Lab - Build a CI/CD Pipeline Using Jenkins

***** MODIFIED FOR NETLAB+ *****

Objectives

Part 1: Commit the Sample App to Git

Part 2: Modify the Sample App and Push Changes to Git

Part 3: Download and Run the Jenkins Docker Image

Part 4: Configure Jenkins

Part 5: Use Jenkins to Run a Build of Your App

Part 6: Use Jenkins to Test a Build

Part 7: Create a Pipeline in Jenkins

Background / Scenario

In this lab, you will commit the Sample App code to a GitHub repository, modify the code locally, and then commit your changes. You will then install a Docker container that includes the latest version of Jenkins. You will configure Jenkins and then use Jenkins to download and run your Sample App program. Next, you will create a testing job inside Jenkins that will verify your Sample App program successfully runs each time you build it. Finally, you will integrate your Sample App and testing job into a Continuous Integration/Continuous Development pipeline that will verify your Sample App is ready to be deployed each time you change the code.

Required Resources

- DEVASC Virtual Machine

Instructions

Part 1: Commit the Sample App to Git

In this part, you will create a GitHub repository to commit the sample-app files you created in a previous lab. You created a GitHub account in a previous lab. If you have not done so yet, visit github.com now and create an account.

Step 1: Login to GitHub, create a new repository and a personal access token.

a. Login at https://github.com/ with your credentials.

b. Select the "**New repository**" button or click on the "**+**" icon in the upper right corner and select "**New repository**".

c. Create a repository using the following information:

Repository name: **sample-app**

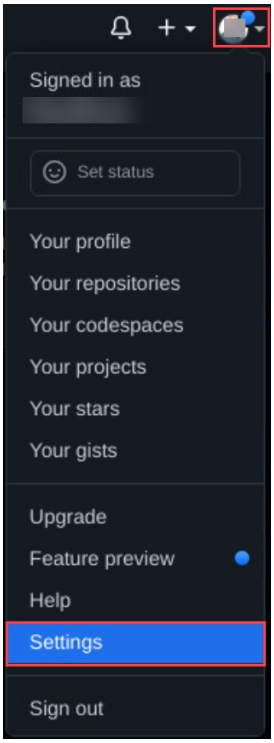
Description: **Explore CI/CD with GitHub and Jenkins**

Public/Private: **Private**

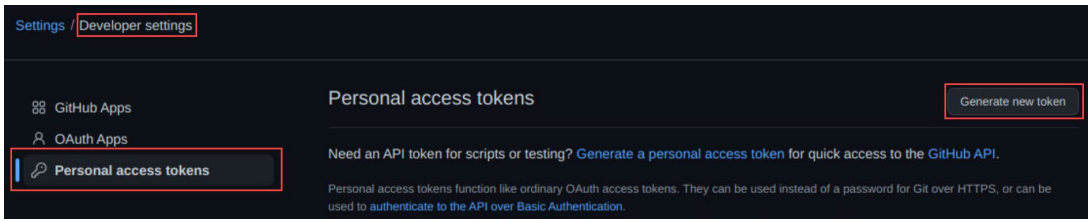
d. Select: **Create repository**

e. Generate a **personal access token** to use for the password, when connecting to your GitHub project.

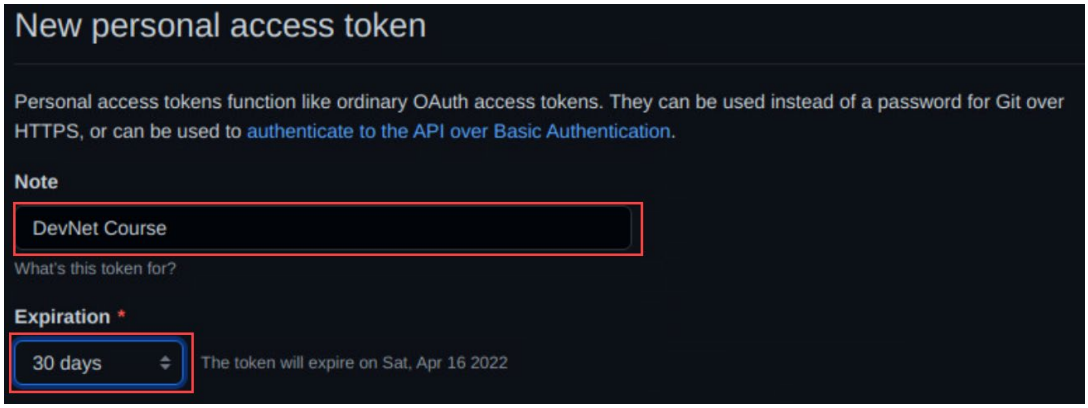
f. In the upper-right, click your **profile photo**, then click **settings**



