QATIP Intermediate
Azure Lab08
Terraform Pipelining with Jenkins

Contents

Lab Objectives

In this lab, you will:

1.   Provision a storage account for use as a terraform remote backend

2.   Deploy and configure an Ubuntu Jenkins instance

3.   Create and run Jenkins pipelines to deploy and destroy Azure resources using Terraform

Teaching Points

This lab aims to demonstrate how businesses can control terraform deployments through the use of change control and release mechanisms. In this case Github will be used for change control combined with Jenkins which will deploy changes once approved. There are many alternative approaches, including...

GitLab CI/CD

GitLab has built-in CI/CD capabilities that can integrate seamlessly with Terraform. GitLab runners execute Terraform plans and applies. GitLab's Terraform integration allows for state file management and monitoring.

Azure DevOps

Azure Pipelines support Terraform workflows directly. Integration with Azure Repos or GitHub for version control. Features like environment staging, multi-step workflows, and approvals make it an excellent choice for CI/CD

GitHub Actions

GitHub Actions supports Terraform via predefined workflows. It allows for triggering Terraform commands on repository events (e.g., push, pull request) and offers native integration with GitHub-hosted runners

Spinnaker

Focuses on application delivery and infrastructure automation. It can be integrated with Terraform to manage infrastructure alongside application releases and supports multi-cloud environments.

Solution

There is no pre-configured solution to this lab. Reach out to your instructor if you have any difficulties completing the tasks and ensure each one is correct before moving on.

Before you begin

Before you begin, ensure you have followed the lab setup instructions for Azure and navigated to the C:\azurelabs\lab6\Jenkins folder in VSC terminal and explorer.

Task1. Create a Jenkins Virtual Machine

1. In the Azure portal, ensure that Resource Group RG1 does not exist. If it is present, then manually delete it before proceeding. In the IDE terminal, run the following command to create an Ubuntu virtual machine onto which we will install Jenkins

    c:\azure-tf-int\labs\08\createjenkinsvm.ps1

2. Take note of your vm public IP address or navigate to it in your Azure Portal and retrieve it when needed.

3. Jenkins requires connectivity over port 8080, so add a rule to allow this inbound traffic (note that in production environments, these rules would be more granular)...

    az vm open-port --port 8080 --resource-group RG1 --name VM1

4. Use SCP to copy a script file to the virtual machine (replace <vm_ip> with your vm public IP address)..

    cd c:\azure-tf-int\labs\08

    scp ./setup_jenkins_azure.sh azureuser@<vm_ip>:/home/azureuser/

5. Connect to the virtual machine using SSH with the following command, replacing <vm-ip> with your vm IP address

    ssh azureuser@<vm_ip>

6. Run following commands to makes the script executable, set linux style line breaks and finally run it (copy and paste all 4 lines)...

    cd /home/azureuser/

    chmod +x setup_jenkins_azure.sh

sed -i 's/\r$//' setup_jenkins_azure.sh

./setup_jenkins_azure.sh

7. You can move on to the next tasks whilst the Ubuntu updates and Jenkins installation is performed.

Task2: Create a Storage Account for Remote State

In this task you will create an Azure storage account and blob container for later use as a terraform remote backend.

1. Navigate to the Azure Portal.

2. Search for and select **Storage Accounts**.

3. Click **Create** to create a new storage account....

>   **Resource Group**: select existing ... **RG1**
>
>   **Storage Account Name**: **jenkinsstate<your-name>** , replacing `<your-name>` with a unique identifier.
>
>   **Region**: **West Europe**
>
>   **Performance**: **Standard**
>
>   **Replication**: **Locally-redundant storage (LRS)**

4. Leaving other settings as defaults, click **Review + Create** and then **Create**.

5. Once the storage account is created, navigate to it and create a new **Blob Container** (Refresh your browser if the blade does not displays correctly):

>   **Name**: **terraform-state**
>
>   **Public access level**: **Private**
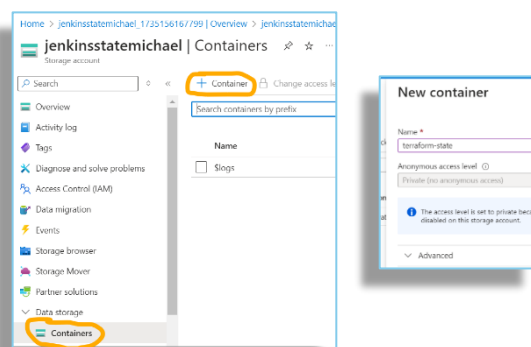
6.   Record the storage account name and container name for later use.

