Deployment Pipeline Training

INTRODUCTION

This training will provide guidance to build a simple deployment pipeline using the following tools:

- Amazon Web Services (AWS)
- Terraform
- Jenkins
- GitHub

When you have finished, you should be able to open a Jenkins interface and create a job that will build a personalized environment (as opposed to dev or prod) based on Terraform code (as opposed to building the environment directly in the AWS console).

Jenkins will be able to download the application files as well as build instructions from GitHub. Those instructions will tell Jenkins how to compile and deploy your application, as well as how to use Terraform (and associated Terraform objects) as infrastructure-as-code (IAC) to build all the necessary components in AWS. For example, it will include instructions to create and populate one of the AWS simple storage service (S3) buckets with the client-side files of your application.

We developed this training on top of AWS's "WildRydes" tutorial, where you build an application using the AWS console.

Why not stop there? *ADO best practice states that we do not build applications directly in AWS because there is no way to track changes to settings and configurations. If the application needs to be re-built in a new environment, using an IAC (deployment pipeline/devOps) process is our standard procedure so that the environments are exactly alike.*

However, it is important that you complete the AWS "WildRydes" tutorial before you start this training. It will make you familiar with the various services and associated relationships before working with them only in code. (The AWS course typically takes about four hours to complete.)

It is also recommended that you complete Hashicorp's Terraform training as well. It is pretty simple, but again, makes you familiar with the basics of the tool and file creation. (Hashicorp's Getting Started guide takes around 75 minutes to complete.)

For this training, you will receive the overall structure that is needed, as well a template Jenkins file, which you will need to complete.

PREP WORK

1. If you have not already done so, add Duo MFA to your mobile phone:
   https://kb.northwestern.edu/62612

2. Confirm that you are set up as a user on AWS Sandbox:
   https://aws.northwestern.edu

3. Build the AWS WildRydes training course using the AWS console:
   https://aws.amazon.com/getting-started/projects/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/
   NOTE: This takes approximately four to five hours to complete.

4. Read the Terraform start-up documentation and complete all Get Started – AWS topics:
   https://learn.hashicorp.com/terraform?track=getting-started#getting-started

   NOTE: This takes approximately 75 minutes to complete.

5. Review the Terraform user documentation:
   https://www.terraform.io/docs/providers/aws/index.html
   NOTE: Learn the terms in the sidebar; you will spend a lot of time on this page. You should spend approximately 30 minutes review. Focus the resource types used in Wild Rydes, such as S3, Dynamo, APIgateway, etc.

6. Review the EACD Cloud Documentation, which gives NUIT-specific guidance:
   https://nit-administrative-systems.github.io/AS-CloudDocs/

   NOTE: This takes approximately 90 minutes to review.

INSTALL TERRAFORM

NOTE: This section assumes you are using a Windows machine.

1. Use PowerShell (as admin) to install windows subsystem:
   https://docs.microsoft.com/en-us/windows/wsl/install-win10

2. Read about Linux distribution:
   https://www.howtogeek.com/132624/htg-explains-whats-a-linux-distro-and-how-are-they-different

3. Go to search on your Windows screen and enter "MS store".
4. Within the store, search for "Linux", select "Ubuntu", and launch.
5. Open Ubunti.
6. Install unzip tool:
   sudo apt-get install unzip

7. Download latest version of the terraform:
   sudo wget https://releases.hashicorp.com/terraform/x.x/fileName.zip

8. Extract the downloaded file archive:
   unzip fileName.zip

9. Move the executable into a directory searched for executables:
   sudo mv terraform /usr/local/bin/

10. Test it:
    terraform –version

    NOTE: The return should display the version of Terraform.

INSTALL AWS-ADFS UTILITY

Note: This utility must be run every couple of hours, as it acts are the sign-in broker between Terraform and AWS. It is what allows you to import from Terraform to AWS without local sign-in.

