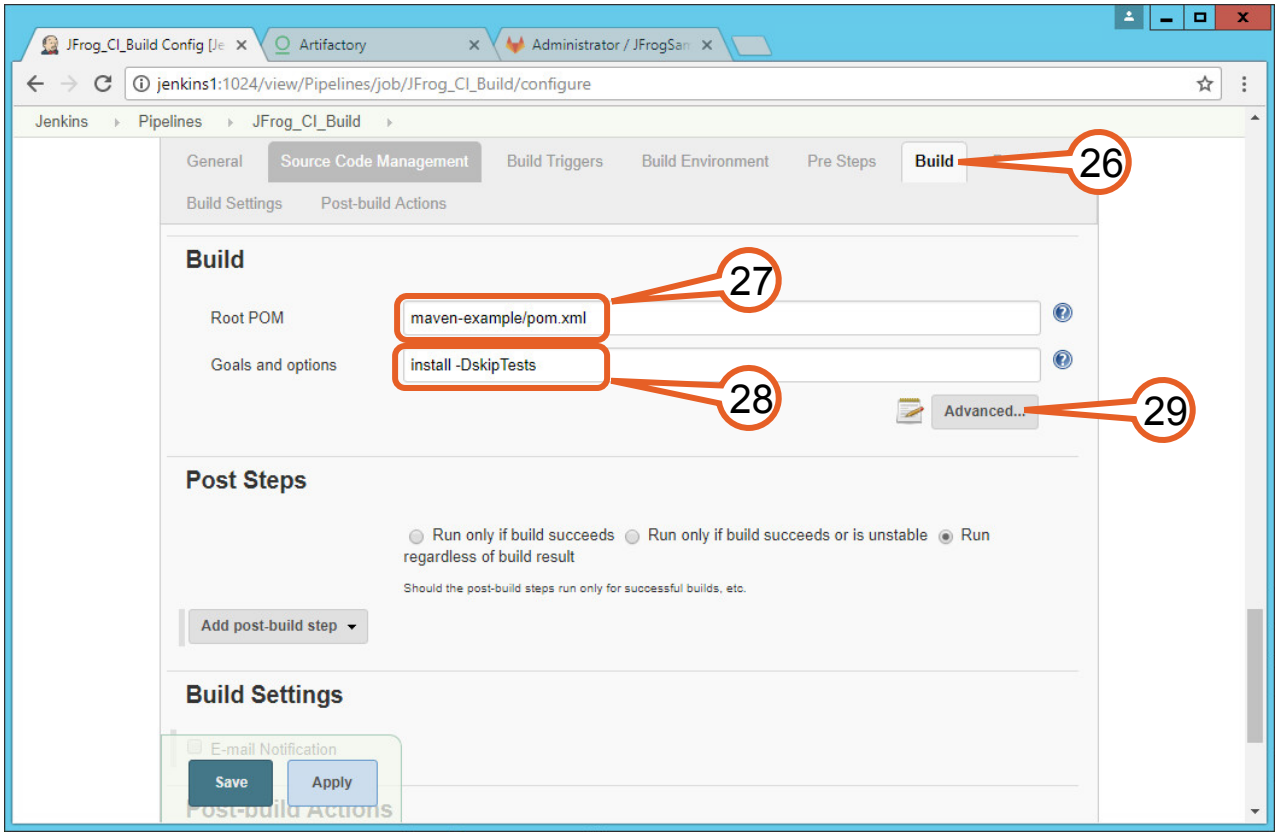
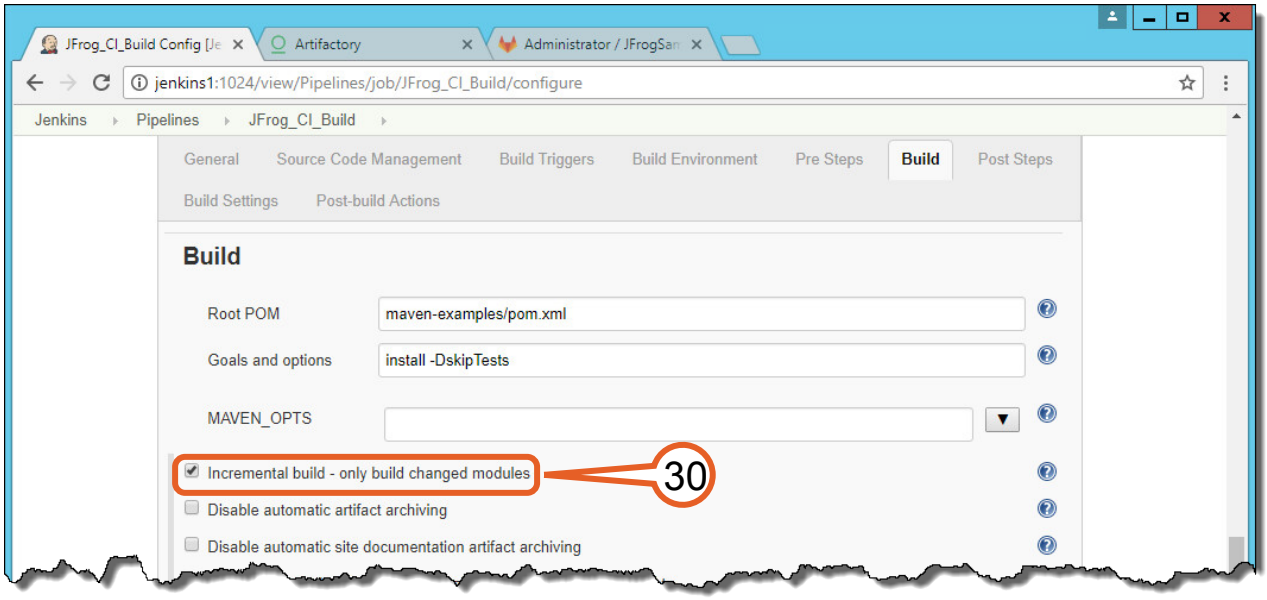


Figure 5-31:

The browser displays an additional set of configuration properties.

30. Check the **Incremental build - only build changed modules** checkbox.

Figure 5-32:



31. Scroll down the page and check the **Use private Maven repository** checkbox, which causes a new sub-field to appear.

32. Set the “Strategy” field dropdown to the value **Local to the workspace**.
33. Check the **Use custom workspace** checkbox, which causes some new sub-fields to appear.
34. Set the “Directory” field to **/workspace/JFrog_2017_1**. Once again, after you enter the value click outside the field to cause it to be accepted.

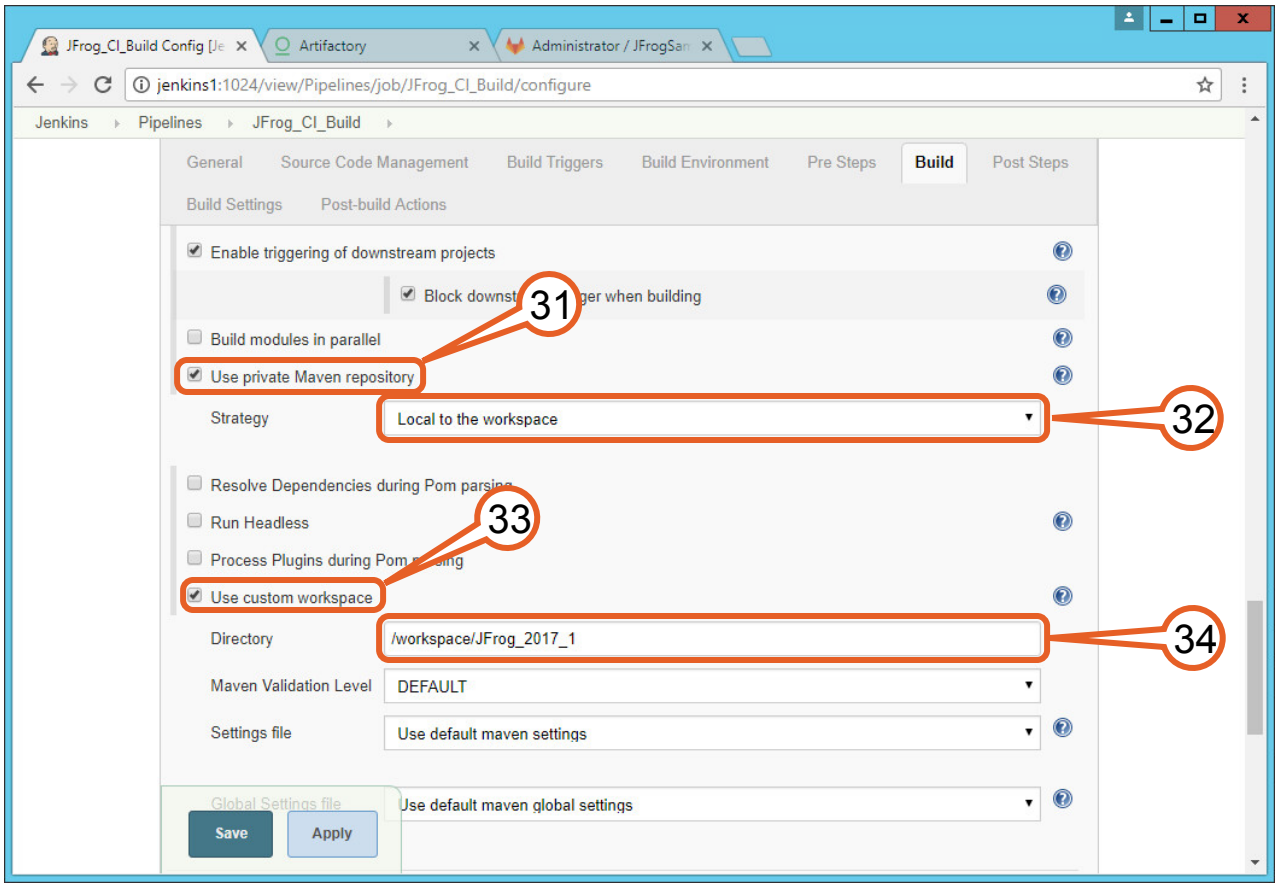


Figure 5-33:

35. Select the **Build Triggers** tab.
36. Check the **Poll SCM** checkbox, which causes a new sub-field to appear.
37. Set the “Schedule” textbox to * * * * *, then click outside the text box so that the UI will accept the text. This value causes Jenkins to poll the SCM every minute for changes to the source tree, and if a change is detected it will trigger the JFrog_CI_Build job to launch a build.
38. Click **Save**.

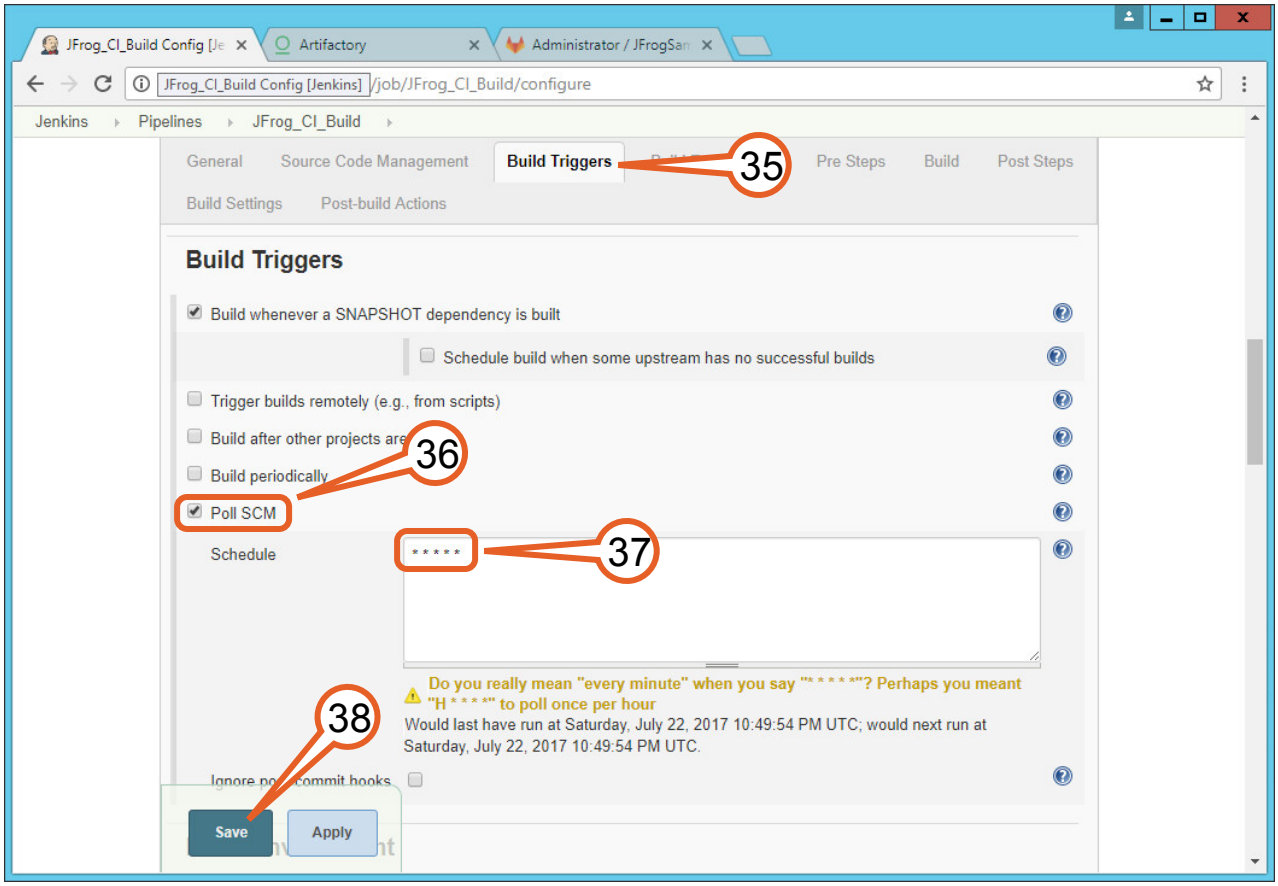


Figure 5-34:

Jenkins saves the configuration changes for the JFrog_CI_Build job, and the browser returns to the job's project page.

39. The build triggers you set up in the preceding step check every minute to see if the SCM has code ready to be built, so if you wait a minute Jenkins will automatically launch the first build for you. When that happens you will see a build job show up in the “Build History” section.

Tip: In a production environment you would almost certainly implement a less aggressive build trigger schedule than is implemented in this lab. In that case you would likely need to manually trigger a build here using the “Build Now” link rather than wait for a scheduled trigger.

40. If for some reason a build does not start automatically within a munite or so, then click **Build Now** to start the build manually.

41. In the “Build History” section, click on the **ball** to the left of the new build job, which will open the console log for this job.

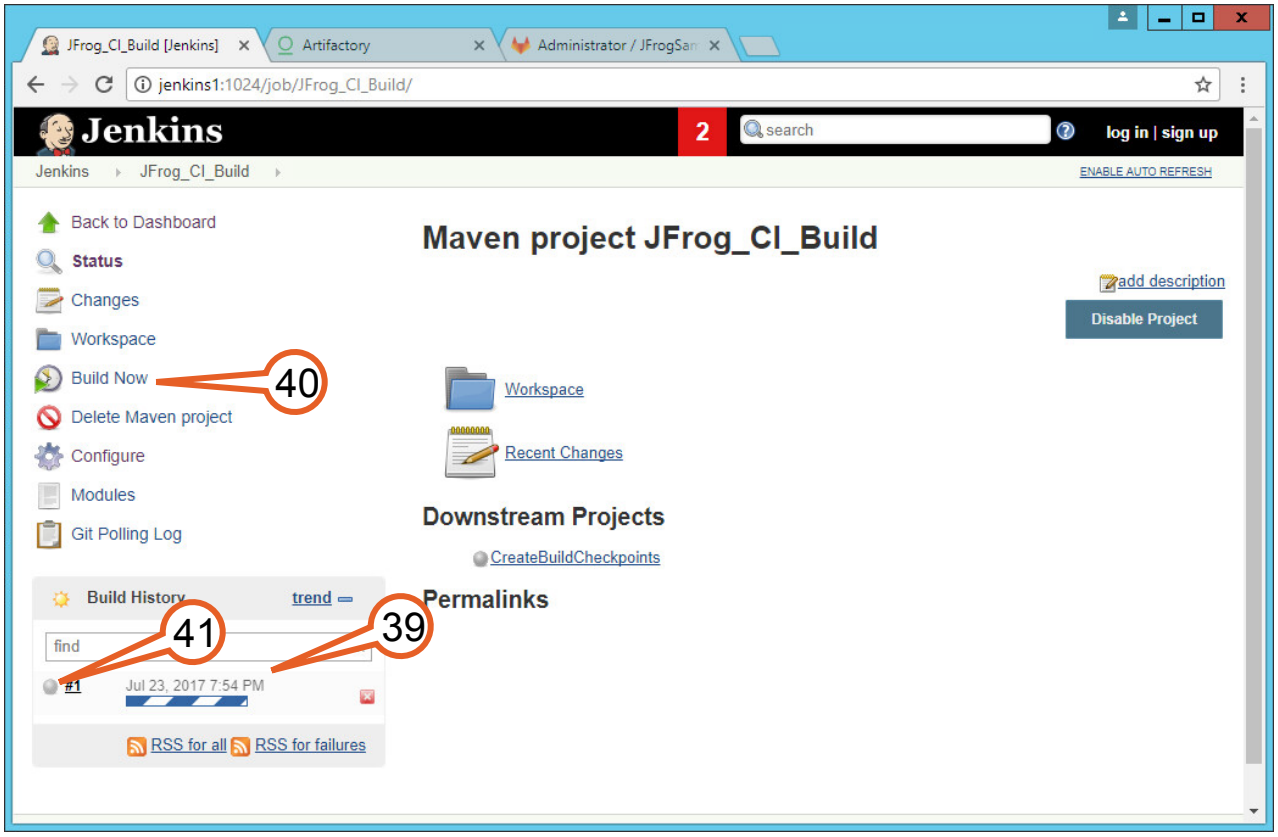


Figure 5-35:

42. In the right pane, examine the console output. As the build runs, it encounters dependencies that it needs to fulfill in order to proceed. In this exercise those dependencies are various maven plugins, and those plugins are available as artifacts in the Artifactory server. The log output shows the plugins being downloaded from the Artifactory server. The build will take a few minutes to complete, as this is the first build for this project which is why it has to download so many artifacts.

