Practice 1
Network Administration (CC 312)

Date: 28/09/2024

Part 1: Packet Tracer - Compare CLI and SDN Controller NetworkManagement

Addressing Table

Note: All subnet masks are /24 (255.255.255.0).

Device	Interface	IP Address
R1	G0/0/0	192.168.101.1
	S0/1/0	192.168.1.2
R2	G0/0/0	192.168.102.1
	S0/1/1	192.168.2.2
R3	G0/0/0	10.0.1.1
	G0/0/1	10.0.2.1
	S0/1/0	192.168.1.1
	S0/1/1	192.168.2.1
SWL1	VLAN 1	192.168.101.2
SWL2	VLAN 1	192.168.102.2
SWR1	VLAN 1	10.0.1.2
SWR2	VLAN 1	10.0.1.3
SWR3	VLAN 1	10.0.1.4
SWR4	VLAN 1	10.0.1.5
Admin	NIC	10.0.1.129
PC1	NIC	10.0.1.130
PC2	NIC	10.0.2.129
PC3	NIC	10.0.2.130
PC4	NIC	192.168.102.3
Example Server	NIC	192.168.101.100
PT-Controller*	NIC	192.168.101.254

* In Part 3, you will add and configure PT-Controller0.

In this Packet Tracer activity, you will compare the differences between managing a network from the command line interface (CLI) and using a software-defined networking (SDN) controller to manage the network.

Instructions

Part 1: Explore the Network Topology

In this Part, you will become familiar with the topology you will use for network programmability activities.

Step 1: Review the network configuration documentation

The network is configured as follows:

- Routers are running OSPFv2.
- SSH is enabled on all devices with user **cisco** and password **cisco123!**
- R1 has no hosts.
- R2 LAN IPv4 is statically configured.
- R3 is the DHCPv4 server for LAN1 and LAN2.
- Switches are Layer 2 (no VLANs).
- All **SWR#** switches belong to LAN1.

Step 2: Verify that all devices can ping each other.

Either use the command line on each device or use the **Add Simple PDU (P)** tool to verify that all devices can ping each other.

Part 2: Use the CLI to Gather Information

In this part, you manually access each device to gather information about the software version.

Step 1: From the Admin PC, securely access the SWR3 switch.

a. Click **Admin > Desktop > Command Prompt**.

b. Enter the command **ssh -l cisco 10.0.1.4**. The -l option is the letter "L", not the number one.

c. When prompted, enter **cisco123!** as the password. You are now logged in to SWR3.

Step 2: Gather information about the software on SWR3.

Enter the following command to filter the output of the **show version** command to view just the RELEASE SOFTWARE installed on the device. Notice that SWR3 is running IOS 16.3.2 and Boot Loader 4.2.6.

```
SWR3# show version | include RELEASE
Cisco IOS Software [Denali], Catalyst L3 Switch Software (CAT3K_CAA-UNIVERSALK9-M),
Version 16.3.2, RELEASE SOFTWARE (fc4)
BOOTLDR: CAT3K_CAA Boot Loader (CAT3K_CAA-HBOOT-M) Version 4.26, RELEASE SOFTWARE (P)
SWR3#
```

Copy the information to your clipboard
Open a text file editor and paste the information into a text fileSave
the file as **software-versions.txt**.

Step3: Gather the software information for the rest of the network devices.

1) From the **Command Prompt** on SWR3, securely access another network device and repeat Step 2 above.
2) Continue documenting the software versions until you have completed all nine network devices: SWL1, SWL2, SWR1, SWR2, SWR3, SWR4, R1, R2, and R3
3) Exit out of all of your SSH sessions.

Part 3: Configure the PT-Controller

For many years, network administrators have used early automation tools such as bash scripts or SNMP-enabledsoftware to complete a process similar to what you did in the previous step. However, with the introduction of SDN, this process has been greatly enhanced. Packet Tracer provides a simple PT-Controller to simulate an SDN controller. In this Part, you will connect and configure the PT-Controller.