1. Read about how NUIT uses it:
   https://nit-administrative-systems.github.io/AS-CloudDocs/iac/terraform-concepts.html#developing-testing-iac

2. Open Ubunti
3. Install the utility:
   sudo -H pip install aws-adfs

   If you get the error "fatal error: gssapi/gssapi.h: No such file or directory"

   Then you will need to download the following APT library:
   sudo apt-get install libkrb5-dev

4. To run the utility, the username format will be ads\netid:
   aws-adfs login --adfs-host=ads-fed.northwestern.edu --provider-id urn:amazon:webservices --region us-east-2

   NOTE: you will then need to accept the DUO MFA on your phone.

NOTE: the utility code listed above will work for the sandbox account, assuming that you have not run this utility code before. The first time you run this code, it sets the account you select as the default. Eventually, you will need to set up profiles for each AWS account (NonProd, Prod, Sandbox, etc.). To learn more about how to set up profiles for the utility, see the AS Cloud Docs website (https://nit-administrative-systems.github.io/AS-CloudDocs/).

INSTALL AWS COMMAND LINE AND BASIC COMMANDS

1. In Ubuntu, enter:
   sudo apt-get update

2. Then:
   sudo apt-get install awscli

IMPORTANT NOTE!

To access files in Ubuntu, you will need to change directories using a mount. For example:
/mnt/c/Users/jem3523/ADO_projects/terraform

Most Common Terraform Commands

- terraform version — to confirm version
- terraform init — to create .terraform directory
- terraform fmt — enables standardization of files
- terraform validate — checks for errors in files
- terraform plan — execution plan before applying it
- terraform apply — applies changes to AWS
- terraform show — shows current state; can make edits
- terraform destroy — deletes the ENTIRE instance

SET UP DIRECTORIES AND BLANK FILES

1. Go to https://github.com/NIT-Administrative-Systems/ADO-EACD-DevOpsTraining and clone the repository to your local machine.

2. In the IDE of choice (Visual Studio Code), review the following directories:
    - .jenkins (for your jenkins instructions)
    - clientSideFiles (for your website files, which go into an S3)
    - dev (sample environment files)
    - modules (for your terraform files)
        i. files_lambda_S3 (temp for your lambda object)
        ii. templates (for your templates)

3. In the "modules" directory, create a blank .tf file for each of the major resource types:
    - APIgateway.tf
    - Cognito.tf
    - Dynamo.tf
    - IAM.tf
    - Lambda.tf
    - S3bucket.tf

NOTE: the "provider.tf" file is provided to you with an example of a region variable included.

HINTS AND GUIDELINES

This is the core of the training, and it is very self-exploratory (meaning that there will not be step-by-step instructions). However, there are some hints and guidelines.

Recommended Steps

1. Review the appendix diagram of how the resources will fit together. Note how the files inter-relate.

2. Cluster the code into the 7 resource-type files listed above. The Terraform documentation will give examples that show all required code, but best practice is to split that code by resource type. For example, all IAM policies in the IAM.tf file.

3. Use the original WildRyde instructions and create your seven terraform files using the same sequence. Try using the same test steps at the end of most AWS training sections.

4. Model the resource from the WildRyde instructions by looking up that resource in the terraform documentation (https://www.terraform.io/docs/providers/aws/index.html) and using the appendix diagram to see which core recourses are referenced by other associated code. ***This is really the key to the entire training, so really explore the documentation samples.***

   IMPORTANT NOTE! As of March 2020, the documentation was not up to Terraform V.12. The main difference is that how syntax around variables, so you may want to research that.