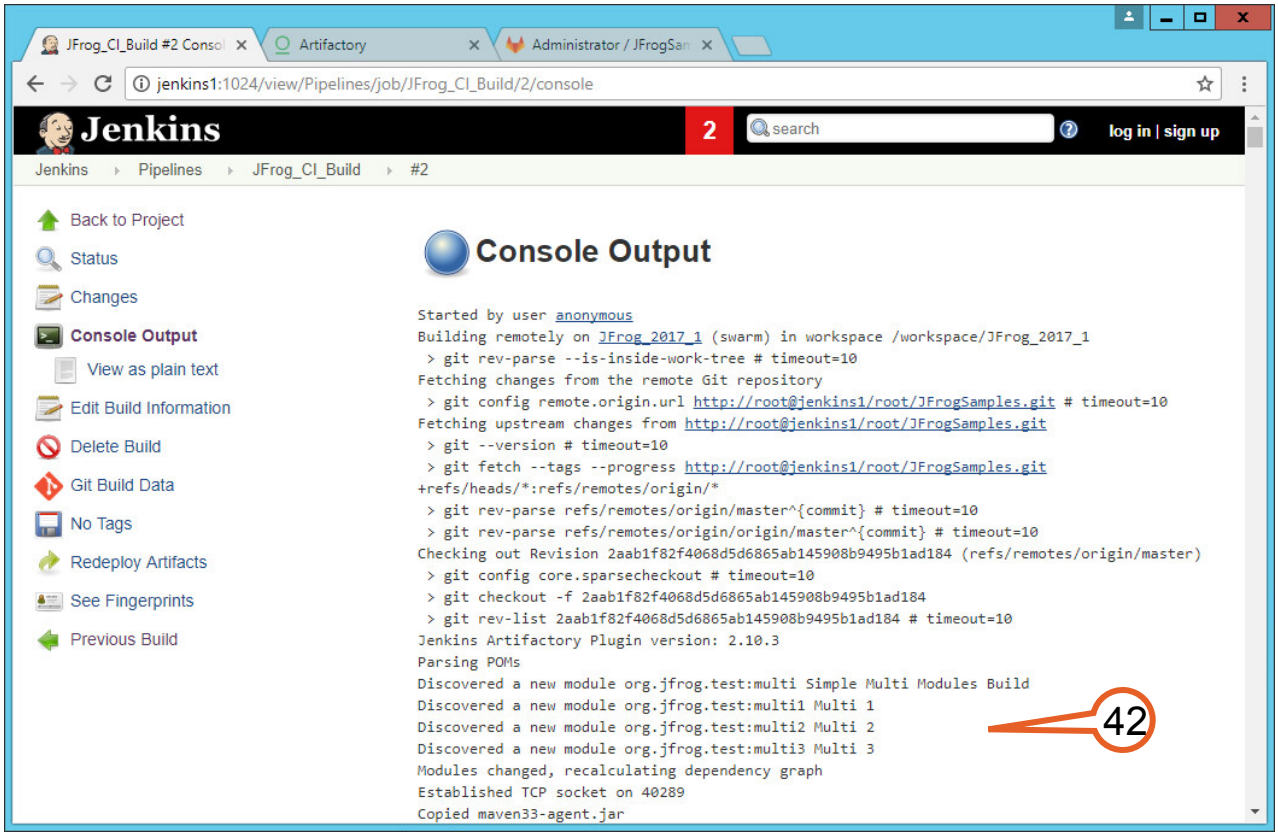


Figure 5-36:

43. When the build finishes, if you scroll down to the bottom of the page you should see a line that states “Finished: SUCCESS” indicating a successful build.

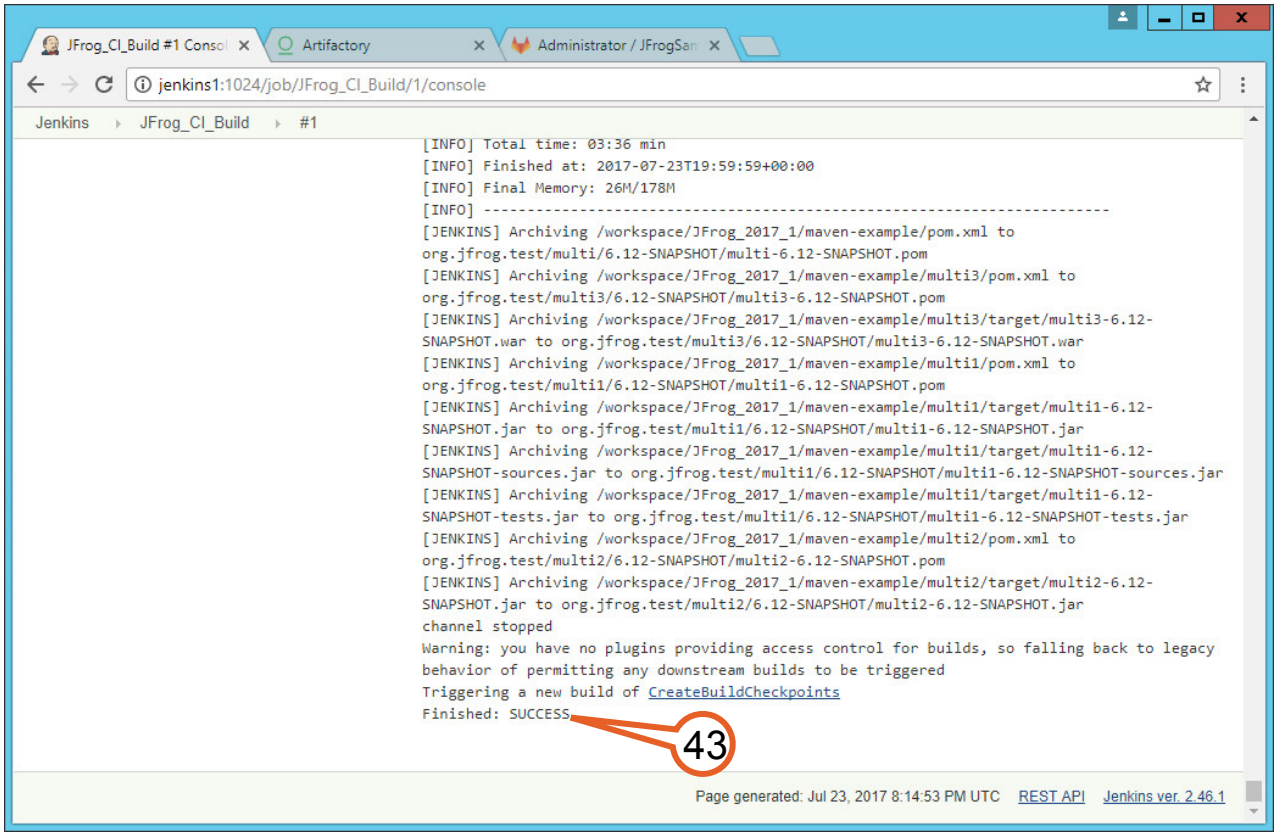


Figure 5-37:

44. Scroll back to the top of the page and click on the **Jenkins** graphic to return to the default “Pipelines [Jenkins]” page.

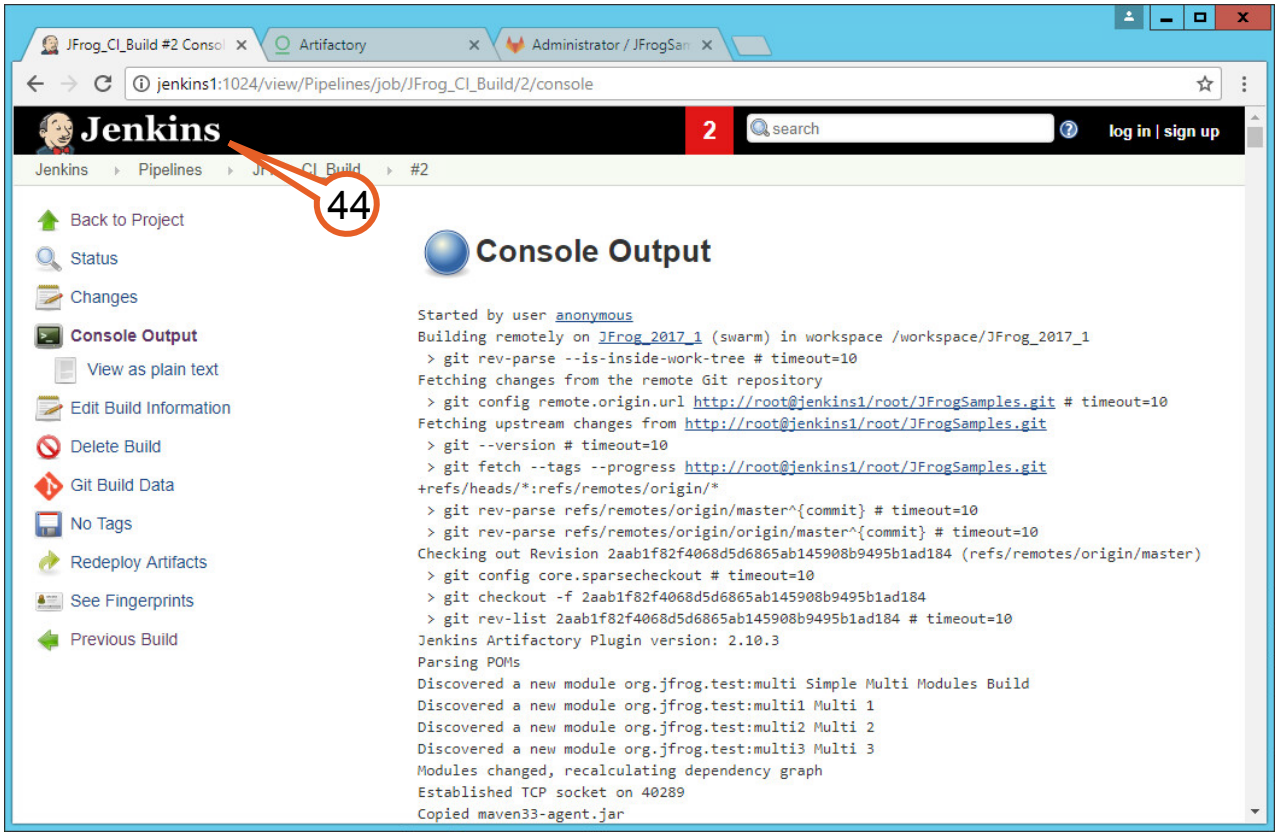


Figure 5-38:

45. In your PuTTY session to jenkins1, open an interactive shell into the CI container, just as you did in the last exercise.

```
[root@jenkins1 ~]# docker exec -it JFrog_2017_1 /bin/bash
root@6d4c59bbfd54:/#
```

46. Display a list of the container's mounted volumes.

```
root@6d4c59bbfd54:/# df
Filesystem
      1K-blocks    Used Available Use% Mounted on
/dev/mapper/docker-253:0-68215161-
e84f18bf0facc6e980e23c183b1d11196c58ba7229854761b388046486cfda39  10474496  920156   9554340
   9% /
tmpfs
         941996       0    941996   0% /dev
tmpfs
         941996       0    941996   0% /sys/fs/cgroup
192.168.0.132:/JFrog_2017_1
        2166208   57216   2108992   3% /workspace/JFrog_2017_1
/dev/mapper/rhel-root
       37202180 9029332  28172848  25% /etc/hosts
shm
          65536       0     65536   0% /dev/shm
tmpfs
         941996       0    941996   0% /sys/firmware
root@6d4c59bbfd54:/#
```

Recall that the /workspace/JFrog_2017_1 mount represents the container's peristent storage.

47. Change directory to the /workspace/JFrog_2017_1 folder for the CI builds that you looked at in the last exercise and list its contents.

```
root@6d4c59bbfd54:/# cd /workspace/JFrog_2017_1
root@6d4c59bbfd54:/workspace/JFrog_2017_1# ls -l
total 84
```

```
-rw-r--r--  1 root root   469 Jul 22 22:51 CONTRIBUTING.md
-rw-r--r--  1 root root 11358 Jul 22 22:51 LICENSE
-rw-r--r--  1 root root    67 Jul 22 22:51 README
drwxr-xr-x  5 root root  4096 Jul 22 22:51 artifactory-maven-plugin-example
drwxr-xr-x  2 root root  4096 Jul 22 22:51 bash-example
drwxr-xr-x  3 root root  4096 Jul 22 22:51 build-info-java-example
drwxr-xr-x  8 root root  4096 Jul 22 22:51 circleci-example
drwxr-xr-x  5 root root  4096 Jul 22 22:51 cpp-example
drwxr-xr-x  4 root root  4096 Jul 22 22:51 gradle-examples
drwxr-xr-x  4 root root  4096 Jul 22 22:51 ivy-example
drwxr-xr-x 12 root root  4096 Jul 22 22:51 jenkins-pipeline-examples
drwxr-xr-x  5 root root  4096 Jul 22 22:51 maven-example
drwxr-xr-x  6 root root  4096 Jul 22 22:51 maven-example-bintray-info
drwxr-xr-x  8 root root  4096 Jul 22 22:51 msbuild-example
drwxr-xr-x  3 root root  4096 Jul 22 22:51 npm-example
drwxr-xr-x  5 root root  4096 Jul 22 22:51 nuget-example
drwxr-xr-x  4 root root  4096 Jul 22 22:51 python-example
drwxr-xr-x  2 root root  4096 Jul 22 22:51 rpm-example
drwxr-xr-x  3 root root  4096 Jul 22 22:51 sbt-example
root@6d4c59bbfd54:/workspace/JFrog_2017_1#
```

Unlike before, this folder is no longer empty. The CI build automatically checked out the project source code here and compiled the maven-example subproject. Note that the files in this directory tree are all owned by root/root, which will become significant in the next exercise.

Feel free to explore the rest of the maven-example directory structure if you wish, then move on to the next step of this exercise.

48. Terminate the interactive shell to the CI instance.

```
root@6d4c59bbfd54:/workspace/JFrog_2017_1# exit
exit
[root@jenkins1 ~]#
```

You now have a CI instance that has run a successful master build, including having pulled in all of the dependant artifacts and compiling the project source code. This is an important because, as you will see in the next exercise, with NetApp you can leverage this pre-compiled master build environment to make your developer more productive. Every time a new feature or functionality is integrated during the CI process, the build times are reduced because the subsequent builds are incremental.

5.6 Create a Developer Workspace

In this exercise you will create a developer workspace for an individual project developer. Creating a developer workspace with the NetApp-Jenkins integration involves instantiating a container from the standard image, checking out the source code, installing all the artifact dependencies, and then running a full build of the source code.

Creating a developer workspace with the NetApp-Jenkins integration is also vastly more efficient. The workflow still instantiates a container from the standard image, but then creates a FlexClone copy of the CI build's persistent volume (the volume for the CI master build) for use as the developer workspace's persistent volume. Creating that clone takes just seconds, and because the clone already includes all the required build artifacts and a full build of the source code, the developer who uses the workspace gains significant productivity by avoiding the need to regenerate those items.

1. In Chrome, you should be on the “Pipelines [Jenkins]” tab from the last exercise.
2. In the right pane, select the **Developer_Workspace(DWS)**.

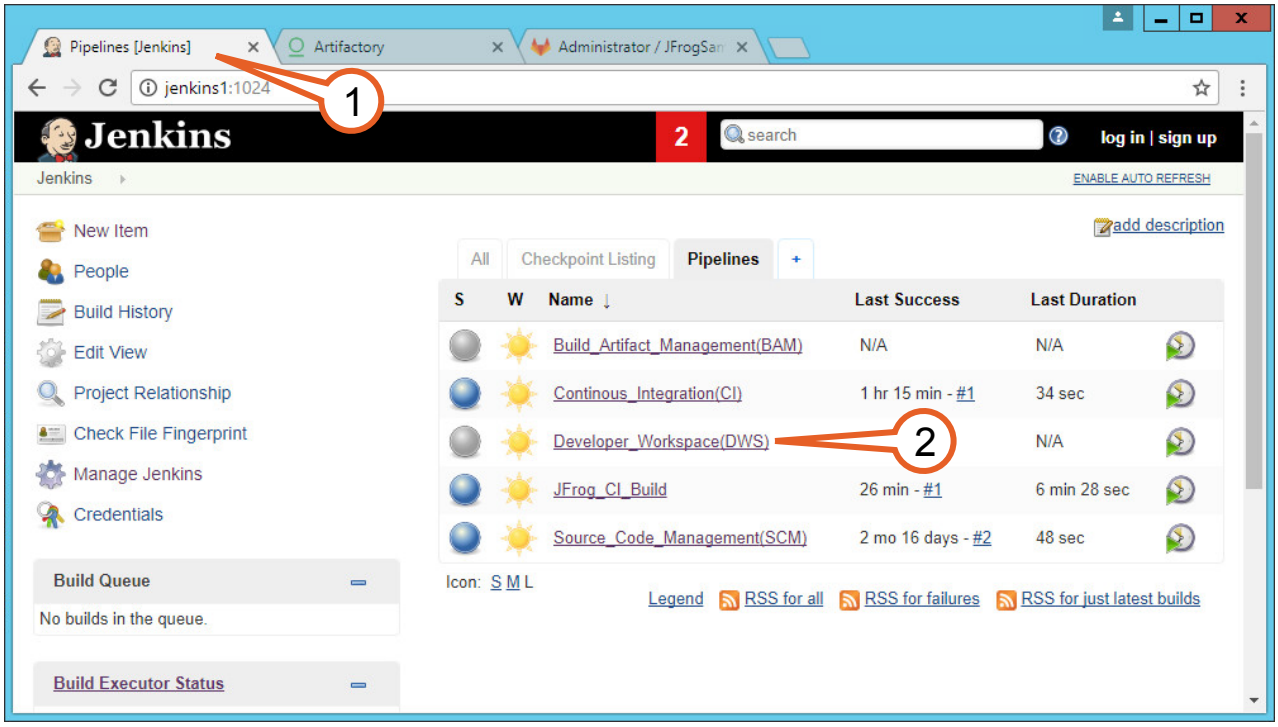


Figure 5-39:

The browser displays the “Pipeline Developer_Workspace(DWS)” page.