Note: To learn more about Packet Tracer's implementation of the Network Controller, click the **Help** menu, then **Contents**. In the Index on the left, about midway down, you will find the heading **Configuring Devices**. Underneath this heading, find **Network Controllers**. Here you will find a wealth of information, much of which you will explore in the activities in this course.

Step 1: Add a Network Controller to the topology.

1) At the bottom left corner of the Packet Tracer interface, click **End Devices > Network Controller**.
2) Add the Network Controller in the blank spot left of the **SWL1** switch. The name should already by **PT-Controller0**. If not, click the name and change it.
3) At the bottom again, click the lightening bolt for **Connections**. Click the solid black **Copper Straight-Through** cable.
4) Click **PT-Controller0** and choose **GigabitEthernet0**. Then click **SWL1** and choose the first available Gigabit Ethernet interface.

Step 2: Configure connectivity for the PT-Controller0.

1) Click **PT-Controller0 > Config**.
2) For **Gateway/DNS IPv4**, enter 192.168.101.1 as the **Gateway** address
3) On the left under **INTERFACE**, click **GigabitEthernet0**.
4) For **IP Configuration**, enter the **IP Address** 192.168.101.254 and **Subnet Mask** 255.255.255.0.
5) On the left, under **REAL WORLD**, click **Controller**. If the **Server** Status is **Stopped**, move on to the next substep. If the **Server Status** is **Disabled in Preferences**, then you will need to enable external access by following these instructions:
   a) Select **Options > Preferences** from the Packet Tracer menus.
   b) Click **Miscellaneous**
   c) Under **External Network Access**, click **Enable External Access for Network Controller REST API**.
   d) Close **Preferences** and click **PT-Controller0 > Config**, if necessary.
   e) On the left under **REAL WORLD**, click **Controller**.

   The **Server Status** should now be **Stopped**. Click **Access Enabled** to enable it. **Server Status** changes to **Listening on port 58000**. If the port is some other value, change it to **58000**. This is the port number in the Python scripts.

Step 3: From Admin, verify connectivity to the PT-Controller0.

Verify that Admin can ping PT-Controller0. If you are not able to ping, make sure your configuration matches the specifications in the previous step.

Step 4: Register a new user and log into the PT-Controller0.

1) Click **Admin > Desktop > Web Browser**.
2) Enter the IPv4 address 192.168.101.254 to access the **User Setup** for **PT-Controller0**
3) Enter **cisco** in the **Username** field and **cisco123!** in the **Password** and **Confirm Password** fields, and then click **SETUP**.

   Note: You can use whatever username and password you want here. For simplicity, we recommend using common credentials used in the rest of the activity.

   On **User Login** screen, enter your credentials and click **LOG IN**.

   You are now logged in to the dashboard for **PT-Controller0**. At this point, it may be helpful to expand the window so you can see the entire interface.

Part 4: Use an SDN Controller to Discover a Topology

In this Part, you will configure PT-Controller0 to use Cisco Discover Protocol (CDP) to automatically discoverthe nine network devices in your topology. The PT-Controller0 will also discover all five host devices attached to the network.

Step 1: Add credentials to access all the network devices in the topology.

1) From the **Network Controller** GUI, click the menu button to the left of the Cisco logo.
2) Select **Provisioning**. From here, you can manually add networking devices. However, you will use CDP to automatically discover devices for you.
3) Click **CREDENTIALS** and then click **+ CREDENTIAL** to add a **New Credential**.
4) For **Username**, enter **cisco**, and for **Password**, enter **cisco123!**. Leave **Enable Password** blank. For **Description**, enter **admin credentials**, and then click **OKAY**.
5) The new CLI Credentials are now stored on PT-Controller0 for use in automation tasks.

Step 2: Use CDP to discover all the devices on the network.