## Task 3. Create a Github account and repository

In this task you will create github repository to which you will upload a Jenkins and terraform file for use in the Jenkins pipeline later.

Task3a. Sign up to Github

1.   If you already have a Github account that you are prepared to use, then skip to Task3b.

2.   Sign up to Github using a personal email address that is not currently associated with Github…

   a. Navigate to https://github.com/

   b. Enter a personal email address and select "Sign up for Github"…
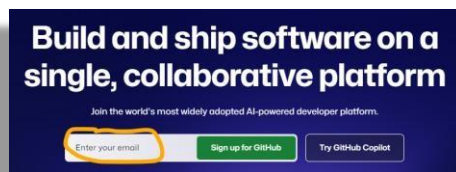
   c. Enter a password and a unique username of your choice..
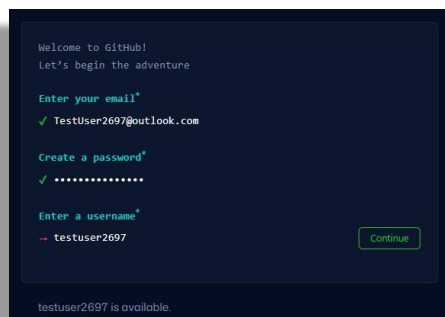
   d. Record and save your chosen **username** and **password** in your session-info file for safekeeping.

   e. Complete the challenge to prove you are a human…
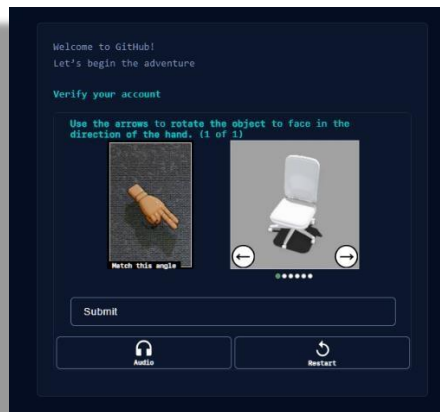
f. An email will sent containing your launch code. Retrieve this and enter it..

g. You will then be prompted to log into github using your new account..
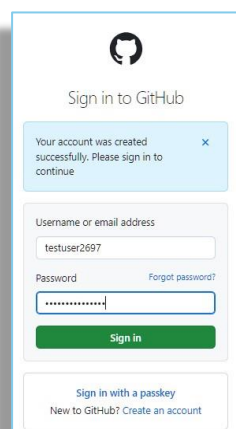
h. Complete the questionnaire…
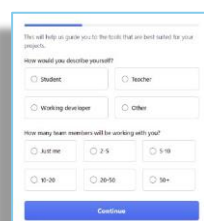
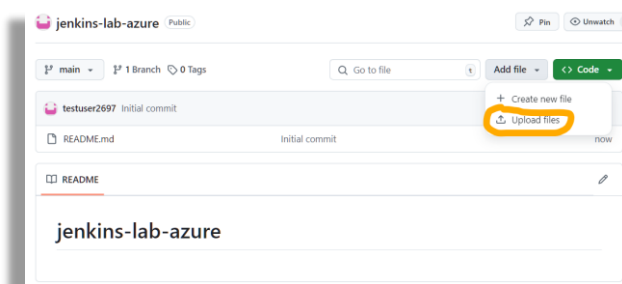i. When asked to select a subscription, select "**Continue for free**"..
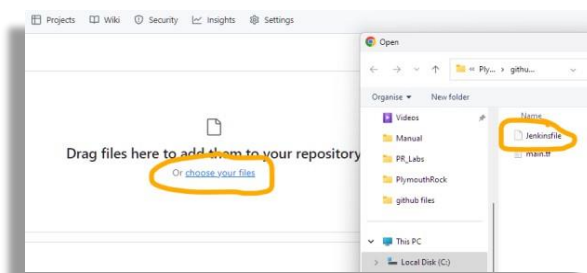
Task 3b. Create a public repository