3. Observe that there are no builds listed under the “Build History” section, indicating that this pipeline (i.e., workflow) has not yet been run.
4. In the left pane, select **Build Now** to trigger an execution of this pipeline to create a new developer workspace.

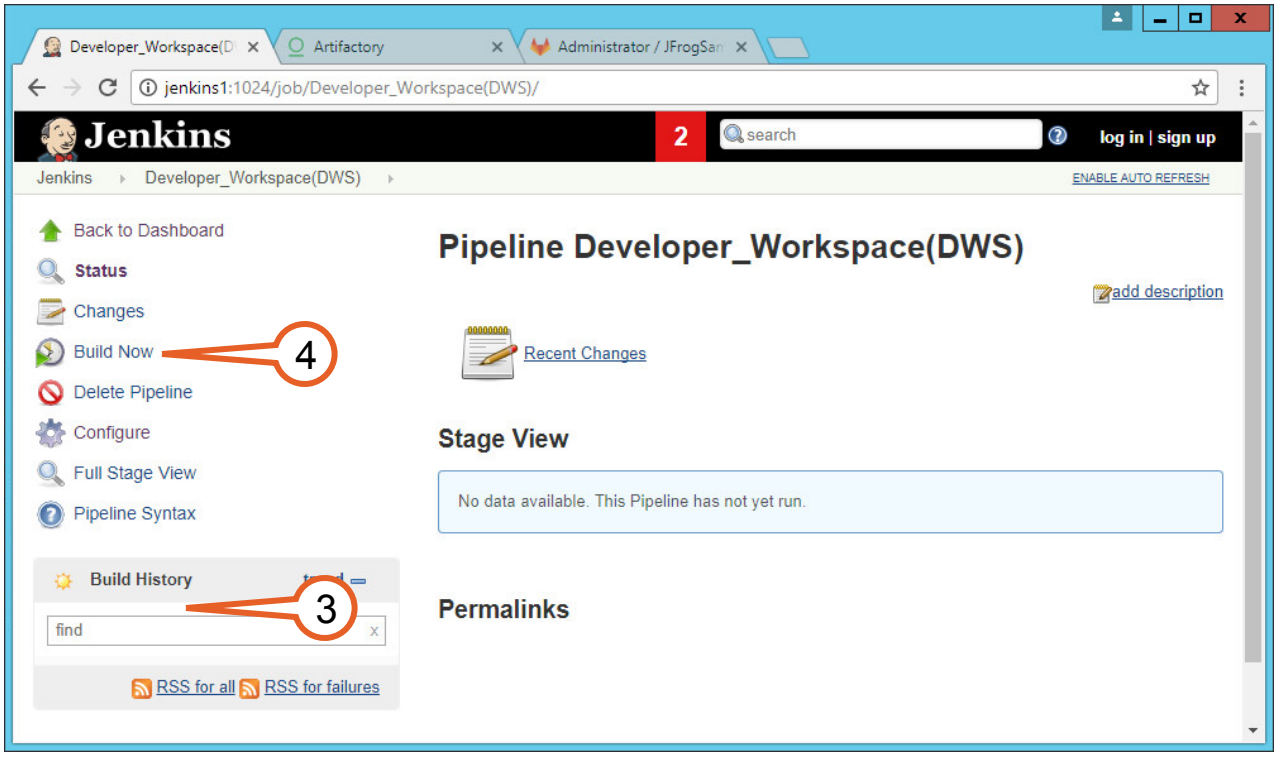


Figure 5-40:

5. A new build job appears under the Build History list, with a progress bar showing the build's progress.
6. In the right pane, under the “Stage View” section, hover your mouse pointer over the small box just above the blue “almost complete” bar. After a few moments a yellow pop-up window will appear to the left of the box.

   This pop-up is where you enter the details that the NetApp integration needs to the create the new developer workspace. That data includes the name of the new developer workspace, the name of the source volume to clone (which in this case will be the CI volume), and the Unix uid and gid that it should assign to the files on the newly cloned copy of this volume.
7. Enter the following values into the “Enter details for User Workspace” pop-up window. (Note that not all of these fields are shown in the accompanying screenshot.)

   Tip: If you start entering values in the pop-up and then mouse out of it before submitting, the pop-up window will close and those values will be gone when you return, so you will then need to re-enter them.

   - UUID: **`301`**
   - GID: **`300`**
   - VOLCI: **`JFrog_2017_1`** (pre-populated)
   - WSNAME: **`Dev1_JFrog_2017_1`**
8. Verify the values you entered from the last step are all specified correctly in the pop-up, then click **Proceed**.

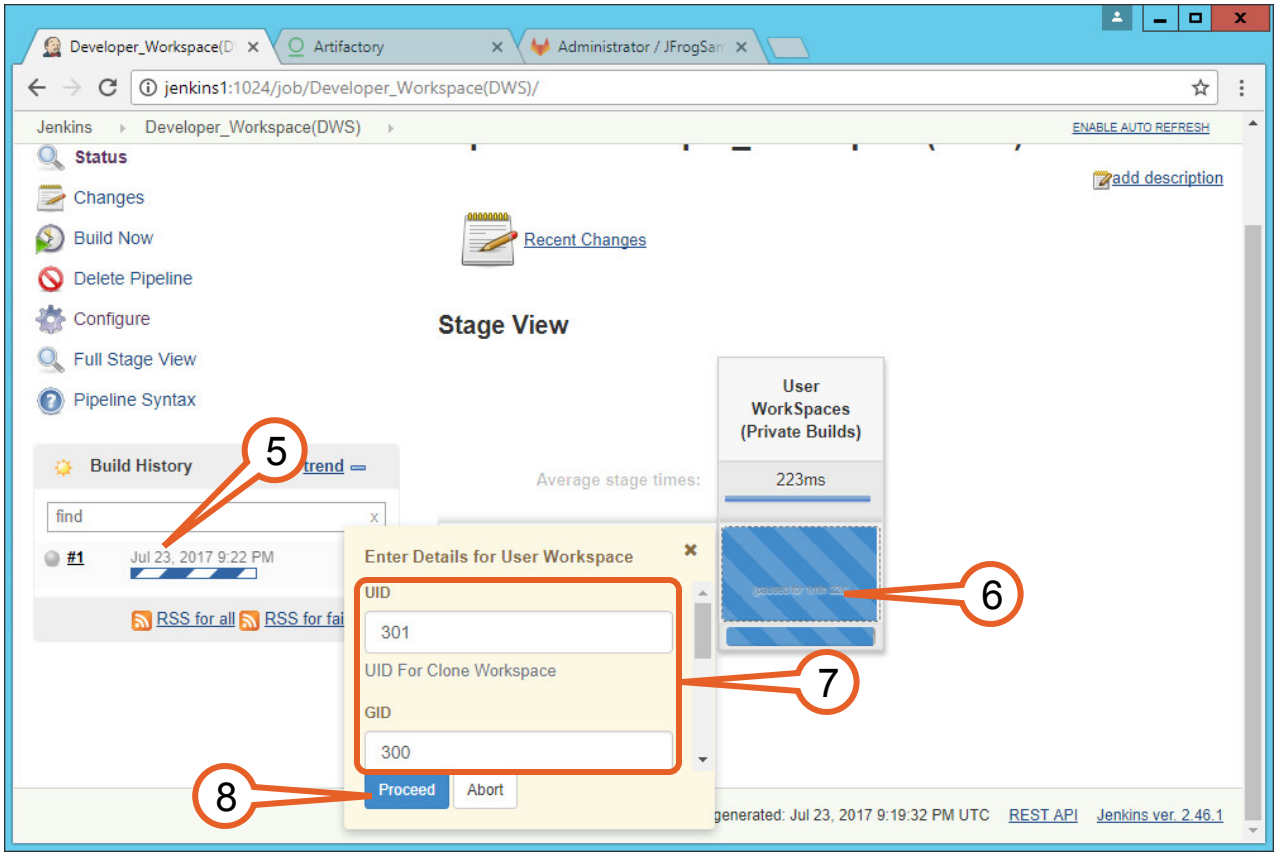


Figure 5-41:

The “Enter Details for User Workspace” pop-up window closes.

9. Once again hover your mouse over the small box above the blue “almost complete” bar. After a moment the “Select Checkpoint” pop-up box appears to the left of the box asking you to approve the selected checkpoint.
10. Click the **Please redirect to approve** link. Jenkins is unable to present the list of checkpoints directly in its standard UI, so this step will re-direct the UI out to the NetApp-Jenkins framework where you can directly select the desired checkpoint.

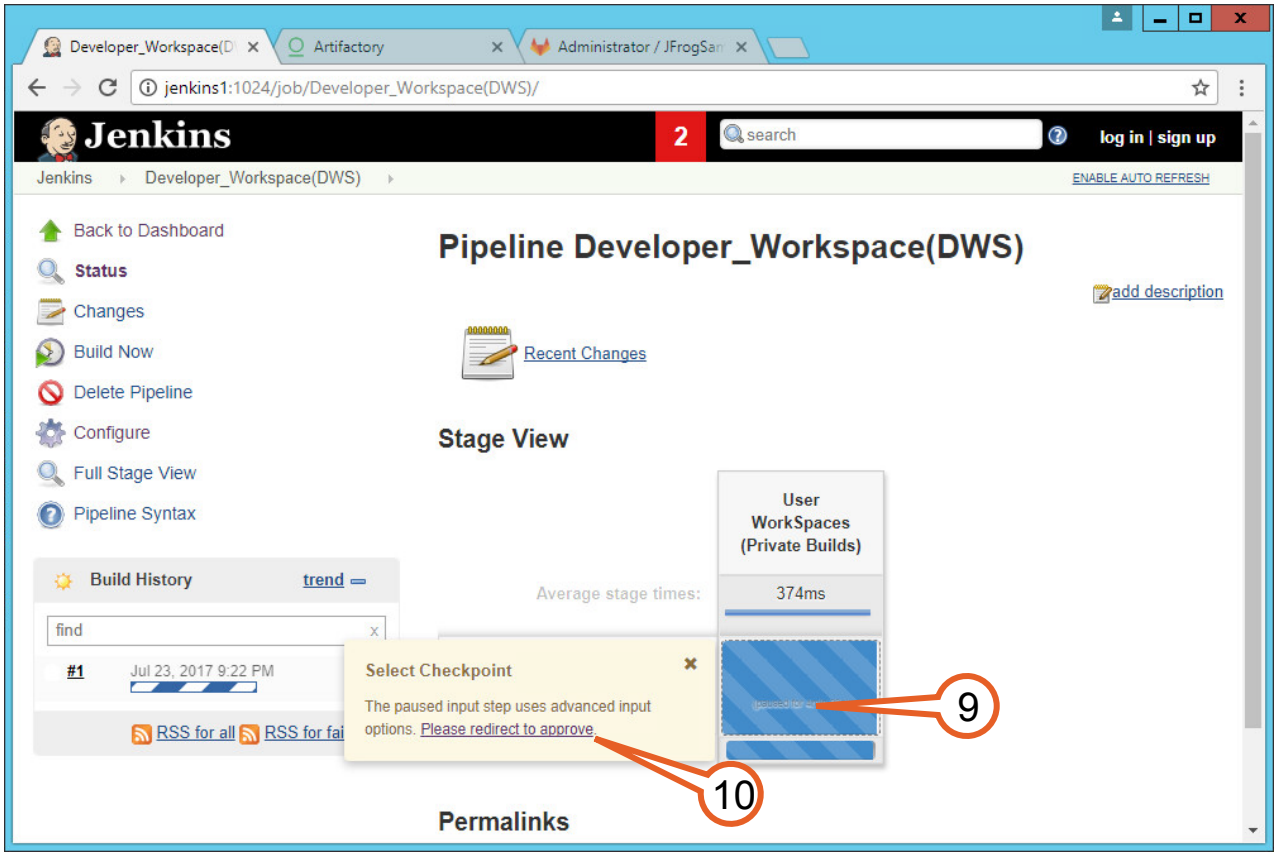


Figure 5-42:

The browser window changes to display the “Select Checkpoint” view. This is where you select checkpoint of the source volume to clone from (i.e., which volume snapshot to use as the FlexClone source).

11. In the “Select Checkpoint” dropdown, select the **JFrogBuild1-Checkpoint1** entry. This is the checkpoint associated with a completed full build of the CI master build volume.
12. Click **Proceed**.

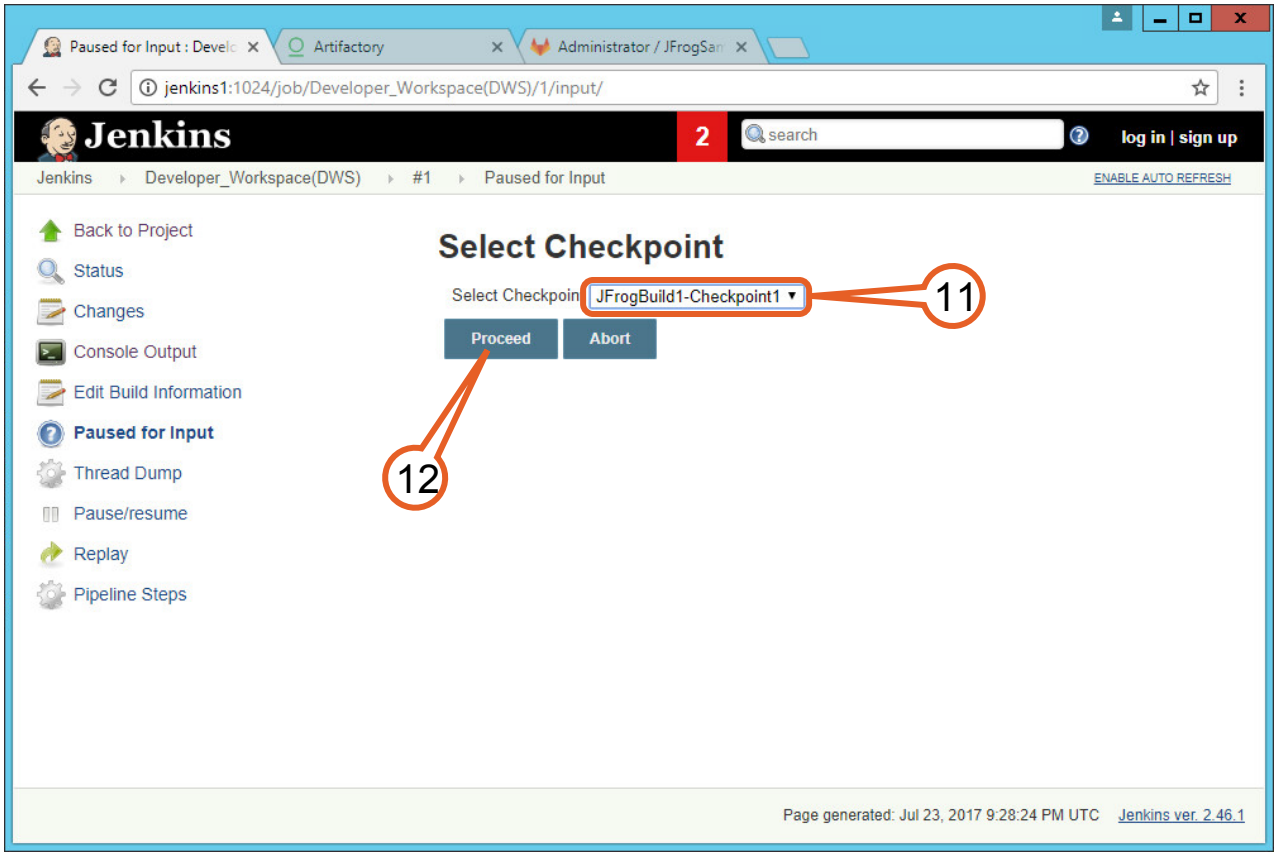


Figure 5-43:

This triggers a Jenkins pipeline job to create the new developer workspace. The browser's right pane changes to display the console output view for the executing pipeline.