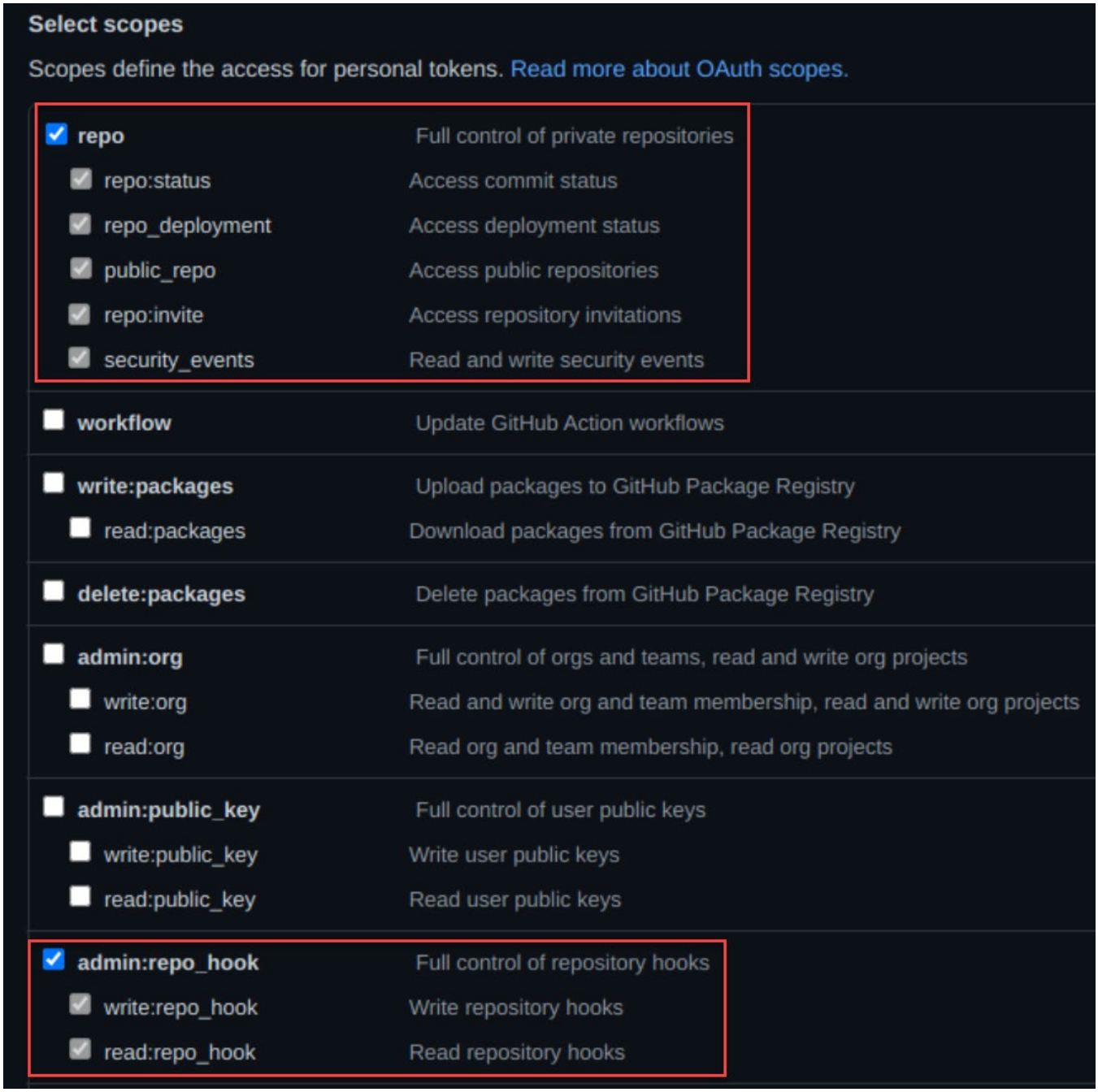
g. In the left sidebar, click **<> Developer settings**, then click **Personal access tokens**.



h. Give your token a descriptive **note** and set the **expiration** timer



i. Select the scopes, or permissions, you'd like to grant this token. To use your token to access repositories from the command line, select **repo** and **admin:repo_hook**.



j. Click **Generate token**.

k. **Copy** your token and store securely as you require it in **Step 4d** and you can't view the token again once you navigate away from the page, so if you lose it you will need to regenerate a new token.

