QATIP Advanced – Automation Lab

Table of Contents

Lab Objectives

In this lab, you will:

- Clone and review a pre-built Terraform and automation repository.

- Bootstrap an Azure Storage-based remote backend for Terraform state.
- Deploy Jenkins (VM1) and AWX (VM2) onto Azure using Terraform.
- Configure Jenkins to authenticate to Azure via a Service Principal.
- Configure AWX with credentials, inventory, and job templates driven from Git.
- Provision a third VM (VM3) with Terraform and manage it via AWX and Ansible.
- Trigger end-to-end automation from Jenkins to AWX to configure and decommission a NGINX website.
- Destroy the lab resources and clean up the Terraform backend.

Teaching Points

This lab builds on your existing Terraform knowledge and extends it into a CI/CD and configuration management pipeline. Rather than running Terraform manually from your local machine, you will:

• Use a remote backend in Azure Storage so state is centralised and shareable.

• Use Jenkins to run Terraform in a controlled, repeatable way.

• Use AWX (the open-source upstream of Ansible Tower) to configure a workload VM using Ansible playbooks.

• Connect Jenkins and AWX so that infrastructure provisioning and configuration are triggered as one pipeline.

The goal is not simply 'can we create VMs?' – it is to show how Terraform, Jenkins, and AWX can work together to deliver full lifecycle automation: provision, configure, test, and tear down.

Before you begin

Ensure you have completed any prerequisite introductory labs and are comfortable running Terraform locally.

Verify you have access to an Azure subscription where you hold at least Contributor rights.

Confirm you have a GitHub account and can create public repositories.

On your lab workstation (Windows), ensure the following tools are installed and available:

- Visual Studio Code
- Terraform CLI (v1.5 or later)
- Azure CLI
- Git for Windows
- An SSH client (e.g. OpenSSH, included with recent Windows versions)

You should also make sure that no previous lab resources are still running in your subscription. If they are, destroy them before starting this lab.

Task1 Review the lab repository and architecture

In this task, you will examine the provided lab repository so you understand what the lab will deploy and how the pieces fit together.

1. git clone https://github.com/qatip/Jenkins-AWX-Kind.git to your local machine
2. Open Visual Studio Code.
3. From the menu, select File > Open Folder and open the folder where you cloned the lab repository (for example C:\Jenkins-AWX-Kind).
4. In the Explorer view, expand the folder structure and review the following key areas:
   - bootstrap – Terraform code to create the Azure Storage account and container used as the remote backend.
   - root (top-level .tf files) – Terraform code to deploy VM1 (Jenkins) and VM2 (AWX) and their networking.
   - jenkinsdemo_repo_files – files that will be pushed into your jenkinsdemo GitHub repository (Terraform for VM3 and Jenkins pipelines).

- awxdemo_repo_files – files that will be pushed into your awxdemo GitHub repository (Ansible playbooks and static website files).

Breaking it down

The architecture consists of three main virtual machines in Azure:

- VM1 – Jenkins; runs the Jenkins server and will pull code from your jenkinsdemo GitHub repository.
- VM2 – AWX; runs AWX on top of a Kubernetes-in-Docker (kind) cluster and pulls playbooks from your awxdemo repository.
- VM3 – Webserver; is created later by Terraform and then configured by AWX to host a NGINX demo website.

Jenkins triggers Terraform to create VM3, then calls AWX via its API to run an Ansible job to configure or remove the NGINX site. This demonstrates how infrastructure provisioning (Terraform) and configuration management (Ansible/AWX) can be orchestrated from a single CI/CD pipeline.

Task2 Bootstrap the Azure remote backend

In this task, you will use the bootstrap Terraform code to create an Azure Storage account and container to hold Terraform state files.

5. In Visual Studio Code, open the bootstrap folder within the lab repository.
6. Open the file terraform.tfvars in the bootstrap folder.
7. Set the subscription value to your Azure subscription ID and save the file.
8. Open an integrated terminal in VS Code (Terminal > New Terminal) and ensure the working directory is the bootstrap folder.

9. Run terraform init to initialize the working directory and download the Azure provider.
10. Run terraform apply and type yes when prompted to create the backend resources.
11. When the apply completes, review the output and make a note of the generated storage account name and resource group.
12. In the Azure portal, browse to Resource groups and confirm the backend resource group and storage account have been created.