1. Click on **Create repository**

2. Enter a Repository name of your choice. Ensure **Public** is selected and check the '**Add a README file'** option. Then click on **Create repository**…
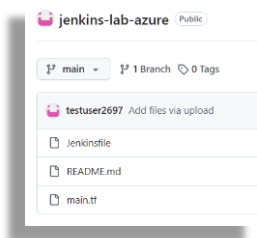

3. Click on **Add file**, **Upload file**..

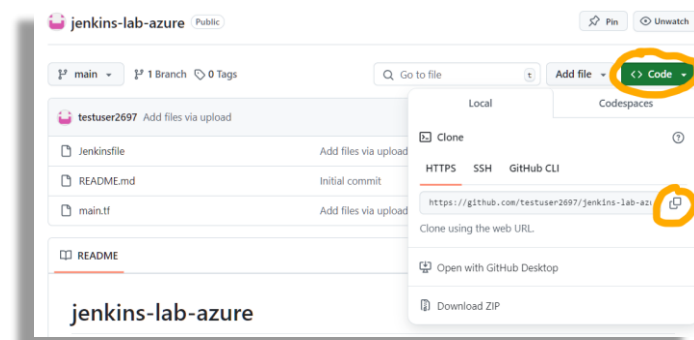4. Select the file **Jenkinsfile** from your **C:\azure-tf-int\labs\08** folder…
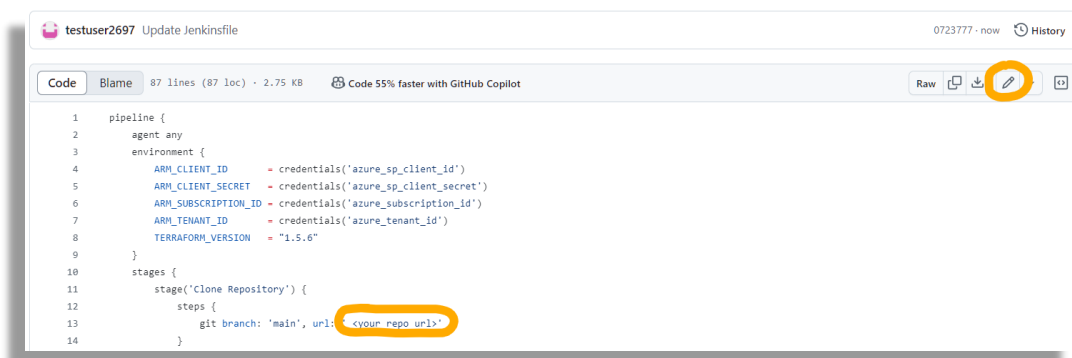

5. Click on Commit changes..
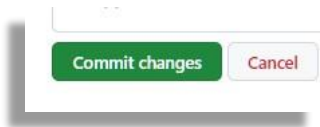
6. Repeat the previous 2 steps to upload the **main.tf** file from the **C:\azure-tf-int\labs\08** folder

7. The 2 files should now be listed..

8. Copy the URL of your repository to your clipboard..

9. Click on the file **Jenkinsfile** and then click on the **edit** icon to open the file for editing.

10. On line 13, replace **<your repo url>** with *your* Repo URL and then click on **Commit changes** twice.

11. Click on **main.tf** and then open it for editing

12. Note lines 24 to 26. This is the resource that will be created by the Jenkins pipeline. Here it is just an empty Resource group but in production this would contain more resources.

13. On line 12, replace **<your subscription id here>** with your lab subscription id.

12. On line 18, replace **<your storage account>** with the name of the storage account you created in Task2…

```
14    terraform {
15      backend "azurerm" {
16        resource_group_name  = "RG1"
17        storage_account_name = "<your storage account>"
18        container_name       = "terraform-state"
19        key                  = "terraform.tfstate"
20      }
```

13. Commit these changes as before.

14. Leave the Github tab open as we will return to it later.

Task 4. Configure Jenkins

1. Open Jenkins in a new browser tab (replace <vm_ip> with the public IP address of your Jenkins server)…

   http://<vm_ip>:8080

2. In your IDE, once the Jenkins installation completes, enter the command shown to reveal the initial Jenkins admin password. Copy and paste this from your IDE terminal into the **Unlock Jenkins** screen the click on Continue..

```
Setup completed successfully!
################################################################
# To retrieve the Jenkins initial admin password, enter the command:   #
# sudo cat /var/lib/jenkins/secrets/initialAdminPassword               #
################################################################
azureuser@VM1:~$ sudo cat /var/lib/jenkins/secrets/initialAdminPassword
98709630c5e84100b589ecbfd043a1d2
azureuser@VM1:~$
```

3. Select **Install suggested plugins**

4. On the **Create First Admin User** screen; Select **Skip and continue as admin**

5.   Click **Save and Finish** to complete the Jenkins configuration.

6.   Click **Start using Jenkins**

## Task 5. Add Azure credentials to Jenkins

1.   In order to use Terraform on Jenkins to interact with Azure, we must supply it with credentials to use, a service principal being recommended.