Step 2: Initialize a directory as the Git repository.

Open a terminal window.

You will use the sample-app files you created in a previous lab. However, those files are also stored for your convenience in the **/labs/devnet-src/jenkins/sample-app** directory. Navigate to the **jenkins/sample-app** directory and initialize it as a Git repository.

```
devasc@labvm:~$ cd labs/devnet-src/jenkins/sample-app/
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git init
Initialized empty Git repository in /home/devasc/labs/devnet-src/jenkins/sample-app/.git/
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Step 3: Point Git repository to GitHub repository.

Use the **git remote add** command to add a Git URL with a remote alias of "origin" and points to the newly created repository on GitHub. Using the URL of the Git repository you created in Step 1, you should only need to replace the ***github-username*** in the following command with your GitHub username.

Note: Your GitHub username is case-sensitive.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git remote add origin https://github.com/github-username/sample-app.git
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Step 4: Stage, commit, and push the sample-app files to the GitHub repository.

a. Use the **git add** command to stage the files in the **jenkins/sample-app** directory. Use the asterisk (*) argument to stage all files in the current directory.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git add *
```

b. Use the git status command to see the files and directories that are staged and ready to be committed to your GitHub repository.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   sample-app.sh
        new file:   sample_app.py
        new file:   static/style.css
        new file:   templates/index.html

devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

c. Use the **git commit** command to commit the staged files and start tracking changes. Add a message of your choice or use the one provided here.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git commit -m "Committing sample-app files."
[master 4030ab6] Committing sample-app files
 4 files changed, 46 insertions(+)
```

```
 create mode 100644 sample-app.sh
 create mode 100644 sample_app.py
 create mode 100644 static/style.css
 create mode 100644 templates/index.html
```

d. Use the **git push** command to push your local sample-app files to your GitHub repository.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git push origin master
Username for 'https://github.com': github-username
Password for 'https://github-username@github.com': personal access token
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 2 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (8/8), 1.02 KiB | 1.05 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To https://github.com/github-username/sample-app.git
   d0ee14a..4030ab6  master -> master
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Note: If after entering your username and personal access token, you get a fatal error stating repository is not found, you most likely submitted an incorrect URL. You will need to reverse your **git add** command with the **git remote rm origin** command.

Part 2: Modify the Sample App and Push Changes to Git

In Part 4, you will install a Jenkins Docker image that will use port 8080. Recall that your sample-app files are also specifying port 8080. The Flask server and Jenkins server cannot both use 8080 at the same time.

In this part, you will change the port number used by the sample-app files, run the sample-app again to verify it works on the new port, and then push your changes to your GitHub repository.

Step 1: Open the sample-app files.

Make sure you are still in the **~/labs/devnet-src/jenkins/sample-app** directory as these are the files that are associated with your GitHub repository. Open both **sample_app.py** and **sample-app.sh** for editing.

Step 2: Edit the sample-app files.

a. In sample_app.py, change the one instance of port 8080 to 5050 as shown below.

```
from flask import Flask
from flask import request
from flask import render_template

sample = Flask(__name__)

@sample.route("/")
def main():
    return render_template("index.html")

if __name__ == "__main__":
    sample.run(host="0.0.0.0", port=5050)
```

b. In sample-app.sh, change the three instances of port 8080 to 5050 as shown below.