13. When the pipeline finishes you will see a final output line that states “Finished: SUCCESS”.

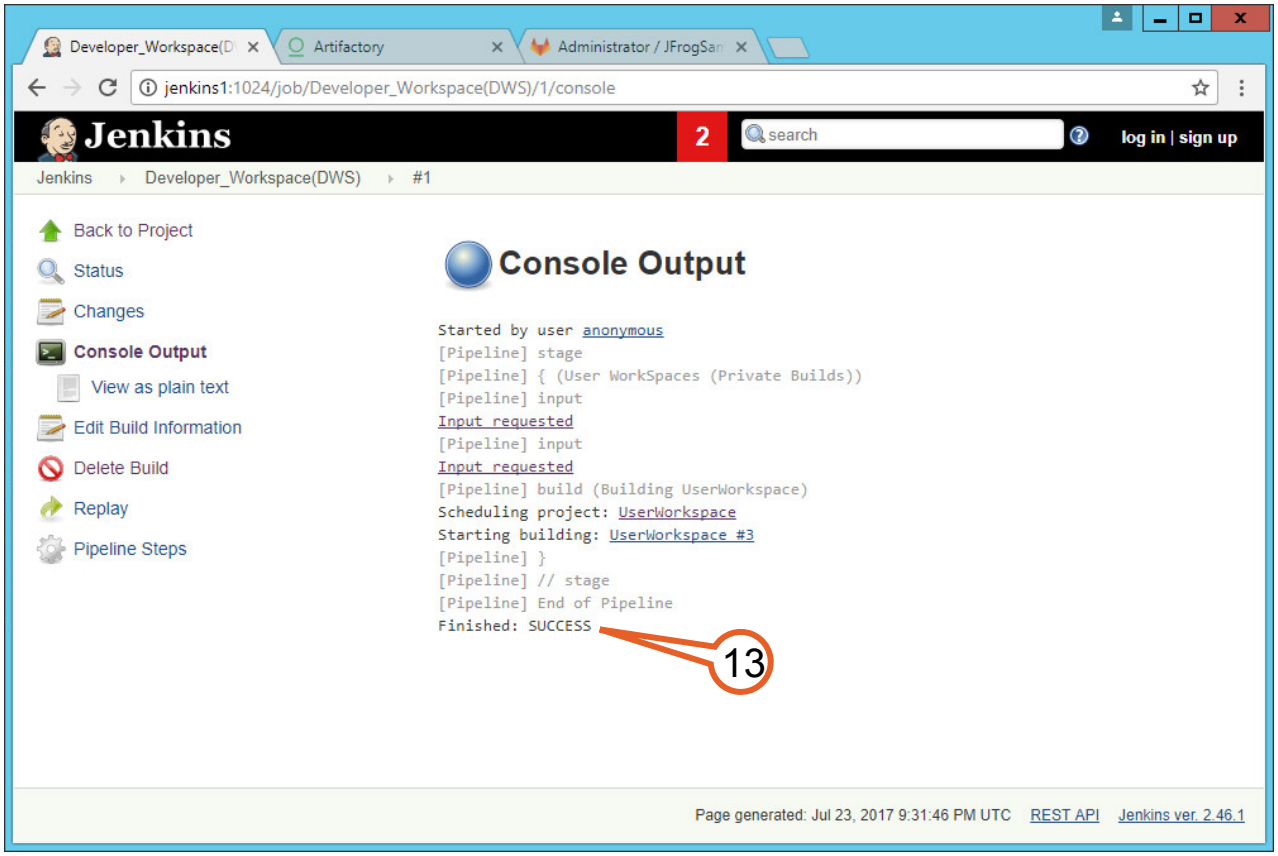


Figure 5-44:

Notice that once you supply all the required inputs the pipeline does not take long to complete, as volume FlexClone only requires a few seconds, and that time is independent of the volume size. Compare this with the time it would take in a non-NetApp environment to check out the source code, download artifacts, and run a full build. On a large, complex project such a process might take many hours, yet with NetApp it would still be seconds.

Now take a closer look at the developer workspace you just created from the container level.

14. Switch to your PuTTY session to the jenkins1 host, and list the running containers.

```
[root@jenkins1 ~]# docker ps
CONTAINER ID        IMAGE                                                 COMMAND
  CREATED             STATUS                  PORTS
                    NAMES
af4cf55d16d5        devopsnetapponaws/netapp-jenkins_slave:lod        "/bin/sh -c 'exec ..."
  About a minute ago   Up About a minute
                    Dev1_JFrog_2017_1
6d4c59bbfd54        devopsnetapponaws/netapp-jenkins_slave:lod        "/bin/sh -c 'exec ..."
  2 hours ago         Up 2 hours
                    JFrog_2017_1
3b9fe3ba4585        devopsnetapponaws/netapp-jenkins-plugin-2.0:lod   "/bin/tini -- /usr..."
  6 hours ago         Up 6 hours              8080/tcp, 50000/tcp
                    jenkins.1.jdf6vqyf2k1ggenjjamju1oqo
22edf3dd1cdc        devopsnetapponaws/netapp-jenkins_gitlab           "/usr/bin/startcon..."
   2 months ago        Up 6 hours (healthy)   0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp,
 0.0.0.0:10022->22/tcp   JFrog_OSS_Repo
[root@jenkins1 ~]#
```

There is now a container named “Dev1_JFrog_2017_1” listed here that was not present when you ran this command in the preceding exercise. This is the new container that the Developer_Workspace(DWS) pipeline just created for you.

15. Open a bash session inside the Dev1_JFrog_2017_1 container.

```
[root@jenkins1 ~]# docker exec -it Dev1_JFrog_2017_1 /bin/bash
root@af4cf55d16d5:/#
```

16. Display the container's list of mounted volumes.

```
root@af4cf55d16d5:/# df
Filesystem
     1K-blocks    Used Available Use% Mounted on
/dev/mapper/docker-253:0-68215161-
da807898255e4d290117e4127603992c99a1ea57eee48e5daf94511789b2c4ed  10474496  872756   9601740
   9% /
tmpfs
         941996       0    941996   0% /dev
tmpfs
         941996       0    941996   0% /sys/fs/cgroup
192.168.0.132:/Dev1_JFrog_2017_1
        2166208   56704   2109504   3% /workspace/Dev1_JFrog_2017_1
/dev/mapper/rhel-root
       37202180 8996620  28205560  25% /etc/hosts
shm
          65536       0     65536   0% /dev/shm
tmpfs
         941996       0    941996   0% /sys/firmware
root@af4cf55d16d5:/#
```

The /workpace/Dev1_JFrog_2017_1 volume is this container persistent volume that was FlexCloned from the CI build's persistent volume. Observe that there is used space on this volume.

Before you continue examining the container, take a quick detour to verify that the Dev1_JFrog_2017_1 volume is actually a FlexClone of the CI master volume.

17. On the taskbar for Jumphost, right click on the **PuTTY** icon.
18. Select **PuTTY** from the context menu.

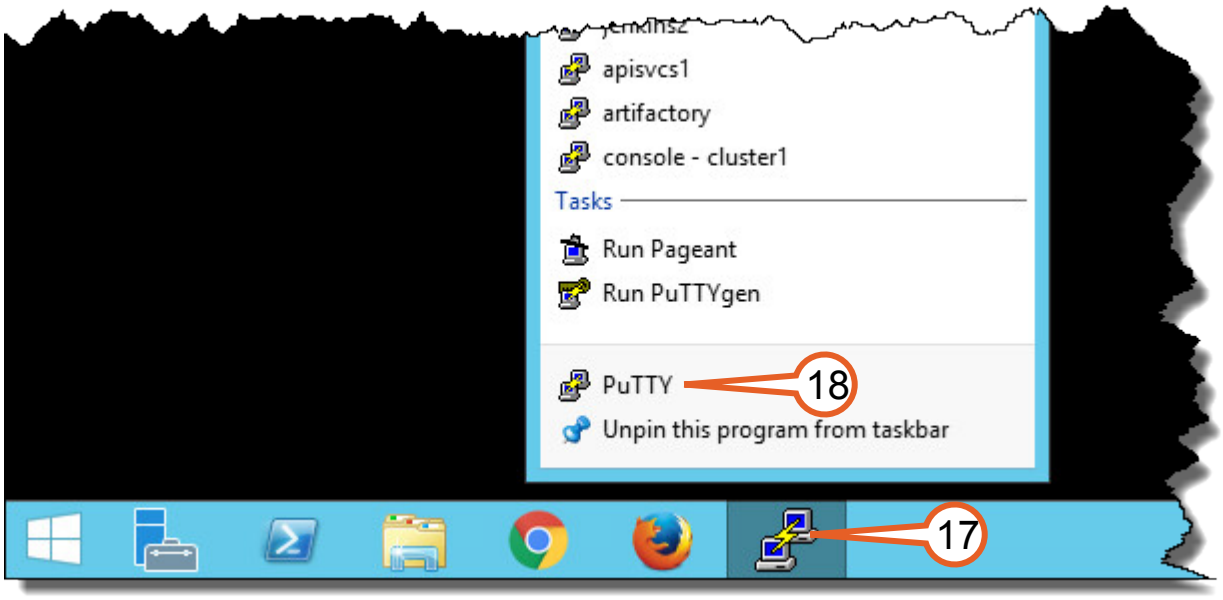


Figure 5-45:

19. In the saved sessions list, double click on **cluster1**.
20. Log in as the user **admin** with the password **Netapp1!**.

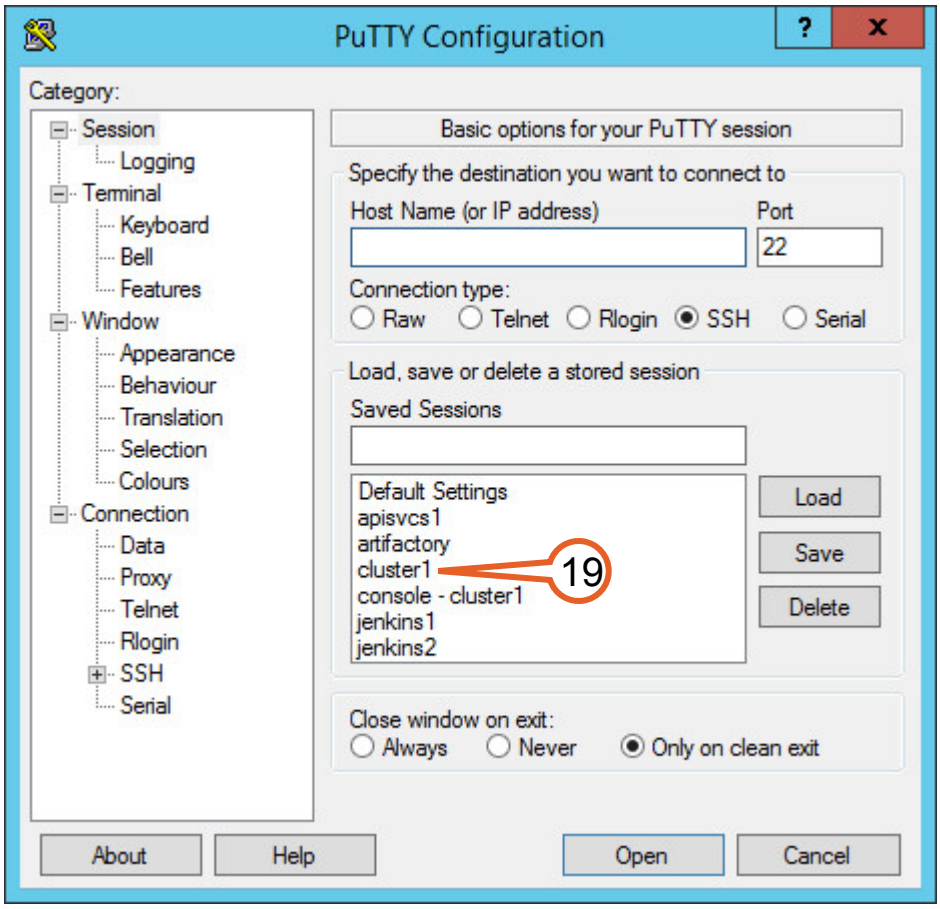


Figure 5-46:

21. Display a list of the volumes on the controller.

```
cluster1::> volume show
Vserver   Volume       Aggregate    State      Type       Size  Available Used%
--------- ------------ ------------ ---------- ---- ---------- ---------- -----
cluster1-01
          vol0         aggr0        online     RW       7.17GB     2.80GB   60%
lab1      Dev1_JFrog_2017_1
                       aggr1        online     RW       2.29GB     2.01GB   12%
lab1      JFrog_2017_1 aggr1        online     RW       2.29GB     2.01GB   12%
lab1      JFrog_OSS_Repo
                       aggr1        online     RW       2.29GB     1.11GB   51%
lab1      JFrog_OSS_Repo_config
                       aggr1        online     RW       5.74GB     5.16GB   10%
lab1      JFrog_OSS_Repo_logs
                       aggr1        online     RW       5.74GB     5.15GB   10%
lab1      artifactory  aggr1        online     RW          1GB    936.2MB    8%
lab1      jenkins_home aggr1        online     RW         10GB     9.21GB    7%
lab1      lab1_root    aggr1        online     RW         20MB    18.18MB    9%
9 entries were displayed.

cluster1::>
```

You should recognize most of these volume names from earlier exercises. Of particular interest here is the JFrog_2017_1 volume, which is the persistent volume for the CI master build container, and the Dev1_JFrog_2017_1 volume, which is the persistent volume for the Dev1_JFrog_2017_1 developer workspace container you just created.

22. Display a list of the volume flexclone relationships.

```
cluster1::> volume clone show
                      Parent  Parent        Parent
Vserver FlexClone     Vserver Volume        Snapshot            State     Type
```

```
------- ------------ ------- ------------ -------------------- --------- ----
lab1    Dev1_JFrog_2017_1
                     lab1    JFrog_2017_1  JFrogBuild1-Checkpoint1
                                                        online    RW

cluster1::>
```

The output shows that the Dev1_JFrrog_2017_1 volume is a flexclone of the JFrog_2017_1 volume state as captured in the JFrogBuild1-Checkpoint1 volume snapshot.

Leave the PuTTY session to cluster1 open as you will need it again in the next exercise.

23. Switch back to your PuTTY session to jenkins1, which still has an interactive shell running inside the Dev1_JFrog_2017_1 container.
24. Change directory to the new workspace volume.

```
root@af4cf55d16d5:/# cd /workspace/Dev1_JFrog_2017_1
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1#
```

25. List the files in this directory.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1# ls -l
total 84
-rw-r--r--  1 301 300   469 Jul 22 22:51 CONTRIBUTING.md
-rw-r--r--  1 301 300 11358 Jul 22 22:51 LICENSE
-rw-r--r--  1 301 300    67 Jul 22 22:51 README
drwxr-xr-x  5 301 300  4096 Jul 22 22:51 artifactory-maven-plugin-example
drwxr-xr-x  2 301 300  4096 Jul 22 22:51 bash-example
drwxr-xr-x  3 301 300  4096 Jul 22 22:51 build-info-java-example
drwxr-xr-x  8 301 300  4096 Jul 22 22:51 circleci-example
drwxr-xr-x  5 301 300  4096 Jul 22 22:51 cpp-example
drwxr-xr-x  4 301 300  4096 Jul 22 22:51 gradle-examples
drwxr-xr-x  4 301 300  4096 Jul 22 22:51 ivy-example
drwxr-xr-x 12 301 300  4096 Jul 22 22:51 jenkins-pipeline-examples
drwxr-xr-x  5 301 300  4096 Jul 22 22:51 maven-example
drwxr-xr-x  6 301 300  4096 Jul 22 22:51 maven-example-bintray-info
drwxr-xr-x  8 301 300  4096 Jul 22 22:51 msbuild-example
drwxr-xr-x  3 301 300  4096 Jul 22 22:51 npm-example
drwxr-xr-x  5 301 300  4096 Jul 22 22:51 nuget-example
drwxr-xr-x  4 301 300  4096 Jul 22 22:51 python-example
drwxr-xr-x  2 301 300  4096 Jul 22 22:51 rpm-example
drwxr-xr-x  3 301 300  4096 Jul 22 22:51 sbt-example
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1#
```

As you can see, this looks like the same content you saw in the persistent volume inside the CI container. The files/folders and their contents are indeed the same, but note that the files/folders here all have the uid 301 and gid 300 (the values you supplied as inputs to the pipeline) rather than the root/root values you saw in the CI instance. NetApp ONTAP 9 includes a new feature that allows for the update of UID and GID values while creating the FlexClone copy, all while still keeping the overall FlexClone creation time to just a few seconds. Compare that with “chown -R” operations in non-NetApp environments, which can take many minutes depending on the size of the code base.

Now you will use this new workspace to check out the master branch (the master code base) from the SCM, modify a file, and commit the change back into the SCM again.

26. Set up the environment variables that git needs to identify you for commits.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1# git config --global user.name "Developer1"
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1# git config --global user.email
 "developer@email.com"
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1#
```

27. Check out the master branch. Here you are just notifying git that you will be working on this project.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1# git checkout master
Branch master set up to track remote branch master from origin.
Switched to a new branch 'master'
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1#
```

28. Change to the source tree directory where the file you will modify is stored, and list its contents.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1# cd maven-example/multi2/src/main/java/
artifactory/test
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test#  ls
App.java
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test#
```