1) Click **DISCOVERY** and the click **+ DISCOVERY** to add a **New Discovery**.
2) For **Name**, enter **SWL1**. For **IP Address**, enter **192.168.101.2**. For **CLI Credential List**, drop down thelist and choose **cisco - admin credentials**.
3) Click **ADD**.
4) You should now see the **Status** as **In Progress**. You can wait for Packet Tracer to finish simulating this process. Or you can **Fast Forward Time** button on the main Topology window to speed up the process.

Part 5: Use an SDN Controller to Gather Information

In this Part, you will use the PT-Controller0 GUI to view information about the network devices and host devices in the topology. You will view the topology created by the controller and then conduct a path trace across the network.

Step 1: View the list of network devices discovered.

a. Click **NETWORK DEVICE**. You should now see all nine network devices listed.

b. Click the Gear icon next to any device's hostname to see the information collected by the discover process. Notice that the **Software Version** is listed as well as a variety of other detailed information about the device.

Step 2: View a list of all the host devices discovered.

a. Return to the Dashboard. Click the menu next to the Cisco logo, then click **Dashboard**. (You can also simply click the **Network Controller** banner to return to the **Dashboard** from anywhere.)

b. On the Dashboard, you will see charts with the number of hosts that can be reached via ping and the number of network devices that are managed. Both should be 100%.

c. You should also see tiles for **QoS**, **Network Device**, and **Host**. Click the Gear icon for **Host**. This will take you to the **HOSTS** tab for **ASSURANCE**.

d. On this page, you can view all the Layer 2 and Layer 3 connectivity information for each host as well as the network device to which each is attached.

e. Click the Gear icon next to any host to view more detailed information.

Step 3: View the topology created by PT-Controller0.

a. Click the **TOPOLOGY** tab. Notice that the PT-Controller dynamically created the same topology you see in Packet Tracer's main window.

b. From this view, you can click any network device to see its details.

c. You can also click and drag the device icons to rearrange the topology. However, your changes will not be saved when you leave the **TOPOLOGY** workspace.

Step 4: Trace the path from one device to another device.

a. Click the **PATH TRACE** tab.

b. Click **+ PATH** to add a **New Path**.

c. Trace the path from one end of the network to the other. For example, you could enter the IP addresses for PC1 to PC4. Then click **OKAY**.

d. Click the new path that was added to initiate the path trace.

You will get a **Route** report that shows all the hops from source to destination. Notice that only Layer 3 device information is listed. The switches are shown as an **UNKNOWN** device. This is because they are all operating at Layer 2 only.

Part 6: Use an SDN Controller to Configure Network Settings

A major benefit of network automation using a controller is the ability to configure global network settings and policies for all devices and then push that configuration with the click of a button. In this Part, you will configure **PT-Controller0** with network settings for DNS, NTP, and Syslog. You will then push this configuration to supported network devices. Finally, you will verify and test the policy.

Step 1: Investigate the configuration of the Example server.

a. Click **Example Server > Services**.

b. Under **SERVICES**, click **DNS**. Notice that the DNS service is enabled and that there is one record for www.example.com.

c. Under **SERVICES**, click **SYSLOG**. Notice that the Syslog service is enabled.

d. Under **SERVICES**, click **NTP**. Notice that the NTP service is enabled.

Step 2: Configure a global policy for DNS, SYSLOG, and NTP.

a. Click **Admin**. If you closed **Admin**, you will need to open the **Web Browser** app and reauthenticate with **PT-Controller0**.

b. Click the menu to the left of the Cisco logo.

c. Click **Policy**.

d. On the **QOS** tab, notice there are options for configuring the **Scope** and **Policy**. In this activity, you will configure **NETWORK SETTINGS**.

e. Click **NETWORK SETTINGS**.

f. Click **DNS**. Enter **example.com** as the **Domain Nam**e and **192.168.101.100** as the **IP Address**.

g. Click **Save**.

h. Click **NTP**.

i. Enter **192.168.101.100** as the **IP Address**.

j. Click **Save**.

k. Click **SYSLOG**.

l. Enter **192.168.101.100** as the **IP Address**.

m. Click **Save**.

n. Click **DNS**, **NTP**, and **SYSLOG** again to verify the information is correct. If not, correct the information saving each time.

o. Click **PUSH CONFIG**.

p. The **Push All Network Settings** dialog box opens. Verify your settings and click **OKAY**. A "Saved Successfully" message appears briefly.