```
#!/bin/bash

mkdir tempdir
mkdir tempdir/templates
mkdir tempdir/static

cp sample_app.py tempdir/.
cp -r templates/* tempdir/templates/.
cp -r static/* tempdir/static/.

echo "FROM python" >> tempdir/Dockerfile
echo "RUN pip install flask" >> tempdir/Dockerfile
echo "COPY  ./static /home/myapp/static/" >> tempdir/Dockerfile
echo "COPY  ./templates /home/myapp/templates/" >> tempdir/Dockerfile
echo "COPY  sample_app.py /home/myapp/" >> tempdir/Dockerfile
echo "EXPOSE 5050" >> tempdir/Dockerfile
echo "CMD python /home/myapp/sample_app.py" >> tempdir/Dockerfile

cd tempdir

docker build -t sampleapp .

docker run -t -d -p 5050:5050 --name samplerunning sampleapp
docker ps -a
```

Step 3: Build and verify the sample-app.

a. Enter the **bash** command to build your app using the new port 5050.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ bash ./sample-app.sh
Sending build context to Docker daemon  6.144kB
Step 1/7 : FROM python
 ---> 4f7cd4269fa9
Step 2/7 : RUN pip install flask
 ---> Using cache
 ---> 57a74c0dff93
Step 3/7 : COPY  ./static /home/myapp/static/
 ---> Using cache
 ---> e70310436097
Step 4/7 : COPY  ./templates /home/myapp/templates/
 ---> Using cache
 ---> e41ed6d0f933
Step 5/7 : COPY  sample_app.py /home/myapp/
 ---> 0a8d152f78fd
Step 6/7 : EXPOSE 5050
 ---> Running in d68f6bfbcffb
Removing intermediate container d68f6bfbcffb
 ---> 04fa04a1c3d7
Step 7/7 : CMD python /home/myapp/sample_app.py
```

```
 ---> Running in ed48fdbc031b
Removing intermediate container ed48fdbc031b
 ---> ec9f34fa98fe
Successfully built ec9f34fa98fe
Successfully tagged sampleapp:latest
d957a4094c1781ccd7d86977908f5419a32c05a2a1591943bb44eeb8271c02dc
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS                  PORTS                    NAMES
d957a4094c17        sampleapp           "/bin/sh -c 'python …"   1 second ago
Up Less than a second   0.0.0.0:5050->5050/tcp   samplerunning
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

b. Open a browser tab and navigate to localhost:5050. You should see the message **You are calling me from 172.17.0.1**.

c. Shut down the server when you have verified that it is operating on port 5050.

Step 4: Push your changes to GitHub.

a. Now you are ready to push your changes to your GitHub repository. Enter the following commands.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git add *
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   sample-app.sh
        modified:   sample_app.py
        new file:   tempdir/Dockerfile
        new file:   tempdir/sample_app.py
        new file:   tempdir/static/style.css
        new file:   tempdir/templates/index.html

devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git commit -m "Changed
port from 8080 to 5050."
[master 98d9b2f] Changed port from 8080 to 5050.
 6 files changed, 33 insertions(+), 3 deletions(-)
 create mode 100644 tempdir/Dockerfile
 create mode 100644 tempdir/sample_app.py
 create mode 100644 tempdir/static/style.css
 create mode 100644 tempdir/templates/index.html
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ git push origin master
Username for 'https://github.com': username
Password for 'https://AllJohns@github.com': password
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 2 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 748 bytes | 748.00 KiB/s, done.
Total 6 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/AllJohns/sample-app.git
```

```
   a6b6b83..98d9b2f  master -> master
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

b. You can verify that your GitHub repository is updated by visiting **https://github.com/*github-user*/sample-app**. You should see your new message (Changed port from 8080 to 5050.) and that the latest commit timestamp has been updated.

Part 3: Download and Run the Jenkins Docker Image

In this part, you will download the Jenkins Docker image. You will then start an instance of the image and verify that the Jenkins server is running.

Step 1: Download the Jenkins Docker image.

The Jenkins Docker image is stored here: https://hub.docker.com/r/jenkins/jenkins. At the time of the writing of this lab, that site specifies that you use the **docker pull jenkins/jenkins** command to download the latest Jenkins container. You should get output similar to the following:

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker pull
jenkins/jenkins
Using default tag: latest
latest: Pulling from jenkins/jenkins
3192219afd04: Pulling fs layer
17c160265e75: Pulling fs layer
cc4fe40d0e61: Pulling fs layer
9d647f502a07: Pulling fs layer
d108b8c498aa: Pulling fs layer
1bfe918b8aa5: Pull complete
dafa1a7c0751: Pull complete
650a236d0150: Pull complete
cba44e30780e: Pull complete
52e2f7d12a4d: Pull complete
d642af5920ea: Pull complete
e65796f9919e: Pull complete
9138dabbc5cc: Pull complete
f6289c08656c: Pull complete
73d6b450f95c: Pull complete
a8f96fbec6a5: Pull complete
9b49ca1b4e3f: Pull complete
d9c8f6503715: Pull complete
20fe25b7b8af: Pull complete
Digest: sha256:717dcbe5920753187a20ba43058ffd3d87647fa903d98cde64dda4f4c82c5c48
Status: Downloaded newer image for jenkins/jenkins:latest
docker.io/jenkins/jenkins:latest
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Step 2: Start the Jenkins Docker container.

Enter the following command on **one line**. This command will start the Jenkins Docker container and then allow Docker commands to be executed inside your Jenkins server.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker run --rm -u root -p
8080:8080 -v jenkins-data:/var/jenkins_home -v $(which
docker):/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock -v
"$HOME":/home --name jenkins_server jenkins/jenkins:lts
```

