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

We developed this training as an extension 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 they have not been checked into Github as Terraform code. If the application needs to be re-built in a new environment, using an IAC (infrastructure-as-code deployment pipeline/devOps) process is our standard procedure so that the environments are exactly alike.*

Because the AWS WildRydes training will make you familiar with the various services and associated relationships before working with them only in code, it is important you complete the AWS WildRydes tutorial before you start the *Terraform* part of this training. It. (The AWS course typically takes about four hours 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.

When you have finished, you should be able to open a Jenkins console 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). *See Hints and Guidelines, Recommended Step 5.*

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.

PREP WORK

1. If you have not already done so, add Duo MFA to your mobile phone:
   https://kb.northwestern.edu/page.php?id=62610

2. Confirm that you are set up as a user on AWS Sandbox:
   https://aws.northwestern.edu
   NOTE: If you do not have access to these, reach out to the EACD – CloudOps team.

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: We recommend that you take notes during this process, including saving the site address and log-in credentials for comparison later in the process. You will most likely need to change the table name within the WildRydes sample to be unique; this will require that you review the template code AWS provides with the training to change the table name when it is referenced. Building WildRydes 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: It is 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. Note that the Terraform training will walk you through installing Terraform; during installation, also refer to section below entitled "Install Terraform" as it will give you additional guidance. This will give you brief introduction to creating infrastructure/building resources, initializing, applying. Hashicorp's Getting Started guide takes around 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. The steps guide you through installing Ubuntu for Linux on Windows. **Do not install the Windows version of Terraform!** We have not had good results in using the Windows version and using Linux keeps steps later in the process consistent.

The following steps cover how to prepare then install the Ubuntu app through the MS Store.

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. If you have questions about the appropriate version, contact the ADO-EACD team for guidance.
5. Open Ubuntu.
6. Install unzip tool:
   sudo apt-get install unzip

7. If you're having trouble installing packages, check the subsystem clock. You made need to set the clock manually with a command like:
   sudo hwclock --hctosys

8. Download latest version of the terraform:
   sudo wget https://releases.hashicorp.com/terraform/x.x/fileName.zip
   NOTE: To figure out which the latest version of terraform, go to https://www.terraform.io/downloads.html. Copy link address for the 64-bit version needed. In this case, choose the Linux 64-bit version, as we are using the Ubuntu app.
   FOR EXAMPLE:
   sudo wget
   https://releases.hashicorp.com/terraform/0.12.24/terraform_0.12.24_linux_amd64.zip/

9. Extract the downloaded file archive:
   unzip fileName.zip
   FOR EXAMPLE: *unzip terraform_0.12.24_windows_amd64.zip*

10. Check to confirm that files have been downloaded and extracted by using the list command: ls
    NOTE: If you see terraform.exe instead of just terraform, you have mistakenly downloaded the Windows version. Delete it and download the linux version.

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

12. 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 as the sign-in broker between Terraform and AWS. It is what allows you to import from Terraform to AWS without local sign-in. For those unfamiliar with linux, both Terraform and AWS-ADFS require superuser access to run properly.

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

2. Open Ubuntu
3. Install the utility:
   sudo -H pip3 install aws-adfs
   NOTE: pip3 refers to python 3. In the future, this could change to reflect a higher version.

   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, enter the code below:
   aws-adfs login --adfs-host=ads-fed.northwestern.edu --provider-id urn:amazon:webservices --region us-east-2

```
scd691@JDYNRQ2:~$ aws-adfs login --adfs-host=ads-fed.northwestern.edu --provider-id urn:amazon:webservices --reg
ion us-east-2
2020-05-05 18:13:54,897 [authenticator authenticator.py:authenticate] [98-MainProcess] [139680246269760-MainThre
ad] - ERROR: Cannot extract saml assertion. Re-authentication needed?
```

5. You will be prompted to enter your username (ads\netid) and password (netid password) and accept DUO MFA on your phone.

6. You will be prompted to choose the “role” you would like to assume (the account you would need to log into). Enter the number corresponding to the appropriate role (sbx) on the list. **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/iac/terraform-concepts.html#developing-testing-iac).

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 local Windows files from within Ubuntu, you will need to change directories using a mount. If you want to terraform commands on the files in the repo you create during the training, change directories in Ubuntu with cd /mnt/.