2.   In the Azure portal, locate and note your Subscription ID

3.   At the IDE terminal, finish your SSH session by typing **exit**.

4.   This will return you to the terminal command prompt. Enter the following command to create a service principal, replacing **<S_ID>** with your subscription ID (this can be found by selecting "Subscription" in the Azure portal Home screen)…

      az ad sp create-for-rbac --name "jenkins-sp" --role Contributor --scopes /subscriptions/<S_ID>

5.   Details of the service principle will be displayed (your values will differ)…
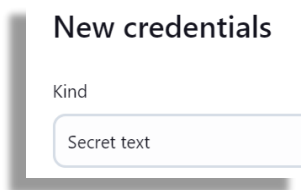
6.   In your **Jenkins** browser session; Navigate to **Manage Jenkins > Credentials**.

7.   On the breadcrumb menu; click on the **Credentials** dropdown and then select **System .**..
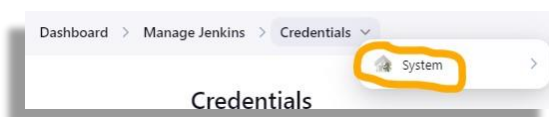
8.   Click on "**Global credentials (unrestricted)**"…
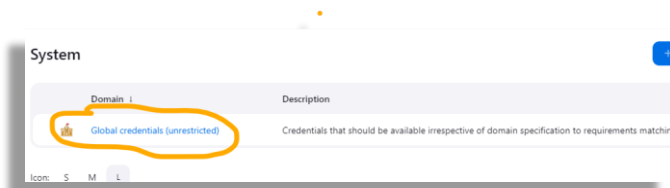
9. Click on "**Add Credentials**" and select "**Secret text**" as the Kind..
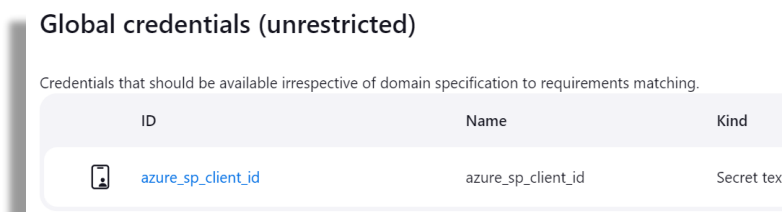
New credentials

Kind

Secret text

10. From your service principal output information, copy the **AppID** parameter value, excluding the quotes, and paste this into the **Secret** field. Enter **azure_sp_client_id** as the ID..

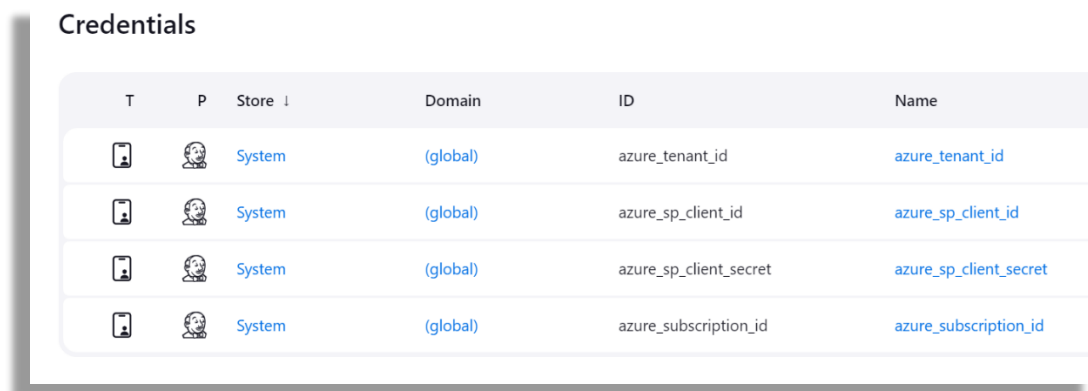Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind
azure_sp_client_id	azure_sp_client_id	Secret text