The options used in this **docker run** command are as follows:

- **--rm** - This option automatically removes the Docker container when you stop running it.
- **-u** - This option specifies the user. You want this Docker container to run as root so that all Docker commands entered inside the Jenkins server are allowed.
- **-p** - This option specifies the port the Jenkins server will run on locally.
- **-v** - These options bind mount volumes needed for Jenkins and Docker. The first **-v** specifies where Jenkins data will be stored. The second **-v** specifies where to get Docker so that you can run Docker inside the Docker container that is running the Jenkins server. The third **-v** specifies the PATH variable for the home directory.

Step 3: Verify the Jenkins server is running.

The Jenkins server should now be running. Copy the admin password that displays in the output, as shown in the following.

Do not enter any commands in this server window. If you accidentally stop the Jenkins server, you will need to re-enter the **docker run** command from Step 2 above. After the initial install, the admin password is displayed as shown below.

```
                                    --- OUTPUT OMMITED ---
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password
generated.
Please use the following password to proceed to installation:

77dc402e31324c1b917f230af7bfebf2 <--Your password will be different

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
                                    --- OUTPUT OMMITED ---
2020-05-12 16:34:29.608+0000 [id=19]    INFO    hudson.WebAppMain$3#run: Jenkins is
fully up and running
```

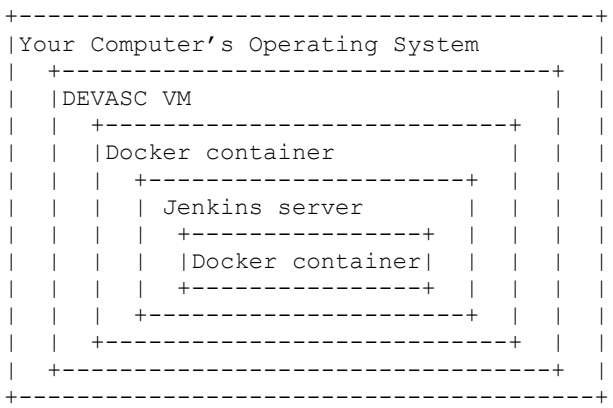
Note: If you lose the password, or it does not display as shown above, or you need to restart the Jenkins sever, you can always retrieve the password by accessing the command line of Jenkins Docker container. Create a second terminal window in VS Code and enter the following commands so that you do not stop the Jenkins server.:

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker exec -it
jenkins_server /bin/bash
root@19d2a847a54e:/# cat /var/jenkins_home/secrets/initialAdminPassword
77dc402e31324c1b917f230af7bfebf2
root@19d2a847a54e:/# exit
exit
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Note: Your container ID (19d2a847a54e highlighted above) and password will be different.

Step 4: Investigate the levels of abstraction currently running on your computer.

The following ASCII diagram shows the levels of abstraction in this Docker-inside-Docker (dind) implementation. This level of complexity is not unusual in today's networks and cloud infrastructures.



Part 4: Configure Jenkins

In this Part, you will complete the initial configuration of the Jenkins server.

Step 1: Open a web browser tab.

Navigate to http://localhost:8080/ and login in with your copied admin password.

Step 2: Install the recommended Jenkins plugins.

Click **Install suggested plugins** and wait for Jenkins to download and install the plugins. In the terminal window, you will see log messages as the installation proceeds. Be sure that you do not close this terminal window. You can open another terminal window for access to the command line. If any plugins fail to download just skip.

Step 3: Skip creating a new admin user.

After the installation finishes, you are presented with the **Create First Admin User** window. For now, click **Continue as admin** at the bottom.