5. Try to build the course in phases:
   - Basic build via command line on your local machine (Ubuntu) without embedded variables for app_name, environment, and region.
   - Add variables so that it can build in a personalized environment. (See the variables in the */dev/variables.tf*).
     IMPORTANT! Copy the "dev" folder and rename the folder / modify the files to make an environment based on your initials (for example: jem). *This is important because we will have multiple people training at the same time!*
   - You will also need to add the following **tag** attribute to each of the core seven resource types:
     tags = {"app_name"= "${var.app_name}", "env"= "${var.env}"}
     NOTE : these tags allow an admin to calculate the cost of the application based on the environment. They should be added to all AWS applications going forward.
   - Then create GitHub repo with your completed files (https://github.com/NIT-Administrative-Systems).
     NOTE: In GitHub, you will need to go to "Manage Access" for your repo and add "awsCloudOpsCJT" as a read-only user. This allows Jenkins permission to access the repo.
   - Finally, go into the sandbox Jenkins interface (https://as-ado-sbx-build.jenkins.northwestern.edu:8443/). Click on the "Training" folder, then click "New Item" in the left nav. Give the pipeline a name, then go to the bottom of the page and enter "WildRydes_Master" into the copy-from field. Then:
     i. Add the name of your personalized environment (based on your initials) to the BUILD_ENV choice parameter.
     ii. In the pipeline section, update your GitHub repo URL and save.

6. To run the full deployment, in the Jenkins interface, go to the pipeline you just created, then click “Build with Parameters”. Change the environment drop-down to your personalized environment and click “Build”.

Hints

- There are two kinds of AWS IAM role policies:
    - managed (attached) is part of a library that you are referencing
    - in-line are created as part of the role and can only be used for that specific role

- The two primary Terraform elements are:
    - Resource: elements you are creating
    - Data: stored elements you are referencing (note that you include “data” at the start of referencing data elements; however, resources do include “resources”)
      For example:
      "${data.template_file.temp_policy_clientbucket.rendered}"
      "${aws_s3_bucket.wildrydes_clientbucket.website_endpoint}"

- In AWS console, the .js file for a lambda (which is server side) is in a different S3 bucket than the client-side web files. This will require a bucket object be created. It will also require that the terraform code zip the lambda file in order to upload it to the S3 bucket for lamdbas. Research the following:

    - aws_s3_bucket
    - aws_s3_bucket_object
    - data template_file
    - data archive_file

- In AWS console, you manually uploaded the files that create the client-side website. You don’t upload these files using Terraform; you will need to modify the Jenkins instruction file to put the files in place as part of the Jenkins steps. Research the following:
    - aws s3 sync
    - Permissions on the bucket that will need to be in the policy in order for the files to be allowed to upload.

- For both S3 buckets, you will find policy templates in the template folder. You will need to create the template file code *before* you create the bucket code (which will include a rendered version of the template).

- In the original WildRydes, you have to configure the config.js file manually. You will need to create this file dynamically in terraform (based on the templates/config_for_S3.tpl) and write it to the correct S3 bucket and js sub-directory in AWS. (Explore "template_file" data type and "aws_s3_bucket_object" resource type.)

WHAT DOES SUCCESS LOOK LIKE?

First, before building via Jenkins, you should be able to run terraform validate, plan, and apply without error.

Follow this up by a quick review within AWS. Do all the resources look exactly like the ones you built using the AWS console to build Wild Rydes? (HINT: compare the instructions from Wild Rydes to what you see now. Does each resource configuration match?)

Then launch the Wild Rydes website, just as you did to check the results when you build it using the AWS console. It should launch and work exactly the same.

Now build again via Jenkins. Make the same checks as above.

Provider & Region
aws_dynamodb_table
AWSLambdaBasic ExecutionRole
aws_s3_bucket (lambdas)
aws_s3_bucket (clients)
aws_iam_role_ policy (inline policy)
aws_iam_role_ policy_attachment (managed policy)
aws_s3_bucket_object
includes
aws_cognito_ user_pool_client
aws_cognito_user_pool
aws_lambda_ permission
aws_api_gateway_ integration
aws_api_gateway _deployment
aws_api_gateway_ rest_api
aws_api_gateway_ authorizer
aws_api_gateway _method
aws_api_gateway_ resource