11. Click on **Create** to create this credential.

12. Repeat steps 9 through 11 to create the following three additional credentials..

Secret	ID
Your Service principal password	**azure_sp_client_secret**
Your Service principal tenant	**azure_tenant_id**
Your Subscription ID	**azure_subscription_id**

13. Ensure your credential IDs match those shown below…

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	azure_tenant_id	azure_tenant_id
		System	(global)	azure_sp_client_id	azure_sp_client_id
		System	(global)	azure_sp_client_secret	azure_sp_client_secret
		System	(global)	azure_subscription_id	azure_subscription_id

Task 6. Configure a Jenkins pipeline job

1. Select "**+ New Item**" from the Jenkins **dashboard**

2. Enter "**Terraform Pipeline**" as the item name

3. Select "**Pipeline**" as the item type

4. Click "**OK**"

5. On the **General** page displayed next, scroll down to the **Pipeline** section. Use the dropdown list to change the **Definition** from "**Pipeline script**" to "**Pipeline script from SCM**"

6. Select "**Git**" from the **SCM** dropdown list

7. In the **Repository URL**: Enter **your** Github repository URL

8. In "**Branches to build**," "**Branch Specifier**;" change from */master to ***/main**
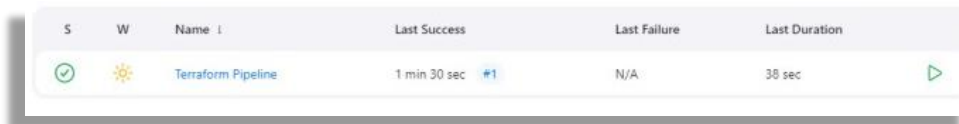

9. Click "**Save**"

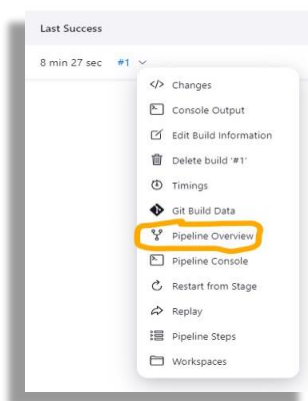10.   Click "**Build Now**"

Task 7. Verify the pipeline run and explore Jenkins

1.   In Jenkins, return to the Dashboard. A record of the pipeline will be displayed showing run success and failure. Refresh the page until a result of the pipeline run is displayed…


2.   Hover over and the select the drop-down menu against **#1** and choose **Pipeline Overview**..

3.   The stages of the pipeline are shown, with ticks (or crosses) indicating success or failure at that stage…


4.   Examine the console logs generated by each stage by selecting the stage and then clicking in the green area to toggle on/off the log display.

5.   Spend a little time exploring the Jenkins interface, using the bread-crumb menu to navigate around, and finally return to the main Dashboard…
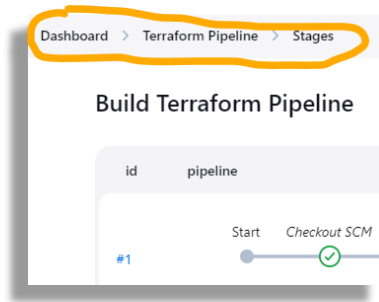
6. In the Azure portal, navigate to the terraform-state container and verify the existence of your state file "**terraform.tfstate**"..
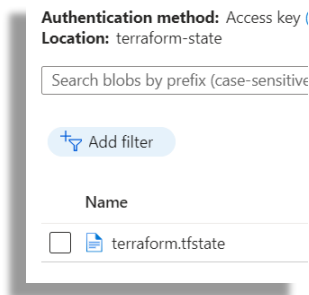
7. Still in the portal, verify the existence of the new resource group "**RG2**" as configured in main.tf…
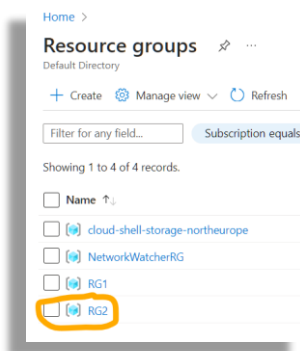

Task 8. Updating the Jenkins pipeline

In a production environment developers would now check-out the contents of the Github repo to their local machine, make updates to the terraform files and then check them back into the repo for approval. Once approved these changes would be merged with the current files and deployed by triggering a new pipeline run. This can be done manually in Jenkins or automatically using Webhooks, whereby Github notifies Jenkins of the changed files, and the pipeline run starts automatically to deploy these changes.