Step 4: Skip creating an instance configuration.

In the **Instance Configuration** window, do not change anything. Click **Save and Finish** at the bottom.

Step 5: Start using Jenkins.

In the next window, click **Start using Jenkins**. You should now be on the main dashboard with a **Welcome to Jenkins!** message.

Part 5: Use Jenkins to Run a Build of Your App

The fundamental unit of Jenkins is the job (also known as a project). You can create jobs that do a variety of tasks including the following:

- Retrieve code from a source code management repository such as GitHub.
- Build an application using a script or build tool.
- Package an application and run it on a server

In this part, you will create a simple Jenkins job that retrieves the latest version of your sample-app from GitHub and runs the build script. In Jenkins, you can then test your app (Part 7) and add it to a development pipeline (Part 8).

Step 1: Create a new job.

a. Click the **create new jobs** link directly below the **Welcome to Jenkins!** message. Alternatively, you can click **New Item** in the menu on the left.

b. In the **Enter an item name** field, fill in the name **BuildAppJob**.

c. Click **Freestyle project** as the job type. In the description, the SCM abbreviation stands for software configuration management, which is a classification of software that is responsible for tracking and controlling changes in software.

d. Scroll to the bottom and click **OK**.

Step 2: Configure the Jenkins BuildAppJob.

You are now in the configuration window where you can enter details about your job. The tabs across the top are just shortcuts to the sections below. Click through the tabs to explore the options you can configure. For this simple job, you only need to add a few configuration details.

a. Click the **General tab**, add a description for your job. For example, "**My first Jenkins job.**"

b. Click the **Source Code Management** tab and choose the **Git** radio button. In the Repository URL field, add your GitHub repository link for the sample-app taking care to enter your case-sensitive username. Be sure to add the .git extension at the end of your URL. For example:

```
https://github.com/github-username/sample-app.git
```

c. For **Credentials**, click the **Add** button and choose **Jenkins**.

d. In the **Add Credentials** dialog box, fill in your GitHub username and password, and then click **Add**.

Note: You will receive an error message that the connection has failed. This is because you have not selected the credentials yet.

e. In the dropdown for **Credentials** where it currently says **None**, choose the credentials you just configured.

f. After you have added the correct URL and credentials, Jenkins tests access to the repository. You should have no error messages. If you do, verify your URL and credentials. You will need to **Add** them again as there is no way at this point to delete the ones you previously entered.

g. At the top of the **BuildAppJob** configuration window, click the **Build** tab.

h. For the **Add build step** dropdown, choose **Execute shell**.

i. In the **Command** field, enter the command you use to run the build for sample-app.sh script.

```
bash ./sample-app.sh
```

j. Click the **Save** button. You are returned to the Jenkins dashboard with the **BuildAppJob** selected.

Step 3: Have Jenkins build the app.

On the left side, click **Build Now** to start the job. Jenkins will download your Git repository and execute the build command **bash ./sample-app.sh**. Your build should succeed because you have not changed anything in the code since Part 3 when you modified the code.

Step 4: Access the build details.

On the left, in the **Build History** section, click your build number which should be the **#1** unless you have built the app multiple times.

Step 5: View the console output.