29. Launch nano to modify the App.java file.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test# nano App.java
```

This command opens a terminal-based nano text editor window.

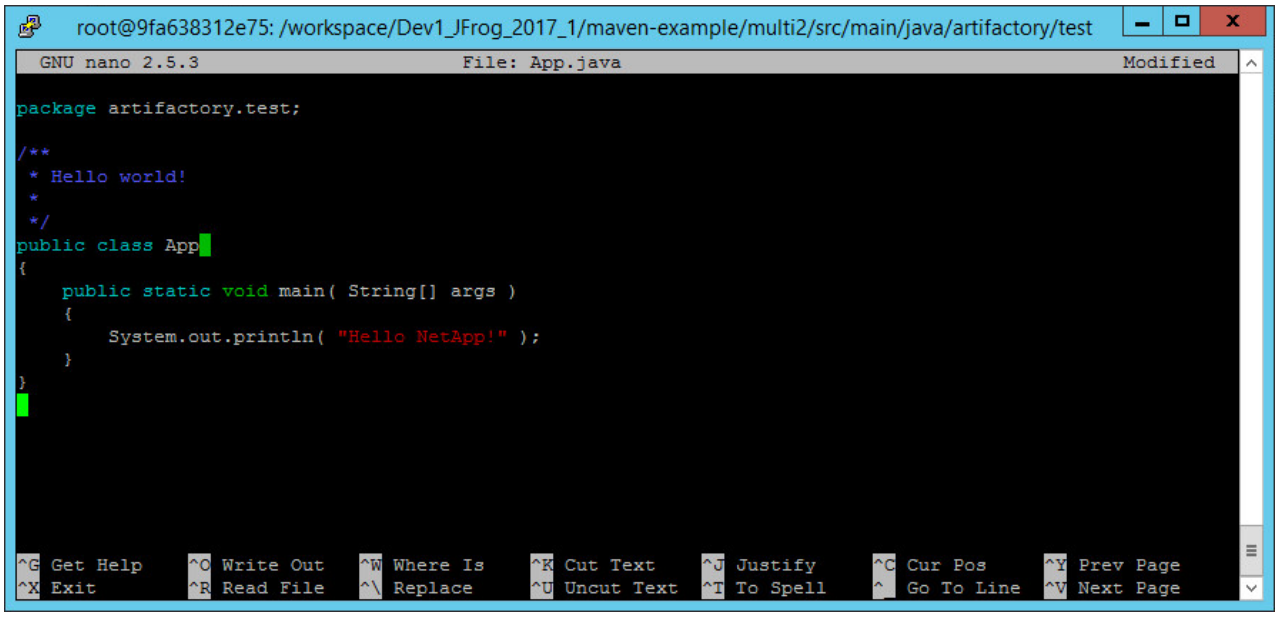


Figure 5-47:

30. Nano is a WYSIWYG (What You See Is What You Get) editor, so use the arrow keys to navigate to the desired location in the file, use the backspace key to delete the word “World”, and insert in its place the word **`NetApp`**.
31. In nano hit **Ctrl-X** to exit.

    Note: If you are a Mac user, you may need to enter **`Cmd-X`** instead of Ctrl-X in order to exit nano.
32. You will be prompted as to whether you want to save the file before exiting. Respond **Y**.

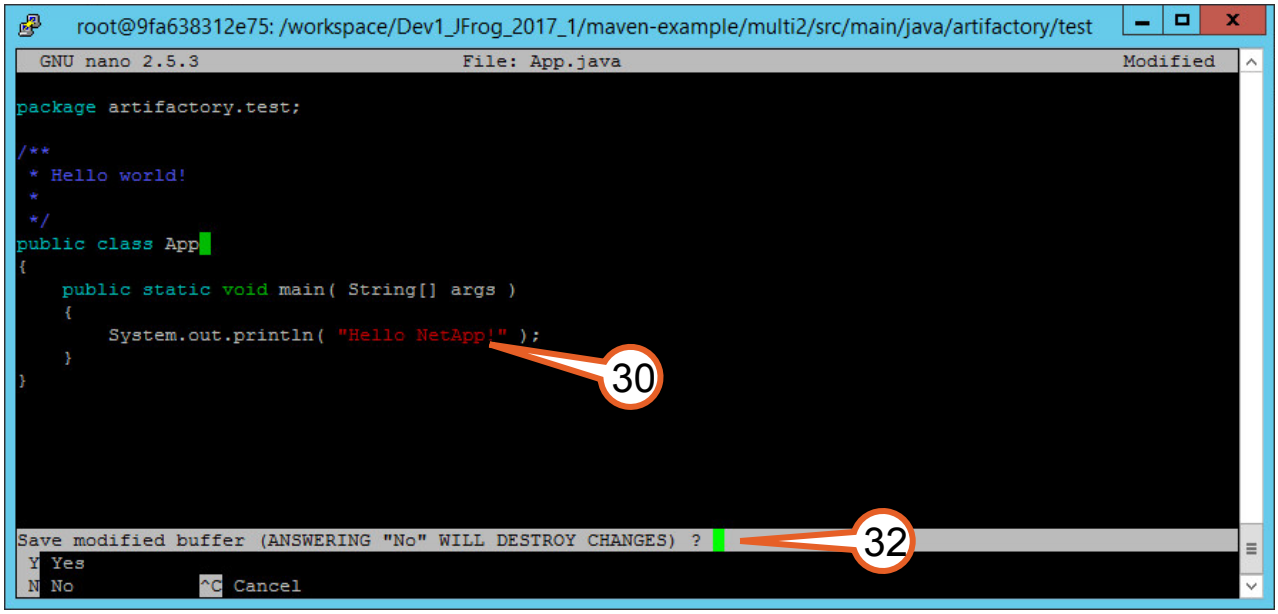


Figure 5-48:

33. When prompted, hit **Enter** to save the file to its original name.

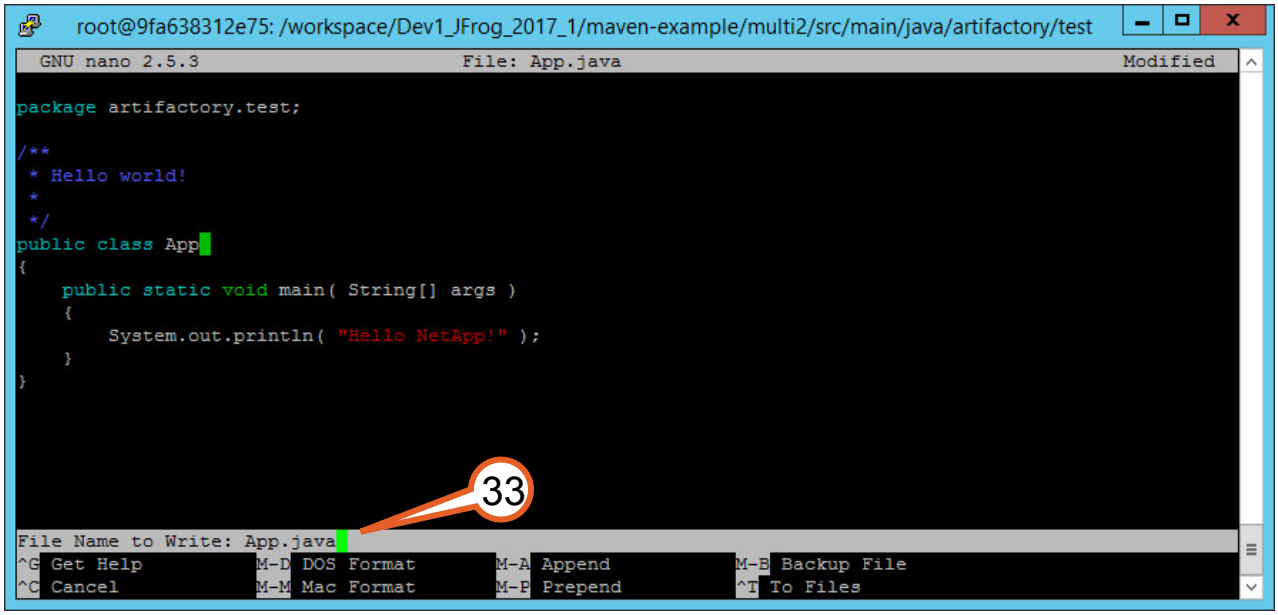


Figure 5-49:

34. Cat the file to confirm the contents.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test# cat App.java
package artifactory.test;

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

```
    }
}
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test#
```

Typically, a software developer would now run a local build and test to verify that the code changes worked, and would perform additional edit/compile/test iterations as necessary in order to stabilize the code changes in preparation for an eventual commit. In this environment these builds would only be incremental, just for the files the developer changed, because the source tree includes a pre-compiled full build. The result is that the developer does not waste time needlessly compiling extra files he did not directly touch.

For expediency's sake you will skip running a local developer build in this lab, and will instead just commit the change to the SCM.

35. Commit the changed file. When prompted for a comment for the commit (which again uses nano), change “Hello World” to **`Hello NetApp`**, and use **Ctrl-X** (or Cmd-X for Mac), **Y**, and **Enter** to save the comment to the default file name.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test# git commit --all
 1 file changed, 1 insertion(+), 1 deletion(-)
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test#
```

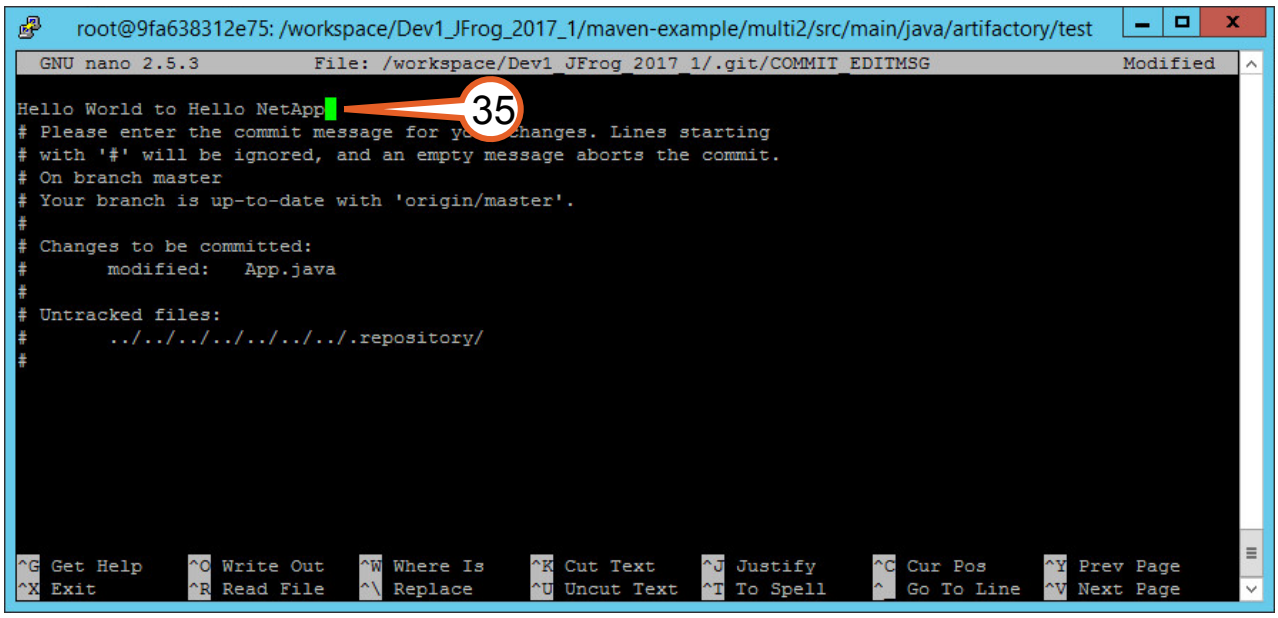


Figure 5-50:

36. Push the changes back to the SCM. When prompted for a password, enter **`Netapp1!`**.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test# git push --all
Password for 'http://root@jenkins1':
Counting objects: 10, done.
Delta compression using up to 2 threads.
Compressing objects:  16% (1/6)   Compressing objects:  33% (2/6)   Compressing objects:
 50% (3/6)   Compressing objects:  66% (4/6)   Compressing objects:  83% (5/6)   Compressing
 objects: 100% (6/6)   Compressing objects: 100% (6/6), done.
Writing objects:  10% (1/10)   Writing objects:  20% (2/10)   Writing objects:  30% (3/10)
 Writing objects:  40% (4/10)   Writing objects:  50% (5/10)   Writing objects:  60% (6/10)
 Writing objects:  70% (7/10)   Writing objects:  80% (8/10)   Writing objects:  90% (9/10)
 Writing objects: 100% (10/10)   Writing objects: 100% (10/10), 759 bytes | 0 bytes/s, done.
Total 10 (delta 2), reused 0 (delta 0)
To http://root@jenkins1/root/JFrogSamples.git
   2aab1f8..2254917  master -> master
```

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test#
```

As soon as the git push completes, the change should be visible in the SCM's browser interface. You'll verify that in a moment.

37. Terminate the bash session to the Dev1_JFrog_2017_1 container.

```
root@af4cf55d16d5:/workspace/Dev1_JFrog_2017_1/maven-example/multi2/src/main/java/
artifactory/test# exit
exit
[root@jenkins1 ~]#
```

38. In Chrome, navigate to the **Gitlab** tab which is pre-configured to open the URL http://jenkins1.
39. You should still be looking at the “JFrogSamples” project page you saw earlier. Refresh the browser window.
40. If you scroll down the page a bit you will see the comment you entered when you commited the change.

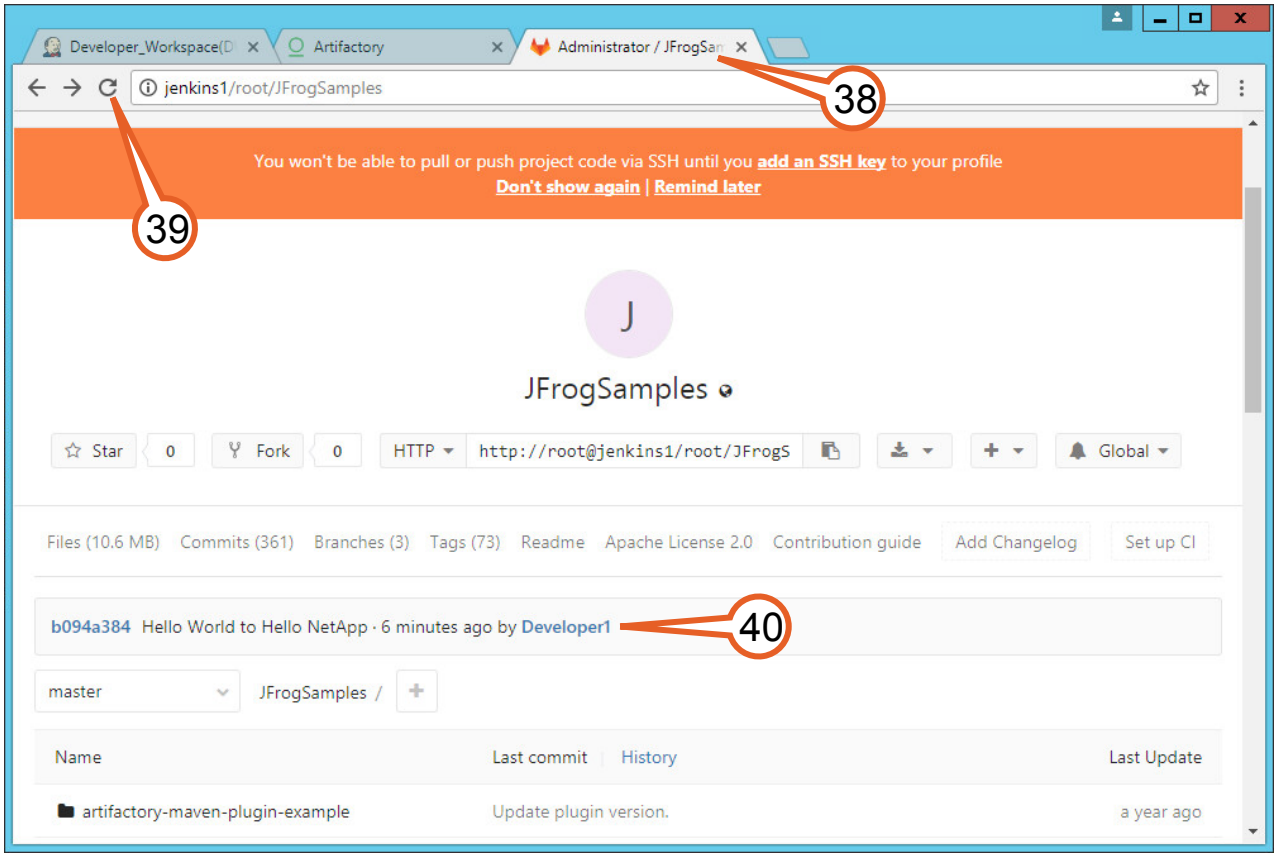


Figure 5-51:

Once you have pushed the code changes into the SCM, Jenkins automatically triggers a CI build so that developers do not have to worry about launching builds manually. This encourages smaller commits, and enhances real-time testing of code changes, both of which lead to shorter development cycles and faster time to market.

Confirm that the automatic build was triggered.

41. In Chrome, navigate to the **Developer_Workspace (DWS)** tab.
42. Click the **Jenkins** graphic at the top of the page.