In this task we will simplify the process; you will modify the terraform files directly in Github before manually triggering a new pipeline run in Jenkins. We will then set up Webhooks to show how changes in Github can automatically trigger the pipeline run.

1. Return to your Github repository, logging back in if necessary.

2. Open **main.tf** for editing

3. Change the name of the resource group account to be created to "**RG3**" and commit this change.

4. Switch to Jenkins and click on the play icon to schedule a manual running of the pipeline..
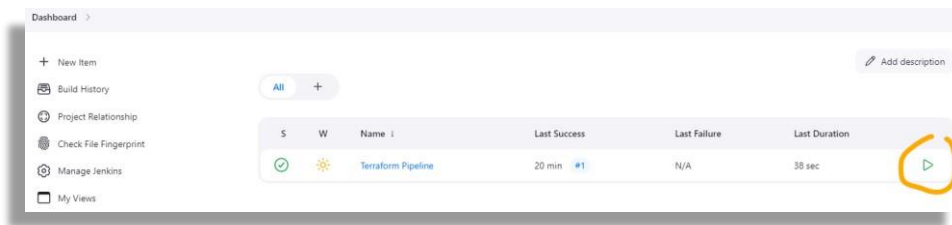

5. The build Executor Status will show the progress of the run..

6. The success runs count should increase to #2 indicating successful manual running of the pipeline. (You may need to refresh the page) Hover-over and then click on the drop-down to the right of 2# and select "**Pipeline Console**"

7. Looking at the logs for the **Terraform Apply** phase we see that the old resource group was replaced as resource group names are immutable and cannot be changed.

8. Switch to your Azure portal and verify the deletion of the old resource group and the creation of a new one, refreshing the display if necessary.

9. To configure automatic pipeline running; **In Jenkins**, re-configure the pipeline by first selecting it on the main dashboard and then choosing the "**Configure**" option..
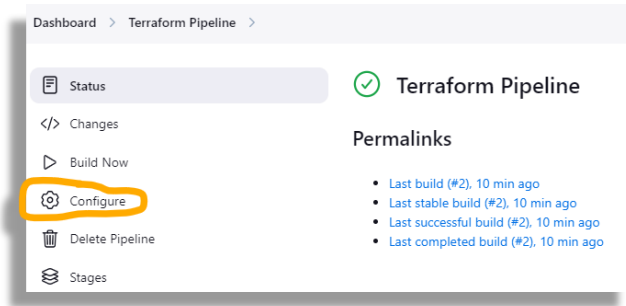
10. Scroll down to the Build Triggers section, select "**Github hook trigger for GITScm polling**" and click on "**Save**" …
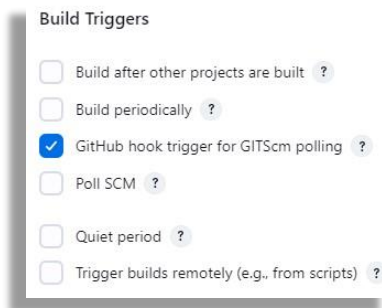
11. **Switch to Github**. Select the **Settings** for your repo. Scroll down and select **Webhooks**. Click on **Add webhook** (you may be prompted to re-authenticate at this point)…
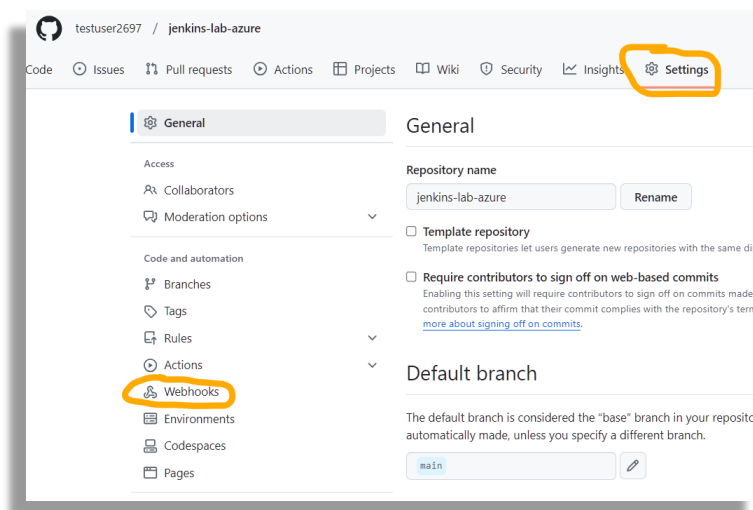
12. For **Payload URL;** enter **http://{Jenkins Public IP}:8080/github-webhook/** replacing (Jenkins Public IP} with the Public IP address of your Jenkins instance