On the left, click **Console Output**. You should see output similar to the following. Notice the success messages at the bottom as well as the output from the **docker ps -a** command. Two docker containers are running: one for your sample-app running on local port 5050 and one for Jenkins on local port 8080.

```
Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/BuildAppJob
using credential 0cf684ea-48a1-4e8b-ba24-b2fa1c5aa3df
Cloning the remote Git repository
Cloning repository https://github.com/github-user/sample-app
 > git init /var/jenkins_home/workspace/BuildAppJob # timeout=10
Fetching upstream changes from https://github.com/github-user/sample-app
 > git --version # timeout=10
using GIT_ASKPASS to set credentials
 > git fetch --tags --progress -- https://github.com/github-user/sample-app
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > git config remote.origin.url https://github.com/github-user/sample-app # timeout=10
 > git config --add remote.origin.fetch +refs/heads/*:refs/remotes/origin/* #
timeout=10
 > git config remote.origin.url https://github.com/github-user/sample-app # timeout=10
Fetching upstream changes from https://github.com/github-user/sample-app
using GIT_ASKPASS to set credentials
 > git fetch --tags --progress -- https://github.com/github-user/sample-app
+refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 230ca953ce83b5d6bdb8f99f11829e3a963028bf
(refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 230ca953ce83b5d6bdb8f99f11829e3a963028bf # timeout=10
Commit message: "Changed port numbers from 8080 to 5050"
 > git rev-list --no-walk 230ca953ce83b5d6bdb8f99f11829e3a963028bf # timeout=10
[BuildAppJob] $ /bin/sh -xe /tmp/jenkins1084219378602319752.sh
+ bash ./sample-app.sh
Sending build context to Docker daemon  6.144kB

Step 1/7 : FROM python
 ---> 4f7cd4269fa9
Step 2/7 : RUN pip install flask
 ---> Using cache
 ---> 57a74c0dff93
Step 3/7 : COPY  ./static /home/myapp/static/
 ---> Using cache
 ---> aee4eb712490
Step 4/7 : COPY  ./templates /home/myapp/templates/
 ---> Using cache
 ---> 594cdc822490
Step 5/7 : COPY  sample_app.py /home/myapp/
 ---> Using cache
```

```
 ---> a001df90cf0c
Step 6/7 : EXPOSE 5050
 ---> Using cache
 ---> eae896e0a98c
Step 7/7 : CMD python /home/myapp/sample_app.py
 ---> Using cache
 ---> 272c61fddb45
Successfully built 272c61fddb45
Successfully tagged sampleapp:latest
9c8594e62079c069baf9a88a75c13c8c55a3aeaddde6fd6ef54010953c2d3fbb
CONTAINER ID        IMAGE                  COMMAND                  CREATED
STATUS                  PORTS                                NAMES
9c8594e62079        sampleapp              "/bin/sh -c 'python …"   Less than a second
ago   Up Less than a second   0.0.0.0:5050->5050/tcp               samplerunning
e25f233f9363        jenkins/jenkins:lts    "/sbin/tini -- /usr/…"   29 minutes ago
Up 29 minutes           0.0.0.0:8080->8080/tcp, 50000/tcp   jenkins_server
Finished: SUCCESS
```

Step 6: Open another web browser tab and verify sample app is running.

Type in the local address, **localhost:5050**. You should see the content of your index.html displayed in light steel blue background color with **You are calling me from 172.17.0.1** displayed in as H1.

Part 6: Use Jenkins to Test a Build

In this part, you will create a second job that tests the build to ensure that it is working properly.

Note: You need to stop and remove the **samplerunning** docker container.

```
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker stop samplerunning
samplerunning
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$ docker rm samplerunning
samplerunning
```

Step 1: Start a new job for testing your sample-app.

a. Return to the Jenkins web browser tab and click the **Jenkins** link in the top left corner to return to the main dashboard.

b. Click the **New Item** link to create a new job.

c. In the Enter an item name field, fill in the name **TestAppJob**.

d. Click **Freestyle project** as the job type.

e. Scroll to the bottom and click **OK**.

Step 2: Configure the Jenkins TestAppJob.

a. Add a description for your job. For example, "My second Jenkins test."

b. Leave **Source Code Management** set to **None**.

c. Click the **Build Triggers** tab and check the box, **Build after other projects are built**. For **Projects to watch**, fill in the name **BuildAppJob**.

Step 3: Write the test script that should run after a stable build of the BuildAppJob.

a. Click the **Build** tab.

b. Click **Add build step** and choose **Execute shell**.

c. Enter the following script. The **if** command should be all on one line including the **; then**. This command will **grep** the output returned from the cURL command to see if **You are calling me from 172.17.0.1** is returned. If true, the script exits with a code of 0, which means that there are no errors in the **BuildAppJob** build. If false, the script exits with a code of 1 which means the **BuildAppJob** failed.

```
if curl http://172.17.0.1:5050/ | grep "You are calling me from 172.17.0.1"; then
   exit 0
else
   exit 1
fi
```

d. Click **Save** and then the **Back to Dashboard** link on the left side.

Step 4: Have Jenkins run the BuildAppJob job again.

a. In the top right corner, click **enable auto refresh** if it is not already enabled, otherwise refresh the page.

b. You should now see your two jobs listed in a table. For the **BuildAppJob** job, click the build button on the far right (a clock with an arrow).

Step 5: Verify both jobs completed.

If all goes well, you should see the timestamp for the **Last Success** column update for both **BuildAppJob** and **TestAppJob**. This means your code for both jobs ran without error. But you can also verify this for yourself.