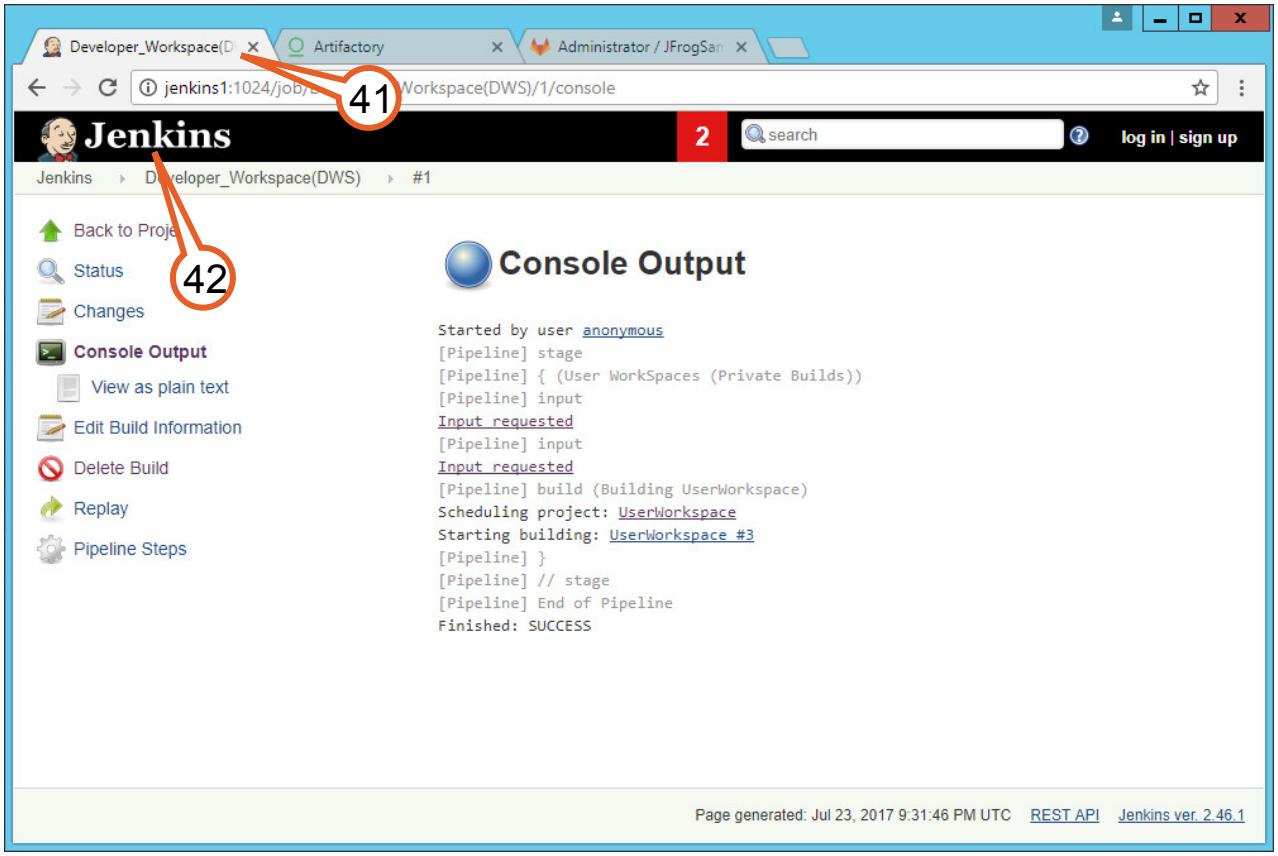
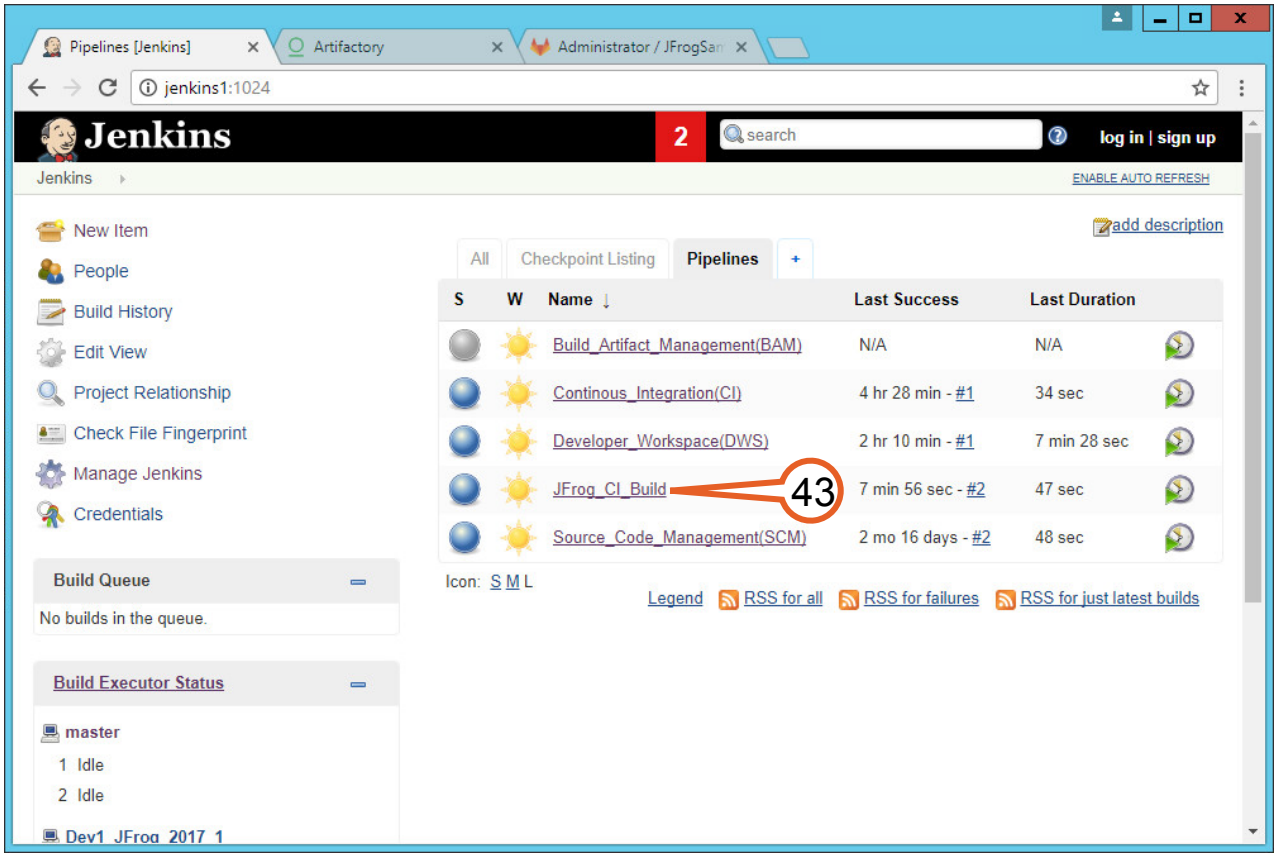


Figure 5-52:

The browser displays the “Pipelines [Jenkins]” page, and the page tab displays this text as well.

43. In the Pipelines list, click on the **JFrog_CI_Build** link.

Figure 5-53:



44. Scroll down to the “Build History” section.

45. Click on the **blue ball** just to the left of the entry for the most recent build.

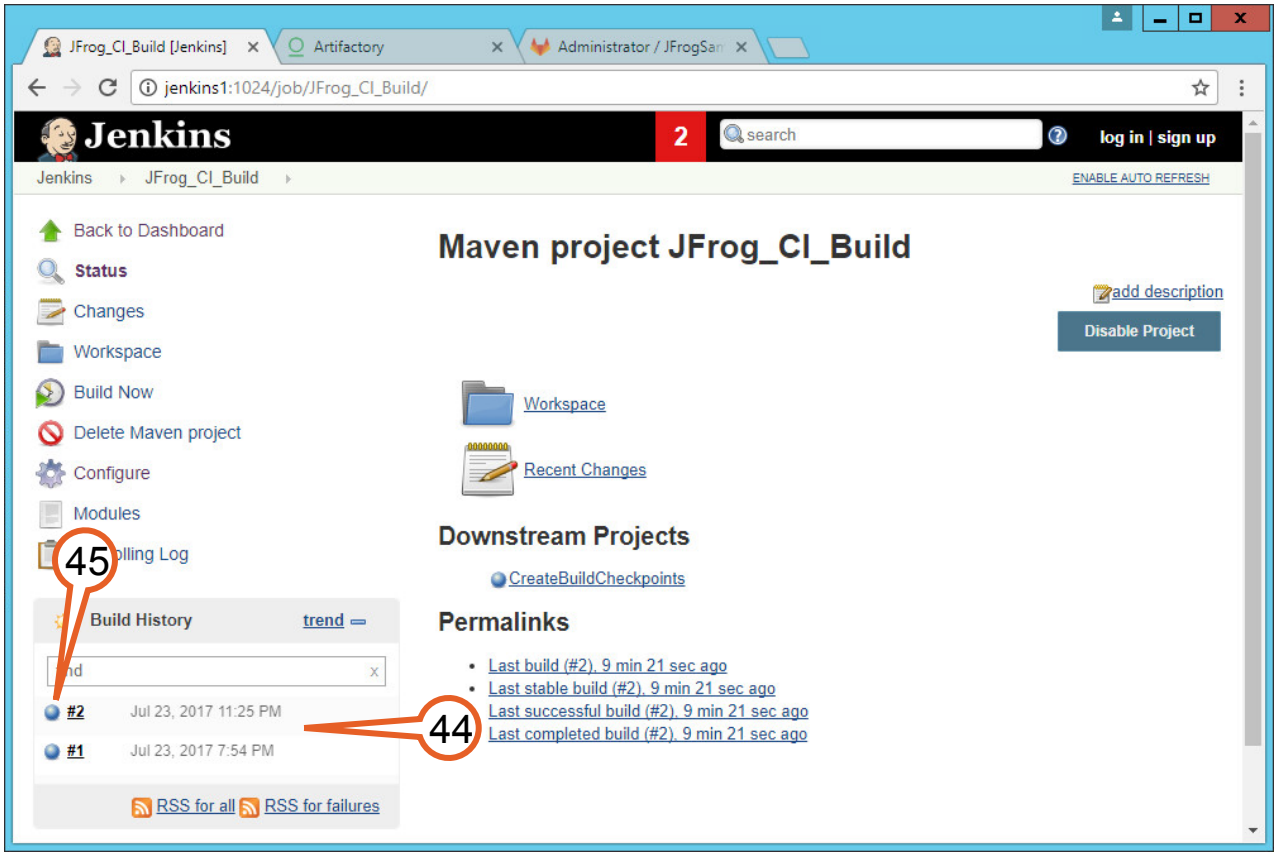


Figure 5-54:

The browser window changes to display the console output for that build.

46. The first line of the console output shows “Started by an SCM change”, verifying that an SCM change triggered this build.

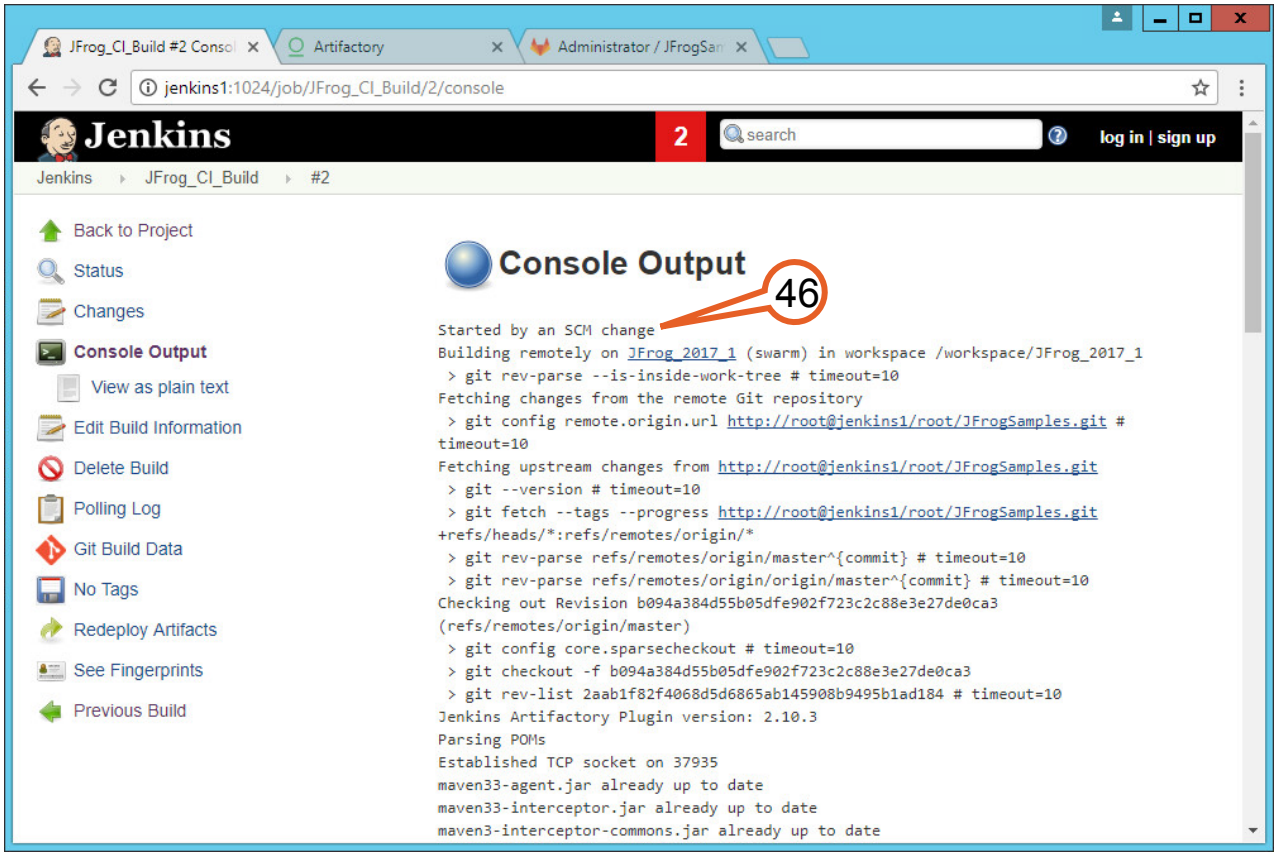


Figure 5-55:

5.7 Build Checkpoints

In this exercise you will more closely examine the checkpoints for the SCM and CI instances under the NetApp-Jenkins framework.

Every successful build in the CI environment generates a checkpoint, which behind the scenes is tied to a NetApp ONTAP snapshot that captures the state of the volume at the time the snapshot is taken. In case a new build fails, the developer can leverage those snapshots to quickly go back to a previous build (without having to rebuild it) and re-run the tests to replicate the bugs, or even to add new features. This can save a lot of time during the development process, but it also reduces storage consumption as ONTAP only has to store the differences between the snapshots. This essentially means the storage is thin provisioned, because snapshots that share a file's state in common can be served from a single copy of that file on the storage.

As part of its zero-touch philosophy, the NetApp-Jenkins framework provides a list of build checkpoints from Jenkins for CI admins and developers without any storage adnmministrator intervention or expertise. In this exercise you will see how to view a list of the available checkpoints.

Whenever a change is pushed to the GitLab [SCM] a checkpoint is created, which under the NetApp-Jenkins framework integration is also a NetApp Snapshot on the SCM Repo. In this lab that SCM Repo volume is named “JFrog_OSS_Repo”.

1. In Chrome, you should still be on the “JFrog_CI_Build” tab from the last exercise.
2. Click the **Jenkins** graphic at the top of the page.

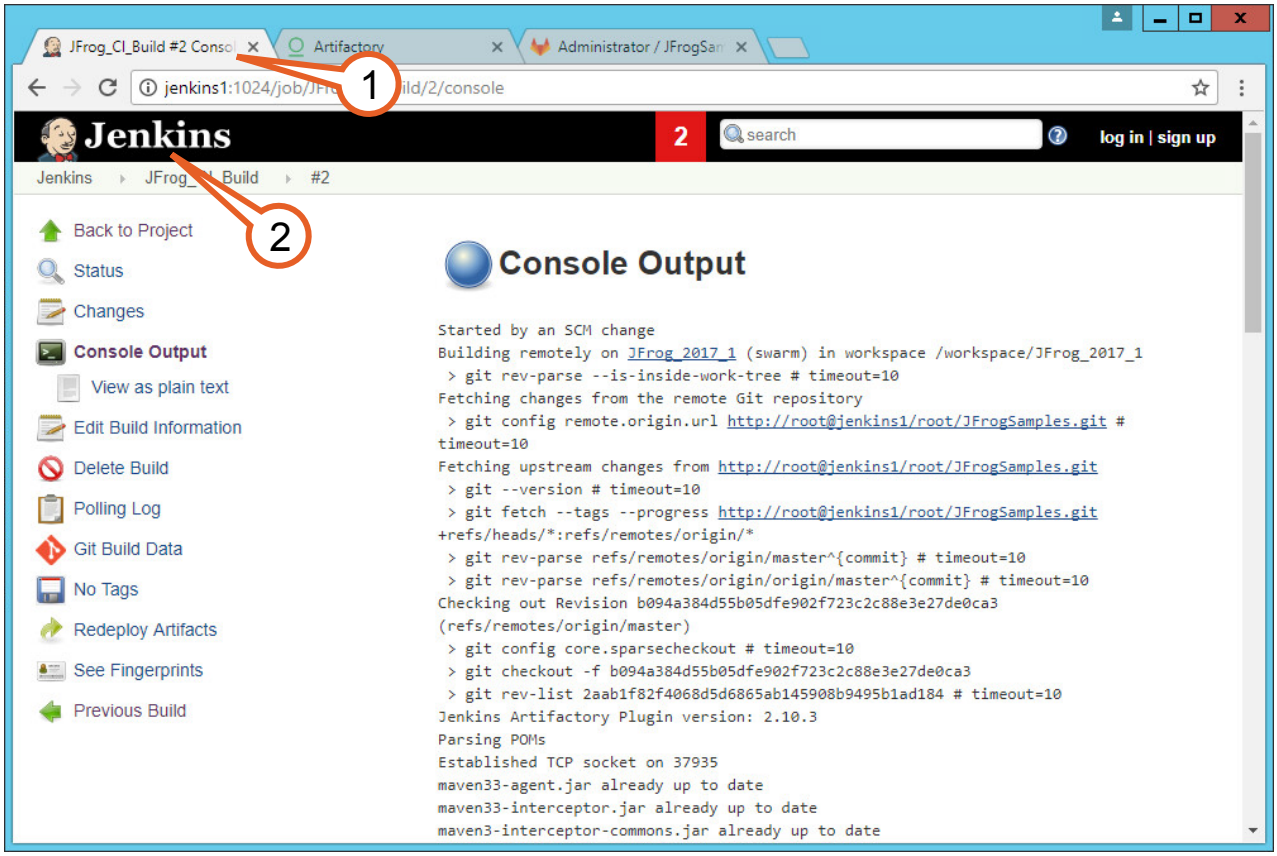


Figure 5-56:

The browser displays the “Pipelines [Jenkins]” page, and the page tab displays this text as well.