Breaking it down

By default, Terraform stores its state in a local file named terraform.tfstate. In collaborative or automated environments this is not ideal, because state must be centralised and reliable. In this lab you are using an Azure Storage account as a remote backend, which allows Jenkins and other tooling to read and update state safely. The bootstrap code is a small, one-time Terraform configuration whose only job is to create the storage account and container for later use.

Task3 Deploy Jenkins (VM1) and AWX (VM2)

In this task, you will deploy the Jenkins and AWX virtual machines using the root Terraform configuration and wire them to use the remote backend you just created.

13. Return to the root of the lab repository in VS Code (where main.tf and terraform.tfvars reside).
14. Open main.tf and update the backend block so that storage_account_name matches the backend created in Task2.
15. Open terraform.tfvars at the root level and view the default values. Update the subscription value with your subscription. Leave other values unchanged. For the lab '*' is used for allow_inbound_cidr, but note that this is not recommended for production.
16. Save the changes.

17. In the integrated terminal at the root of the repo, run **terraform init**. Confirm that Terraform is now configured to use the azurerm backend.
18. Run **terraform plan** and review the resources that will be created. You should see a resource group, a virtual network, a subnet, network security rules, and two Linux virtual machines with public IP addresses.
19. Run **terraform apply** and type **yes** when prompted. The deployment will take several minutes as Azure provisions the VMs and AWX installation begins in the background.
20. When the deployment completes, review the Terraform outputs. Make a note of:

    - The public IP address for VM1 (Jenkins)
    - The public IP address for VM2 (AWX)
    - Any example SSH commands provided.

21. In the Azure portal, verify that both virtual machines are running and that their public IP addresses match the values output by Terraform.

Breaking it down

The root Terraform configuration is responsible for deploying the shared infrastructure and the two 'controller' VMs that will orchestrate the rest of the lab. It uses the AzureRM provider to define a resource group, virtual network, subnet, network security group, and two Linux virtual machines. Cloud-init scripts are passed to each VM so that Jenkins and AWX are installed automatically when the VMs first boot. This pattern—using Terraform to create the base infrastructure and cloud-init or similar mechanisms to bootstrap the software—is very common in real-world environments.

Task4 Configure Jenkins

In this task, you will complete the initial Jenkins setup and configure it to authenticate to Azure using a Service Principal.

22. Obtain the initial Jenkins password by running the command output at the end of Task3. Repeat until the password is show.

23. From your lab workstation, open a web browser and navigate to **http://<Jenkins-IP>:8080** using the public IP address of VM1 noted in Task3.
24. Log into Jenkins as '**admin**' and using the generated password.
25. Choose the option to install suggested plugins and wait for the plugin installation to complete.
26. When prompted, create the initial Jenkins admin user (record the username and password somewhere safe for the duration of the lab).
27. Once the Jenkins dashboard appears, leave the browser open.
28. Using your VSC terminal, create a Service Principal (SP) for Terraform using:

    ```
    az ad sp create-for-rbac --name terraform-sp-<suffix> --role
    Contributor --scopes /subscriptions/<sub_id>
    ```

    replacing **<suffix>** with a short unique string and **<sub_id>** with your subscription ID.
29. Record the **appId**, **password**, and **tenant** values returned. These will be used as Jenkins credentials.
30. In Jenkins, navigate to Manage Jenkins > Credentials and add the following credentials as Secret text entries in the global scope:
    - **azure_sp_client_id** – value is the appId from the SP.
    - **azure_sp_client_secret** – value is the password from the SP.
    - **azure_tenant_id** – value is the tenant from the SP.
    - **azure_subscription_id** – value is your Azure subscription ID.
31. Add a new Jenkins credential of type Secret file named **vm-pubkey-file** and upload your local SSH public key file (azure_automation_rsa.pub).

Breaking it down

Jenkins needs a way to authenticate to Azure when running Terraform. The recommended pattern is to use an Azure Service Principal (an application identity) with a specific role and scope, rather than relying on your personal credentials. In this lab, you store the Service Principal values as Jenkins credentials so that pipeline code can reference them safely without hard-coding secrets. You also upload the SSH public key so that Terraform can inject it into VM3 when that VM is created later.

Task5 Configure AWX

In this task, you will log into AWX, create an API token, and set up the credentials and inventory needed to manage VM3.