13. For Content type; select **application/json**

14. Select to disable **SSL verification** for this lab environment.

15. Verify your settings as shown in example below (your IP will differ) before clicking on **Add webhook**..

16. In **main.tf** change to the name of your resource group to "**RG4**" and commit the changes as before.

17. Re-visit your Webhooks setting and you should see confirmation that there was a successful push of the changes to Jenkins…
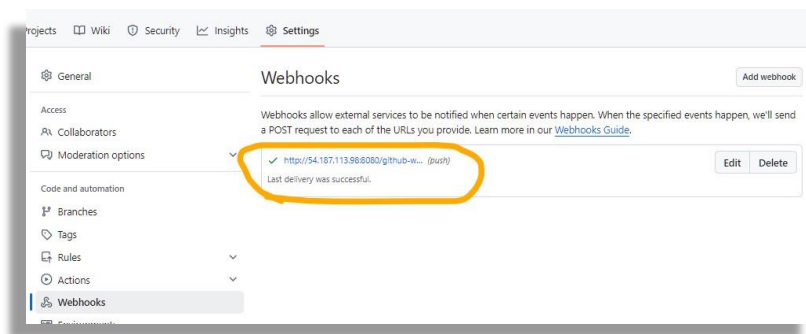
18. Switch to Jenkins. Return to the Dashboard and check that there is now a record of a third successful running of the pipeline (it may still be running so refresh the page periodically)…


19. Switch to your Azure portal and verify the new resource group has been created. You may need to refresh the display.

Task 9. (Time permitting). Configure a Jenkins Destroy pipeline

1. Select " **+ New Item**" from the Jenkins **dashboard**

2. Enter "**Terraform Pipeline Destroy**" as the item name

3. Select "**Pipeline**" as the item type

4. Click "**OK**"

5. On the **General** page displayed next, scroll down to the **Pipeline** section. Use the dropdown list to change the **Definition** from "**Pipeline script**" to "**Pipeline script from SCM**"

6. Select "**Git**" from the **SCM** dropdown list

7. In the **Repository URL**: Enter **your** Github repository URL

8. In "**Branches to build**," "**Branch Specifier;**" change from ***/master** to ***/main**
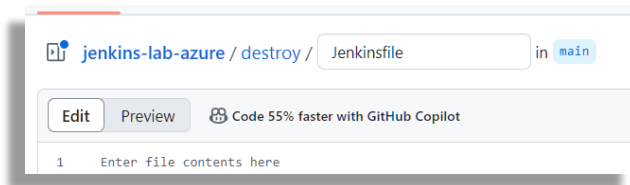
9. For Script path, enter **destroy/Jenkinsfile** …
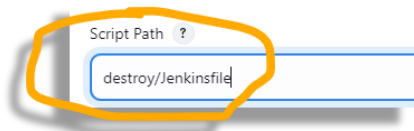
10. Save the new pipeline but **do not** build it yet