Step 3: Verify and test the network settings that were pushed to devices.

At the bottom of the **NETWORK SETTINGS** window, there is the following:

Note: This functionality is only supported on devices running IOS-XE OS and Switch 2960-24TT

This means that, for this version of Packet Tracer, your global settings were only applied to the routers.

a. Click any of the three routers. **R1** is shown in the following output.

b. Click **CLI**.

c. Click inside the window and press **Enter** to get a command prompt.

d. Enter the privileged EXEC mode and verify the DNS settings.

```
R1> enable
R1# show run | begin ip domain
ip domain-name example.com
ip name-server 192.168.101.100
!
<output omitted>
R1#
```

e. Enter the following commands to verify the NTP settings. The time on R1 should match your current time. Packet Tracer may take a little time to propagate NTP messages. You can click the **Fast Forward Time** button to speed up the process.

```
R1# show ntp associations

address         ref clock       st   when     poll    reach  delay          offset
disp
*~192.168.101.100127.127.1.1     1    12       16      377    0.00           0.00
0.12
 * sys.peer, # selected, + candidate, - outlyer, x falseticker, ~ configured
R1# show clock
15:30:54.268 UTC Thu Jun 11 2020
R1#
```

f. Enter the following command to verify logging is configured.

```
R1# show run | include logging
logging 192.168.101.100
R1#
```

g. To test logging, shut down the Serial0/1/0 interface and then reactivate it.

```
R1# configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)# interface s0/1/0
R1(config-if)# shutdown
%LINK-5-CHANGED: Interface Serial0/1/0, changed state to administratively down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1/0, changed state to down
15:36:37: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.2.1 on Serial0/1/0 from FULL to DOWN,
Neighbor Down: Interface down or detached
R1(config-if)# no shutdown
%LINK-5-CHANGED: Interface Serial0/1/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/1/0, changed state to up
15:36:53: %OSPF-5-ADJCHG: Process 1, Nbr 192.168.2.1 on Serial0/1/0 from LOADING to
FULL, Loading Done
R1(config-if)# end
R1#
```

Click **Example Server > Services > SYSLOG**. You should see the same syslog messages you saw on in the CLI are also logged to the server. Double-click any of the entries to review the messages.

Part 2: Build a CI/CD Pipeline Using Jenkins (10 points)

In this section, 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

- 1 PC with operating system of your choice
- Virtual Box or VMWare
- DEVASC Virtual Machine

Instructions

Part 1: Launch the DEVASC VM

If you have not already completed the **Lab - Install the Virtual Machine Lab Environment**, do so now. If you have already completed that lab, launch the DEVASC VM now.

Part 2: 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 and create a new repository.**

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**

Step 2:**Configure your Git credentials locally in the VM.**

Open a terminal window **with VS Code** in the DEVASC VM. Use your name in place of "Sample User" for the name in quotes " ". Use @example.com for your email address.

```
devasc@labvm:~$ git config --global user.name "Sample User"
devasc@labvm:~$ git config --global user.email sample@example.com
```

Step 3:**Initialize a directory as the Git repository.**

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 4:**Point Git repository to GitHub repository.**

Use the **git remote add** command to add a Git URL with a remote alias of "origin" and point 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 5: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 ofyour 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': 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% (5/5), done.
```

```
Writing objects: 100% (8/8), 1.05 KiB | 1.05 MiB/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To https://github.com/AllJohns/sample-app.git
   d0ee14a..4030ab6  master -> master
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Note: If, instead of a request for your username, you get a message from VS Code with the message,**The extension 'Git' wants to sign in using GitHub**, then you misconfigured either your GitHub credentials in Step 2 and/or the GitHub URL in Step 4. The URL must have the correct case-sensitiveusername and the name of the repository that you created in Step 1. To reverse your previous **git add** command, use the command **git remote rm origin**. Then return to Step 2 making sure to enter the correct credentials and, in Step 4, entering the correct URL.