32. In your web browser, open a new tab and navigate to the AWX URL. This is **http://<AWX-IP>:30080** where <AWX-IP> is the public IP of VM2.
33. Log in with the default credentials **admin/ChangeMe123!** and accept any initial prompts.
34. Click on your user profile in the top-right corner and choose Tokens. Create a new token for use by Jenkins and copy the token value. This value will only be shown once.
35. In Jenkins, navigate to Credentials and add a new credential of type secret text with an id of **awx_api_token**. Paste in the token you just created. This will later be referenced by Jenkins when calling the AWX API.
36. Back in AWX, create a credential of type Machine. Set the privilege to Write and the username to '**azureuser**' Browse to and select your private SSH key (azure_automation_rsa) from your local machine.
37. Navigate to Inventories and create a new inventory named **DynamicHosts**. For now you do not need to add any hosts manually; the dynamic host will be supplied by Jenkins when triggering jobs. Make a note of the **Inventory ID**, as shown in the browser URL for later use.

38. Navigate to Projects and add a new project '**demo**' that pulls from your awxdemo GitHub repository. Use Git as the source control type and ensure the URL points to your awxdemo repository.
39. Sync the project and confirm the Last Job Status shows Successful.
40. Navigate to Templates and create two job templates:
    - **Deploy NGINX Website** – uses the **DynamicHosts** inventory, the **azureuser** credentials, and the **site.yml** playbook from your awxdemo project. Make a note of the **Template ID** as shown in the browser URL for later use.
    - **Remove NGINX Website** – uses the same inventory and credential but runs **remove-nginx.yml** instead.

Breaking it down

AWX acts as the control plane for Ansible automation. It needs three core pieces of information: where to run automation (inventories), how to authenticate (credentials), and what to run (projects and job templates). In this lab, the hosts for the inventory are not hard-coded; instead, Jenkins will provide the IP address of VM3 dynamically when triggering jobs. This is typical of cloud environments where servers are frequently created and destroyed.

Task6 Configure the Terraform backend for the VM3 deployment

In this task, you will update the Terraform configuration that will live inside your jenkinsdemo repository so that it uses the same remote backend and subscription as the main lab deployment. This ensures that when Jenkins runs Terraform for VM3, it writes state to the Azure Storage backend you created earlier, rather than to a local file.

41. Open **jenkinsdemo_repo_files/terraform_vm/main.tf** and update backend block so that the storage account matches the backend you created in Task2

42. Open **jenkinsdemo_repo_files/terraform_vm/terraform.tf** and update <your subscription here> with your subscription

43. Open each of the 4 Jenkins files in jenkinsdemo_repo_files/ and update the **<JENKINS REPO URL>** placeholder (line 17) with your jenkinsdemo repo url.

44. Open **JenkinsAWX** in jenkinsdemo_repo_files/ and update the **2** **<AWX_VM_PUBLIC_IP>** place holders with the public IP address of your AWX instance (VM2). Update the 1 place holder **<YOUR_INVENTORY_ID>** with the **Inventory ID** you noted in Task5 and the 1 placeholder **<YOUR_TEMPLATE_ID>** with the **Template ID** you noted in Task5.

Breaking it down

The terraform_vm configuration is the part of your lab that Jenkins uses to create and destroy VM3. It needs to write its state to the same Azure Storage backend as the rest of the lab so that state is centralised and consistent. By updating the backend block and subscription variables here, you avoid situations where Jenkins writes local state in its workspace or accidentally targets a different subscription from the one you used when bootstrapping the environment. The Jenkins files updates linked the configurations with the Jenkins repo and the AWX Inventory and Template, allowing Jenkins to register the new VM with AWX and trigger the playbook that will deploy and install nginx.

Task7 Prepare the GitHub repositories

In this task, you will create and populate two GitHub repositories that Jenkins and AWX will use as their source of truth.

45. On GitHub, create a new public repository named jenkinsdemo. Do not add any initial files.

46. Create a second public repository named awxdemo.
47. On your lab workstation, clone the jenkinsdemo repository into a convenient folder.
48. Copy the contents of the **jenkinsdemo_repo_files** folder from the lab repository into your local jenkinsdemo working directory.
49. Commit and push the changes so that the JenkinsPlan, JenkinsApply, JenkinsDestroy, JenkinsAWX pipeline files and the terraform_vm folder are present in your GitHub jenkinsdemo repo.
50. On your lab workstation, clone the awxdemo repository and copy the contents of the **awxdemo_repo_files** folder into it.
51. Commit and push the site.yml, remove-nginx.yml, and any supporting files (such as index.html and logo.png) so they are available to AWX via Git.