Note: If timestamps do not update, make sure enable auto refresh is turned on by clicking the link in the top right corner or refresh the page.

a. Click the Link for **TestAppJob**. Under **Permalinks**, click the link for your last build, and then click **Console Output**. You should see output similar to the following:

```
Started by upstream project "BuildAppJob" build number 13
originally caused by:
 Started by user admin
Running as SYSTEM
Building in workspace /var/jenkins_home/workspace/TestAppJob
[TestAppJob] $ /bin/sh -xe /tmp/jenkins1658055689664198619.sh
+ grep You are calling me from 172.17.0.1
+ curl http://172.17.0.1:5050/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100   177  100   177    0     0  29772      0 --:--:-- --:--:-- --:--:-- 35400
    <h1>You are calling me from 172.17.0.1</h1>
+ exit 0
Finished: SUCCESS
```

b. It is not necessary to verify your sample app is running because the **TestAppJob** already did this for you. However, you can open a browser tab for **172.17.0.1:5050** to see that it is indeed running.

Part 7: Create a Pipeline in Jenkins

Although you can currently run your two jobs by simply clicking the Build Now button for the **BuildAppJob**, software development projects are typically much more complex. These projects can benefit greatly from automating builds for continuous integration of code changes and continuously creating development builds that are ready to deploy. This is the essence of CI/CD. A pipeline can be automated to run based on a variety of triggers including periodically, based on a GitHub poll for changes, or from a script run remotely. However, in this part you will script a pipeline in Jenkins to run your two apps whenever you click the pipeline **Build Now** button.

Step 1: Create a Pipeline job.

a. Click the **Jenkins** link in the top left, and then **New Item**.

b. In the **Enter an item name** field, type **SamplePipeline**.

c. Select **Pipeline** as the job type.

d. Scroll to the bottom and click **OK**.

Step 2: 2. Configure the SamplePipeline job.

a. Along the top, click the tabs and investigate each section of the configuration page. Notice that there are a number of different ways to trigger a build. For the **SamplePipeline** job, you will trigger it manually.

b. In the **Pipeline** section, add the following script.

```
node {
   stage('Preparation') {
       catchError(buildResult: 'SUCCESS') {
          sh 'docker stop samplerunning'
          sh 'docker rm samplerunning'
       }
   }
   stage('Build') {
       build 'BuildAppJob'
   }
   stage('Results') {
       build 'TestAppJob'
   }
}
```

This script does the following:

- In creates a single node build as opposed to a distributed or multi node. Distributed or multi node configurations are for larger pipelines than the one you are building in this lab and are beyond the scope of this course.
- In the **Preparation** stage, the **SamplePipeline** will first make sure that any previous instances of the **BuildAppJob** docker container are stopped and removed. But if there is not yet a running container you will get an error. Therefore, you use the **catchError** function to catch any errors and return a “SUCCESS” value. This will ensure that pipeline continues on to the next stage.
- In the **Build** stage, the **SamplePipeline** will build your **BuildAppJob**.
- In the **Results** stage, the **SamplePipeline** will build your **TestAppJob**.

c. Click **Save** and you will be returned to the Jenkins dashboard for the **SamplePipeline** job.

Step 3: Run the SamplePipeline.

On the left, click **Build Now** to run the **SamplePipeline** job. If you coded your Pipeline script without error, then the **Stage View** should show three green boxes with number of seconds each stage took to build. If not, click Configure on the left to return to the **SamplePipeline** configuration and check your Pipeline script.

Step 4: 4. Verify the SamplePipeline output.

Click the latest build link under **Permalinks**, and then click **Console Output**. You should see output similar to the following:

```
Started by user admin
Running in Durability level: MAX_SURVIVABILITY
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/SamplePipeline
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Preparation)
[Pipeline] catchError
[Pipeline] {
[Pipeline] sh
+ docker stop samplerunning
samplerunning
[Pipeline] sh
+ docker rm samplerunning
samplerunning
[Pipeline] }
[Pipeline] // catchError
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Build)
[Pipeline] build (Building BuildAppJob)
Scheduling project: BuildAppJob
Starting building: BuildAppJob #15
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Results)
[Pipeline] build (Building TestAppJob)
Scheduling project: TestAppJob
Starting building: TestAppJob #18
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```