3. Click on the **Checkpoint Listing** tab on the right side of the page.

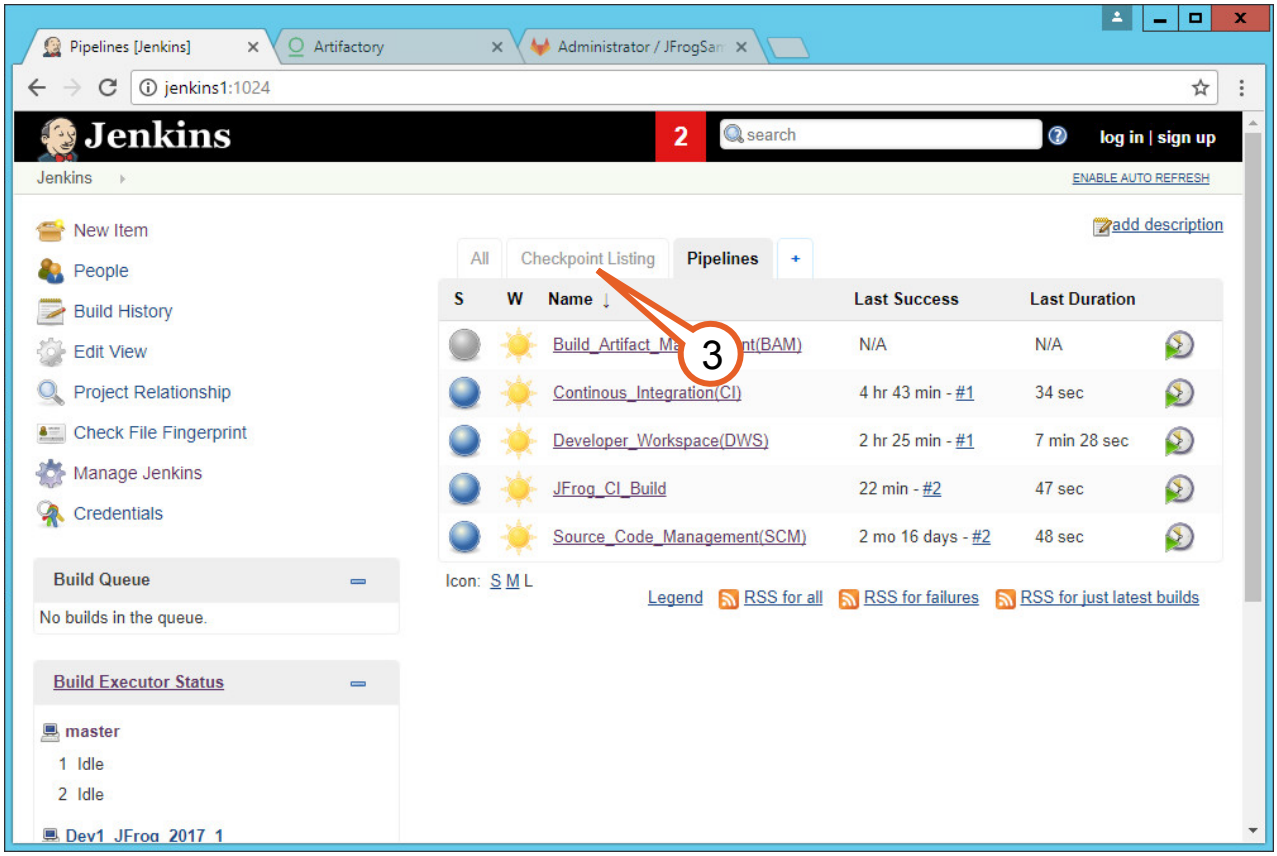


Figure 5-57:

The browser displays the “Checkpoint Listing” page, and the page tab displays this text as well.

4. Click on **List_SCM_Checkpoints**.

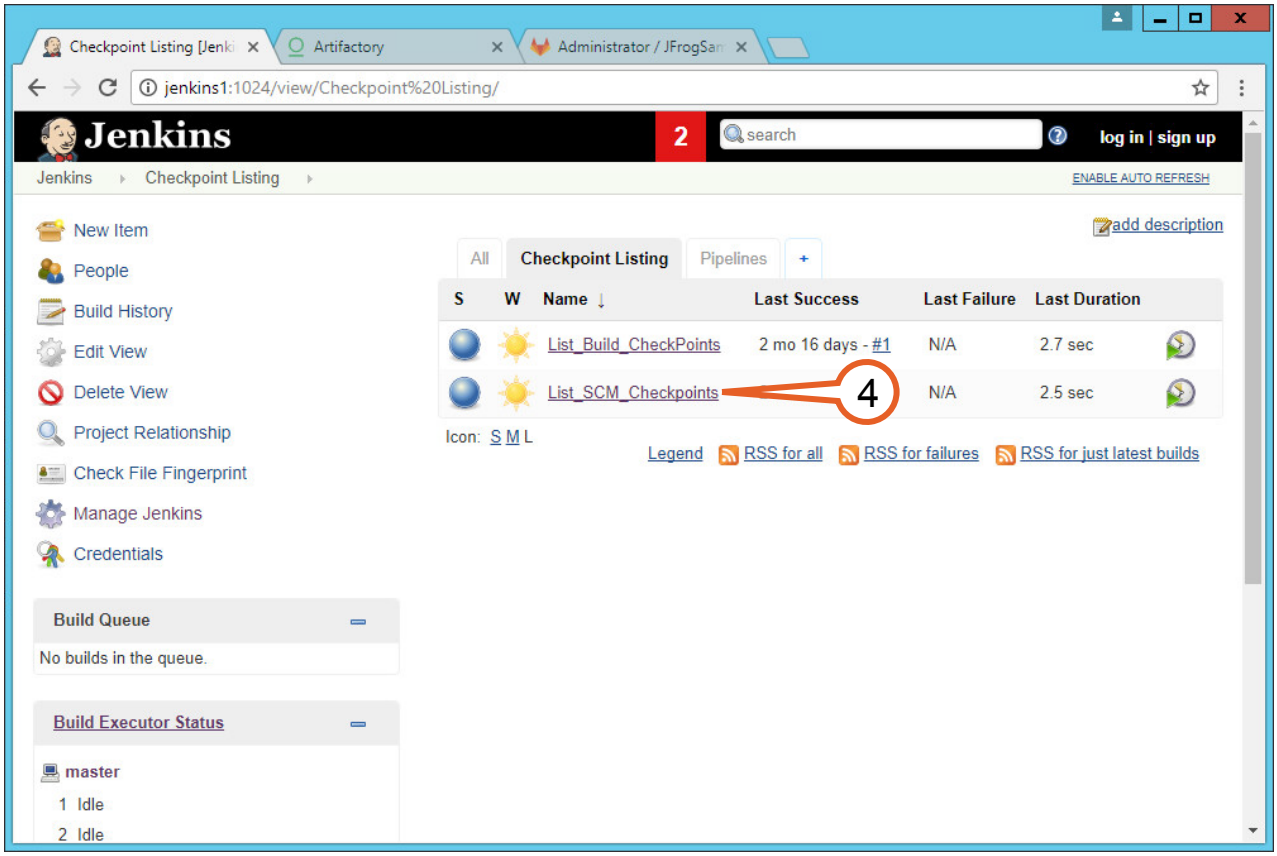


Figure 5-58:

The browser displays the “Project List_SCM_Checkpoints” page, and the page tab displays “List_SCM_Checkpoints [Jenkins]”.

5. The “Build History” section shows all the pipeline executions (AKA builds) for listing the checkpoints. This pipeline is automatically triggered when you push code changes into the SCM, as you did in the preceeding exercise.
6. View the console output for the listed build by clicking the **blue ball** to the left of the build entry.

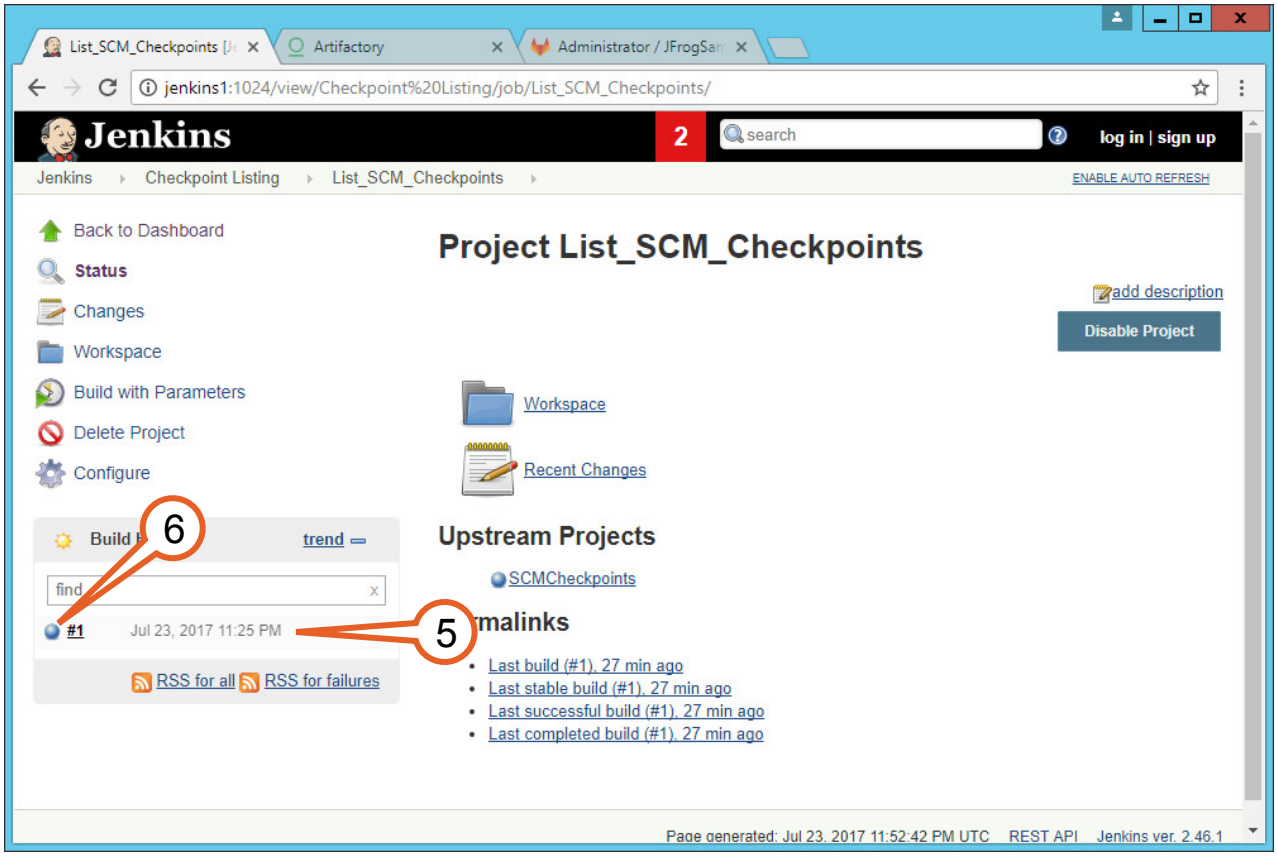


Figure 5-59:

The browser changes to display the console output for this pipeline execution.

7. The console outputs shows the execution of a script that queries the NetApp storage for the snapshots on the JFrog_OSS_Repo volume. This script leverages API services available as part of the Jenkins integration with NetApp's Service Level Manager appliance.

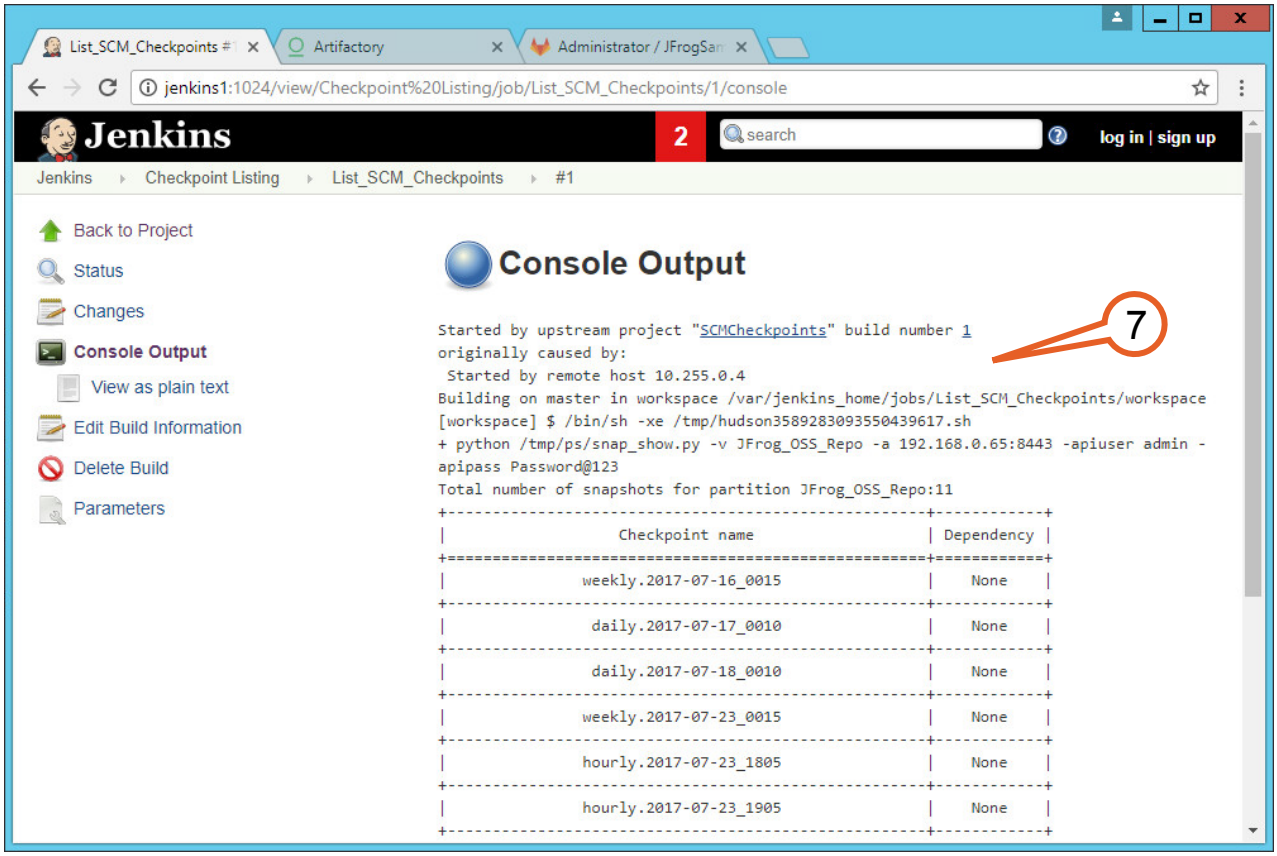


Figure 5-60:

8. Scroll down to see the last checkpoint name in the list. That checkpoint has a different name pattern than the others, and indicates a snapshot related to a checkpoint. The other listed entries are not actual checkpoints as far as the SCM is concerned, but rather periodic snapshots initiated by ONTAP as part of its default data protection policy.

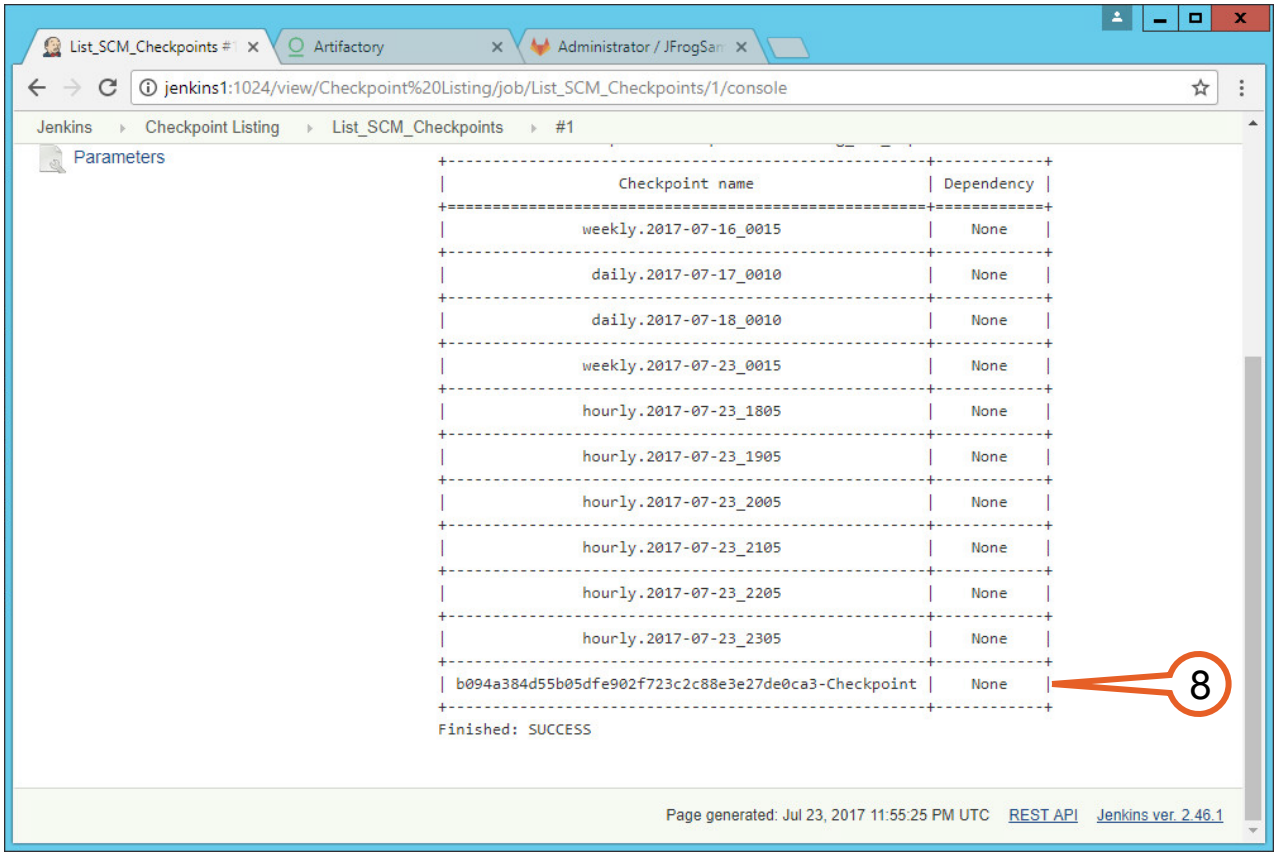


Figure 5-61:

9. Switch to the PuTTY session to cluster1 that you left running from the preceding exercise. If you accidentally closed that session, open it again and log in as the user **`admin`**, with the password **`Netapp1!`**.
10. Display a list of the NetApp snapshots for the JFrog_OSS_Repo volume.

```
cluster1::> snapshot show -volume JFrog_OSS_Repo
                                                                  ---Blocks---
Vserver  Volume   Snapshot                                  Size Total% Used%
-------- -------- ------------------------------------ -------- ------ -----
lab1     JFrog_OSS_Repo
                  weekly.2017-07-16_0015                 148.5MB     6%   40%
                  daily.2017-07-18_0010                  159.6MB     7%   42%
                  weekly.2017-07-23_0015                 119.7MB     5%   35%
                  hourly.2017-07-23_1905                 116.6MB     5%   34%
                  hourly.2017-07-23_2005                 99.86MB     4%   31%
                  hourly.2017-07-23_2105                 106.7MB     5%   32%
                  hourly.2017-07-23_2205                 110.1MB     5%   33%
                  hourly.2017-07-23_2305                 31.18MB     1%   12%
                  b094a384d55b05dfe902f723c2c88e3e27de0ca3-Checkpoint
                                                         91.10MB     4%   29%
                  hourly.2017-07-24_0005                 13.46MB     1%    6%
                  daily.2017-07-24_0010                  30.07MB     1%   12%
11 entries were displayed.

cluster1::>
```

Compare the snapshot list to the list of SCM checkpoints. They match!