Note: If, after entering your username and password, 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 **gitremote rm origin** command.

Part 3: 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 python3 /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 python3 /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. Return to the terminal window where the server is running and press CTRL+C to stop the server.

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 4: 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:lts
lts: 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:lts
docker.io/jenkins/jenkins:lts
devasc@labvm:~/labs/devnet-src/jenkins/sample-app$
```

Step 2:**Start the Jenkins Docker container.**

Enter the following command on **one line**. You may need to copy it to a text editor if you are viewing a PDFversion of this lab to avoid line breaks. 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** specifiesthe 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 displayedas shown below.

```
<output omitted>
*************************************************************
*************************************************************
*************************************************************

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 omitted>
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 server, 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.

```
+----------------------------------------------------------+
|Your Computer's Operating System                          |
|   +--------------------------------------------------+   |
|   |DEVASC VM                                         |   |
|   |   +------------------------------------------+   |   |
|   |   |Docker container                          |   |   |
|   |   |   +----------------------+               |   |   |
|   |   |   | Jenkins server       |               |   |   |
|   |   |   |  +----------------+  |               |   |   |
|   |   |   |  |Docker container|  |               |   |   |
|   |   |   |  +----------------+  |               |   |   |
|   |   |   +----------------------+               |   |   |
|   |   +------------------------------------------+   |   |
|   +--------------------------------------------------+   |
+----------------------------------------------------------+
```

Part 5: 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.

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 **Skip and 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 6: 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 a job** 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 asthere 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 python3 /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 7: 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 first 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. Refresh the web page with the refresh button for your browser.

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.

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 8: 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: **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:

- It 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: **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
```

In spanish:

En esta parte para puntuar debes presentar gráficos de los pasos de ejecución de esta sección y responder las siguientes preguntas (**recuerda que no contestar alguna pregunta implica 0 como puntuación**)

1. ¿Qué es una pipeline de Jenkins? , ¿cuáles son los tipos de pipelines de Jenkins?
2. ¿Cómo se pueden activar (triggered)/ detener / controlar los trabajos de Jenkins mediante programación?
3. ¿Cómo se logra la integración continua con Jenkins?. Explica tu respuesta.
4. Analize lo siguiente:

   Para aplicar CI / CD a la red, el proceso de cambio de red debe modelar el hardware y el software de la infraestructura de red existente. Luego, podemos crear pruebas para validar los cambios de configuración de la red antes de implementarlos en la red de producción.

   Para construir el modelo, necesitamos sistemas de descubrimiento que identifiquen los componentes, interconexiones y configuraciones de la infraestructura de red. El uso de técnicas de virtualización de red como instancias de enrutamiento y reenvío virtual (VRF) y Ethernet VPN (EVPN) complica la tarea al colocar redes virtuales en capas sobre la infraestructura física.

Idealmente, también se determinaría los flujos de red que requieren soporte, identificando potencialmente cada aplicación y sus requisitos de interconexión. Esta tarea es problemática en redes grandes que admiten cientos de aplicaciones. Sería inapropiado modelar y replicar la conectividad utilizada por un paquete de malware desconocido en un modelo de red.

También necesitamos herramientas que utilicen la topología y la configuración descubiertas para crear un "gemelo digital de red" y utilizarlo para las pruebas previas a la implementación. Este gemelo digital debeser un gemelo "que actúe igual" que pueda modelar con precisión los cambios de configuración y topología. Para verificar la operación, deberás replicar suficientes flujos de tráfico para validar si la conectividad deseada existe (o no existe) después de un cambio propuesto.

Por último, necesitamos métodos para evaluar la corrección de la operación de la red, tanto en la red de prueba como eventualmente en la red operativa. Las comprobaciones simples verificarán la accesibilidad del dispositivo de red vecino o las entradas de la tabla de enrutamiento. Esta área es donde se encuentran las capacidades de la industria de las redes en la actualidad.

En el futuro, se espera ver una mayor fidelidad con respecto a los gemelos digitales de red donde el tráfico de red sintético puede potencialmente permitir la verificación de la seguridad y la funcionalidad de calidad de servicio (QoS).

El proceso de diseño y desarrollo de la red debe ser el mismo que usamos para el desarrollo de software. Es decir, desarrolla y prueba el cambio en un entorno de desarrollo que refleje el entorno real. Cuando pasas todas las pruebas, el cambio se envía al proceso de CI/CD. La fase CI construye la red de prueba virtual, aplica el cambio de red propuesto y evalúa los resultados. ¿El cambio logró el resultado deseado?

Las fallas en las pruebas pueden deberse a que el cambio fue incorrecto, incompleto o que las pruebas fueron insuficientes. Una prueba fallida también podría deberse a un conflicto entre dos cambios.
Independientemente, el ingeniero de red recibe los resultados de la prueba para su evaluación y corrección. Sin embargo, si el proceso de CI indica un cambio exitoso, el proceso de CD puede aplicar este cambio a la red de producción.

Sin duda, muchas organizaciones querrán verificar los resultados de las pruebas exitosas antes de aplicarlos a la red de producción como parte de su proceso normal de control de cambios. El objetivo del proceso CI / CD es evitar los errores humanos que prevalecen en los procesos operativos de la red actual.

La adopción de los procesos de cambio de red de CI/CD es un cambio operativo importante y es mejor hacerlo con un enfoque lento y fácil.

La administración comienza con tareas de automatización simples que tengan poco impacto en las operaciones de la red cuando cometa un error. Sí, hay errores, hay que tratar de hacerlos pequeños y de bajo impacto.

Mantiene tus cambios pequeños dividiendo los cambios más grandes en partes pequeñas e independientes. Por ejemplo, una implementación de QoS podría comenzar con la clasificación y el marcado, mientras que el siguiente paso implementa la lógica de reenvío y cola.

Debes adoptar un proceso CI/CD para subconjuntos de la configuración general de la red. Por ejemplo, comienza administrando la configuración de todos los elementos de configuración estática global, como el protocolo de tiempo de red (NTP), el protocolo de administración de red simple (SNMP), el protocolo de registro del sistema (Syslog), el protocolo de configuración dinámica de host (DHCP) y la autenticación de inicio de sesión. Luego, aplica lo que aprendes a partes más avanzadas de la configuración, como interfaces y protocolos de enrutamiento. Al comenzar, haga que el diseño y la configuración de la red sean consistentes en construcciones similares. Otros sitios del mismo tamaño general deben usar un diseño y una configuración que solo difiera en cosas como las direcciones IP. Esta coherencia simplifica enormemente la creación de entornos de prueba y facilita la automatización de los cambios de configuración.

El proceso de CI / CD debería convertirse en la única forma de implementar nuevos cambios en las partes cubiertas de la configuración. De lo contrario, crearía inconsistencias en la red que agregan complejidad innecesaria a tu proceso de automatización de CI / CD.

El objetivo de esta práctica es automatizar el ciclo de compilación con herramientas como Jenkins.. Todo el proceso puede parecer mucho trabajo, pero el objetivo es reducir las interrupciones de la red mediante la adopción de las mejores prácticas de la industria del desarrollo de software.

5. Explica qué es virtual routing and forwarding (VRF) y Ethernet VPN (EVPN).