Breaking it down

Both Jenkins and AWX are designed to treat version control systems such as Git as the primary source of configuration. Jenkins pulls pipeline definitions and Terraform code from the jenkinsdemo repository, while AWX pulls Ansible playbooks and static content from the awxdemo repository. Treating Git as the single source of truth makes it easier to review, test, and roll back automation changes.
[Screenshot: GitHub showing the jenkinsdemo and awxdemo repositories populated]

Task8 Create and run the Jenkins pipelines

In this task, you will create four Jenkins pipelines from the supplied Jenkinsfiles and run them to provision VM3, configure it via AWX, and then remove it again.

52. In Jenkins, click New Item and create a new Pipeline job named PlanPipeline.

53. Configure the pipeline to pull from the **main** branch your jenkinsdemo GitHub repository and point it at the **JenkinsPlan** file. Save the job.
54. Repeat the process to create three more pipelines:
    - ApplyPipeline – uses the **JenkinsApply** file.
    - DestroyPipeline – uses the **JenkinsDestroy** file.
    - AWXPipeline – uses the **JenkinsAWX** file.
55. Save the AWXPipeline changes.
56. Run the PlanPipeline. Confirm from the console output that Terraform initializes successfully, validates the configuration, and produces a plan with the expected actions but does not apply any changes.
57. Run the ApplyPipeline. This should create VM3 (the webserver VM) using the terraform_vm configuration. Make a note of the VM3 public IP address from the pipeline output or from the Azure portal.
58. Once ApplyPipeline completes, use the Azure portal to confirm VM3 is created.
59. The Apply pipeline demonstrates using Jenkins in isolation. AWX was not notified of the VN creation and therefore did not configure it.
60. Run the DestroyPipeline. This will remove the VM ahead of re-deploying with AWX integration
61. Once the VM is destroyed, run the AWXPipeline. This pipeline should deploy the VM and register it with AWX which will run the playbook to install and configure nginx.
62. In the AWX web UI, navigate to Jobs and confirm that the Deploy NGINX Website job has run successfully.
63. Browse to http://<VM3-IP> and confirm that the NGINX demo site (with your lab's index.html content) is now displayed.
64. In AWX, manually run the Remove NGINX Website job template and confirm that the page content changes to show that the site has been decommissioned.

Breaking it down

The four Jenkins pipelines separate concerns clearly:

- PlanPipeline checks what Terraform would do without making changes.
- AWXPipeline hands off configuration to AWX using its REST API, passing in the dynamically discovered IP address of VM3.
- DestroyPipeline removes VM3 when you no longer need it.

This separation mimics common CI/CD patterns where planning, applying, configuration, and cleanup are distinct stages. By using Terraform outputs and environment variables, Jenkins can pass just enough information to AWX to let Ansible do its job without hard-coding host details.

Task9 Destroy the deployment and clean up the backend

In this final task, you will remove VM3 using Jenkins, destroy the Jenkins and AWX infrastructure, and then remove the backend storage account created earlier.

65. In Jenkins, run the DestroyPipeline and confirm from the console output that VM3 has been destroyed.
66. In the Azure portal, verify that VM3 no longer appears in the list of virtual machines.
67. On your lab workstation, return to the root Terraform directory for this lab.
68. Run terraform destroy, review the plan, and type yes when prompted. This will remove the resource group, virtual network, and the two controller VMs (Jenkins and AWX).
69. In the Azure portal, confirm that the resource group used for Jenkins and AWX has been deleted.
70. Return to the bootstrap folder in your lab repository.

71. Run terraform destroy in the bootstrap folder and confirm that the backend storage account and container are deleted.
72. Finally, in the Azure portal, verify that the backend resource group has been removed and that no lab-related resources remain.

Breaking it down

Destroying lab resources is an important habit, especially when working in cloud environments where resources incur cost. You use Jenkins to destroy the workload VM (VM3) via the DestroyPipeline, then use Terraform directly to remove the controller infrastructure and backend. In a production environment, you might also automate backend creation and destruction or retain it for ongoing use across multiple deployments.

Congratulations, you have completed this lab