11. In Chrome, still on the Jenkins tab, navigate back to the **Checkpoint Listing** tab. This is most easily done by clicking on the **Checkpoint Listing** control at the top of the page.

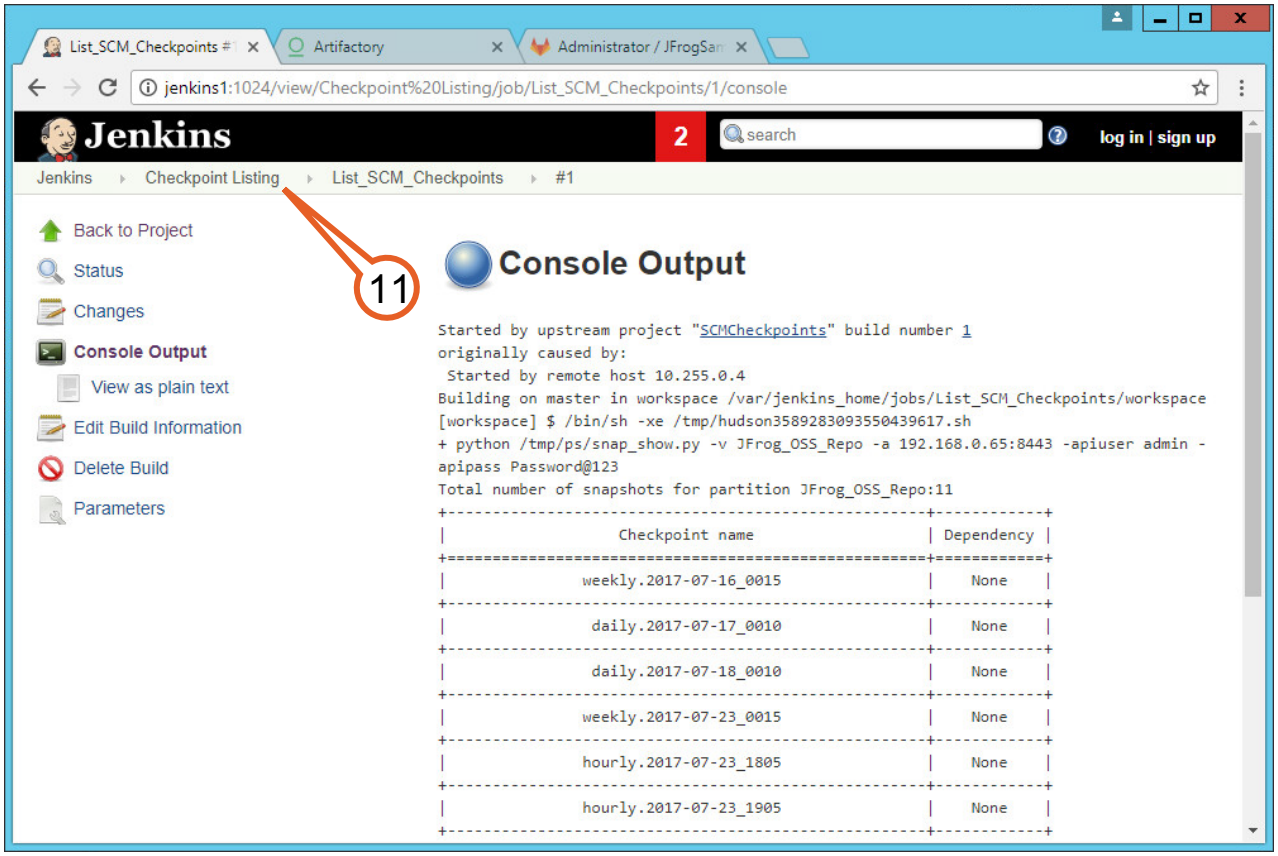


Figure 5-62:

12. Under the “Checkpoint Listing” tab, click on **List_Build_Checkpoints**.

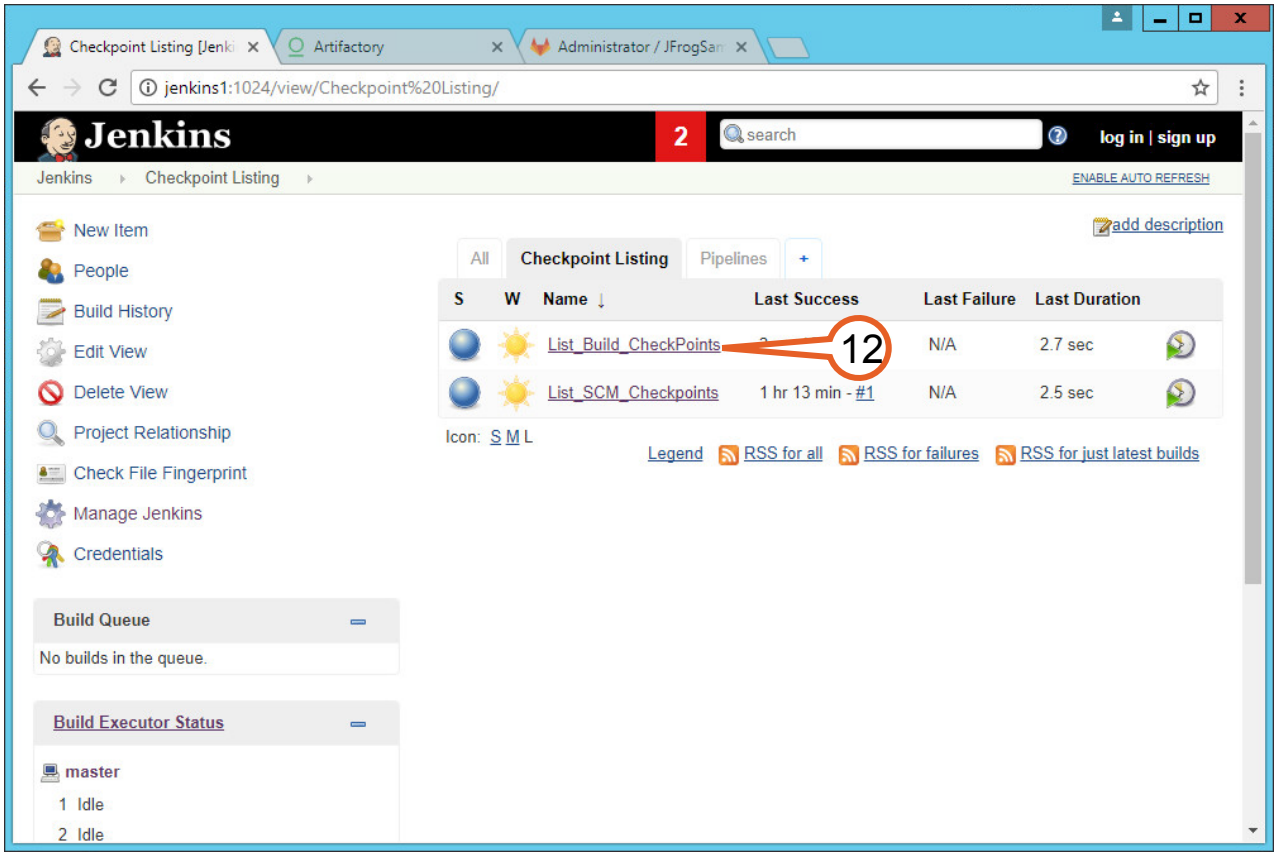


Figure 5-63:

13. Once again, the “Build History” section lists the executions for this pipeline. Click on the **blue ball** to the left of the most recent execution entry.

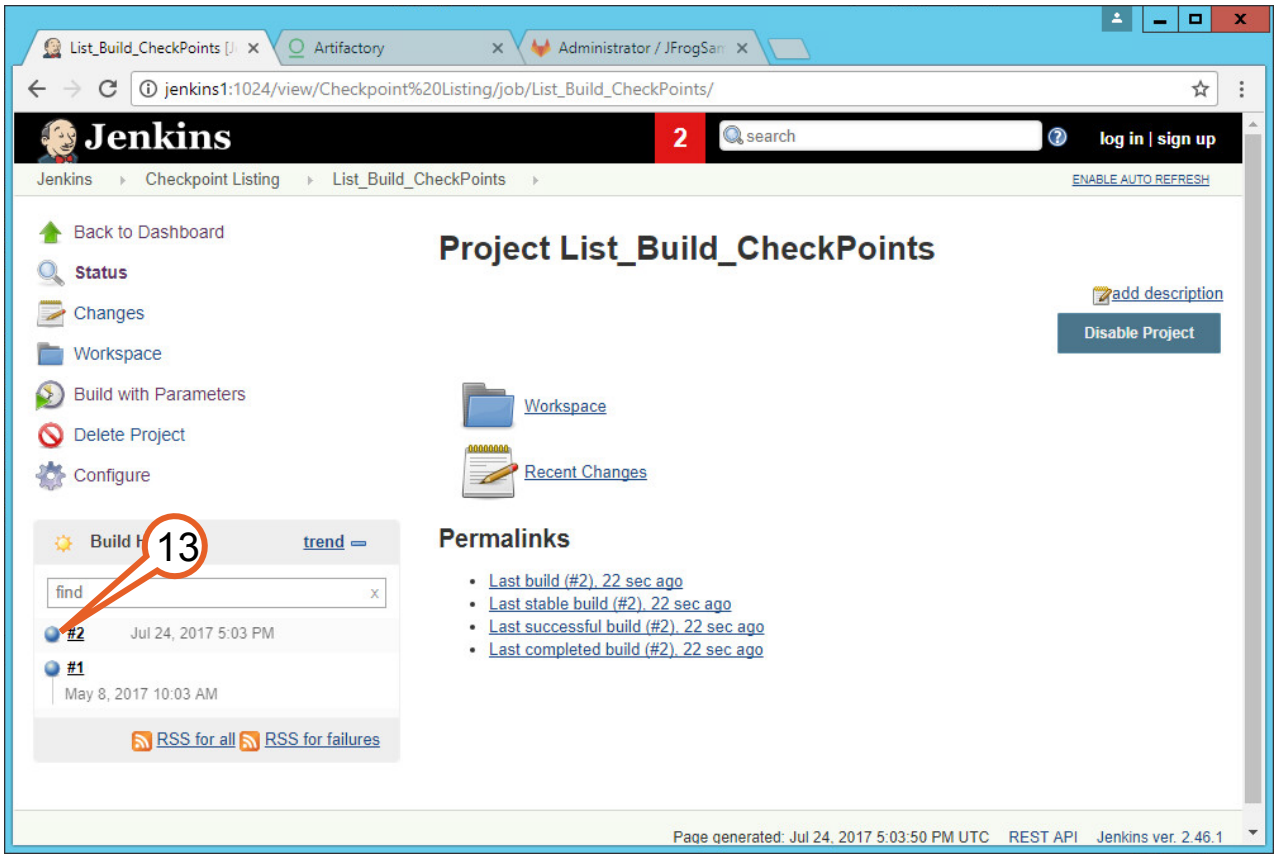


Figure 5-64:

The Console Output window for this execution opens and displays the results.

14. The output shows that the pipeline used the NetApp integration to query the snapshots for the JFrog_2017_1 volume.
15. The checkpoint list shows two snapshots that contain “Checkpoint” in the name. These are the snapshots that correspond to the two builds you saw in the Build History list.
16. The other checkpoints correspond to the snapshots that ONTAP periodically automatically creates as part of its default data protection policy.

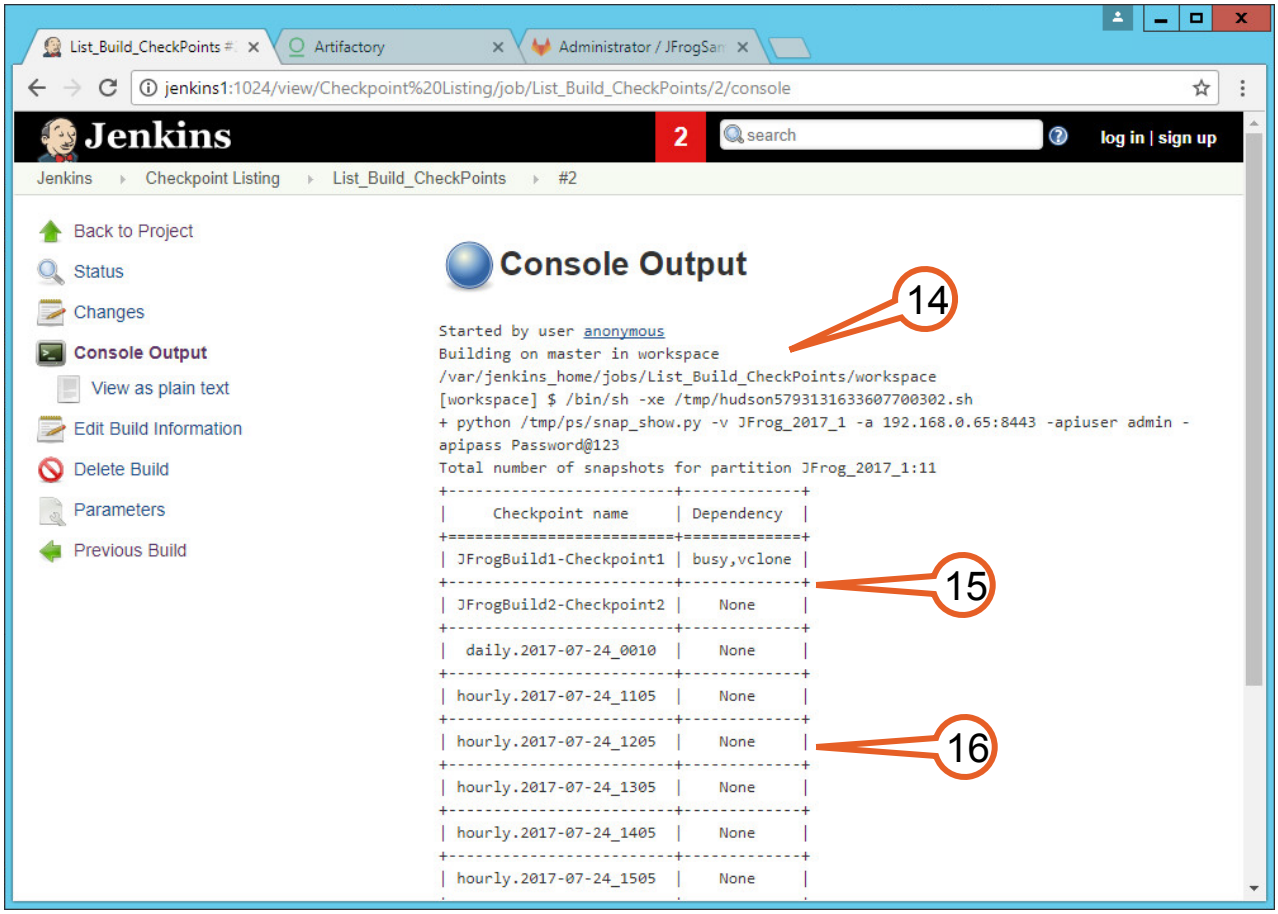


Figure 5-65:

17. In the PuTTY session to cluster1 that you used earlier in this exercise, display a list of the snapshots for the JFrog_2017_1 volume.

```
cluster1::> snapshot show -volume JFrog_2017_1
                                                                  ---Blocks---
Vserver  Volume   Snapshot                                  Size Total% Used%
-------- -------- ------------------------------------ -------- ------ -----
lab1     JFrog_2017_1
                  JFrogBuild1-Checkpoint1                  776KB     0%    1%
                  JFrogBuild2-Checkpoint2                   80KB     0%    0%
                  daily.2017-07-24_0010                  12.14MB     1%   19%
                  hourly.2017-07-24_1205                    88KB     0%    0%
                  hourly.2017-07-24_1305                    88KB     0%    0%
                  hourly.2017-07-24_1405                    88KB     0%    0%
                  hourly.2017-07-24_1505                    88KB     0%    0%
                  hourly.2017-07-24_1605                  1.90MB     0%    3%
                  hourly.2017-07-24_1705                    80KB     0%    0%
9 entries were displayed.

cluster1::>
```

Note how they match the snapshot list you saw in the preceding step.

6 References

The following references were used to create this lab guide.

- TR-4547: Continuous Integration (CI) Pipeline with CloudBees Enterprise Jenkins and ONTAP 9

  http://www.netapp.com/us/media/tr-4547.pdf
- Download the NetApp-Jenkins Framework:

  https://github.com/NetApp/Jenkins-Plugin

7 Version History

Version	Date	Document Version History
1.0	Aug 2017	Initial Release

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

NetApp provides no representations or warranties regarding the accuracy, reliability, or serviceability of any information or recommendations provided in this publication, or with respect to any results that may be obtained by the use of the information or observance of any recommendations provided herein. The information in this document is distributed AS IS, and the use of this information or the implementation of any recommendations or techniques herein is a customer's responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. This document and the information contained herein may be used solely in connection with the NetApp products discussed in this document.

Go further, faster®

www.netapp.com