11. Switch to your Github account

12. Create a new empty Jenkinsfile in a new folder "destroy"…
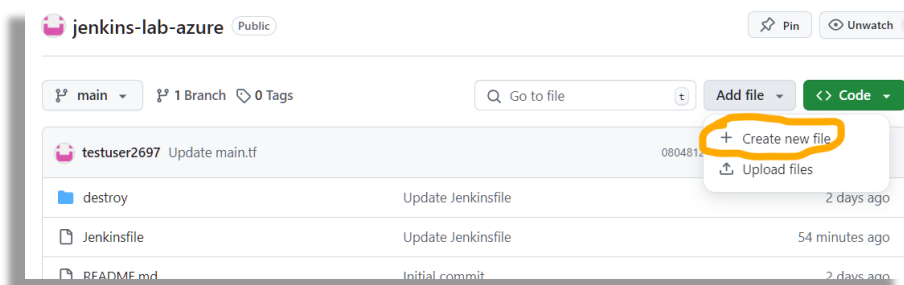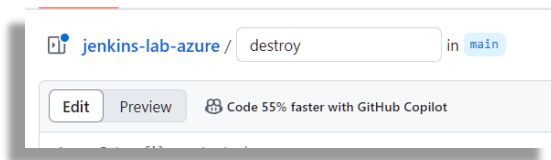
Note: Entering **destroy/** will create the **destroy** folder

13. Using explorer in your IDE, open **Jenkinsfile** in the **c:\azurelabs\labs\08\destroy** folder and copy the contents into your new github file. (Ensure you select the destroy folder!)

14. Update line 13 with **your** repo URL and then commit the changes.

15. Switch back to Jenkins and **Build** the pipeline

16. This Jenkinsfile mandates that approval must be granted for the deletion to proceed. Click on the pipeline, and under **Builds**, select the running Terraform Pipeline Destroy job and then select **Console Output**..
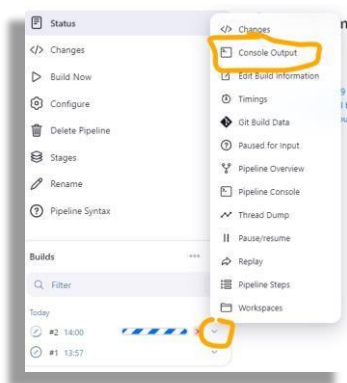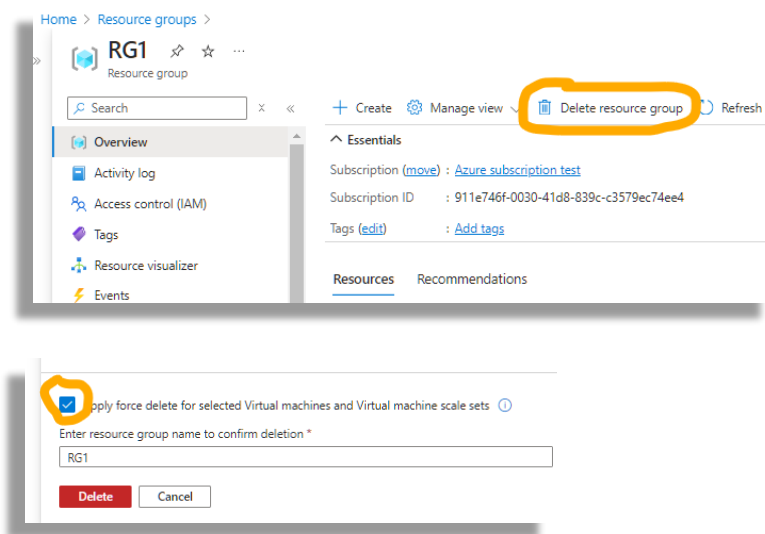

17. The run is waiting for approval to continue. Click on **Yes** to confirm the deletion ..
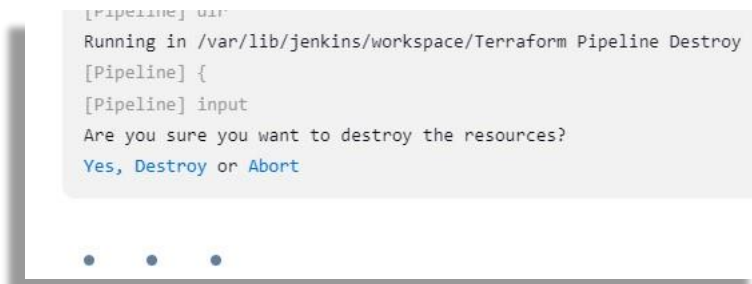
18. The destruction should now proceed. Switch to the Azure portal to verify the deletion of your resource group.

Task 10. Lab cleanup

1. In the Azure Portal, manually delete resource group "**RG1**" as this was created manually and is not therefore known to, nor configurable by, terraform…

*** **Congratulations, you have completed the final lab of the course. Destroy your Github repository at your own discretion** ***