FOR EXAMPLE: To access files in Ubuntu, you would enter:
cd /mnt/c/Users/xyz123/ADO_projects/terraform

Most Common Terraform Commands
You can enter *terraform help* in the command line to view a full list of available commands. Some of the most common commands include:

- 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 fork the repository to your local machine.
   NOTE: Please do not clone the repo, as you should not be uploading into the course materials.

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)

   NOTE: Remember that you use VSC for working with the code and Ubuntu for executing (validate, plan, apply). Once you have built out your Jenkins file, you will execute via Jenkins console and will no longer use Ubuntu.

   NOTE: Some users install the VSC Terraform extension by Michael Olenfalk. You can also install the language server to reduce TF code showing up as errors/problems. https://medium.com/@gmusumeci/how-to-install-update-enable-and-disable-language-server-for-terraform-extension-in-visual-116e73f58722

3. Remember to switch to your mounted directory and run *terraform init* within your personalized directory (called "dev" as a sample environment folder above).

4. 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.

NOTE: additionally, add a basic named *.terraform-version* to the root folder that contains the following one line of code:

```
1 latest:0.12
```

Although we are using V12, much of the documentation for Terraform is still in an earlier version.

For information about upgrading Terraform files, see https://nit-administrative-systems.github.io/AS-CloudDocs/iac/tf-upgrading.html#strategy

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 add the TF code to your seven terraform files using the same sequence. Try using the same test steps at the end of most AWS training sections. If the comparison gets confusing, it may be helpful to copy the WildRydes steps text from the website into a document, then add what will need to be added in Terraform code to accomplish the same outside the AWS console.

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 resources are referenced by other associated code. ***This is really the key to the entire training, so really explore the documentation samples.***

   REMINDER 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. For more information see: https://nit-administrative-systems.github.io/AS-CloudDocs/iac/tf-upgrading.html#strategy

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. What do we mean by "personalized environment"? If each person using this manual uses "dev" or "qa", then they will over-write each other's work in AWS, so just use your initials or something unique to create your personalized environment

for this exercise. (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 invite "awsCloudOpsCJT" as a read-only user. This allows Jenkins permission to access the repo.
- If you are off-campus, confirm that you are logged in via Global Connect VPN before accessing the Jenkins console.
- Finally, go into the sandbox Jenkins console (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. The "dev" and "qa" environments should already be there. Since you are creating a personalized environment, you won't need them them.
  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 or updating
  - **Data**: read-only elements already within AWS that you are referencing (note that you include "data" at the start of referencing data elements; however,

resources do not include "resources")

When you are creating a resource, the first two parameters you give it are 1) the type of resource, and 2) a kind of nickname for that resource that is used by Terraform during the build. Once Terraform executes and populates AWS, those nicknames go away. They are only used within Terraform, but Terraform relies on them as short cuts.

If you were to create a new resource using Terraform, you use the "resource" element. For a new S3 bucket, the first line of your resource code would look something like the following:

```
resource "aws_s3_bucket_object" "my_new_bucket"
```

But if you need to reference something in your Terraform files that already exists in AWS, you use the "data" label (instead of "resource") to give that AWS resource a nickname while you work with it within your Terraform file, then it can be used within your Terraform code along with all the resources that are creating.

For example, let's say that you need to refer to a bucket that already exists in AWS. In your Terraform code, you would start with "data" so that Terraform knows that it needs to reach back into AWS. Then you enter the type of resource and the nickname. Finally, you need to reference exactly which bucket in AWS that is being referenced.

```
data "aws_s3_bucket" " existingBucketNickname"
{bucket = "my_existing_bucketname"}
```

NOTE: you do not use "data" within the WildRydes exercise, but the concept is very important.

- You may come across instances where lambda files have been created in their own repos in GitHub. If this is the case, think through the best way to sequence multiple Jenkins builds. Typically, you would build the lambda build before you run a full IAC build. This will require a bucket and bucket object be created in addition to the function itself and any role that is associated with the function. 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
    - aws_iam_role
    - aws_lambda_function
    - 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 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). There have been some changes in V12 in this area, to see more, go to: https://www.terraform.io/docs/configuration/functions/templatefile.html

- Because you will likely want to test the site before running Jenkins, you will need to upload the necessary files to your s3 bucket through your terraform file. Studying content_type may help.

- 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 from your personalized folder.

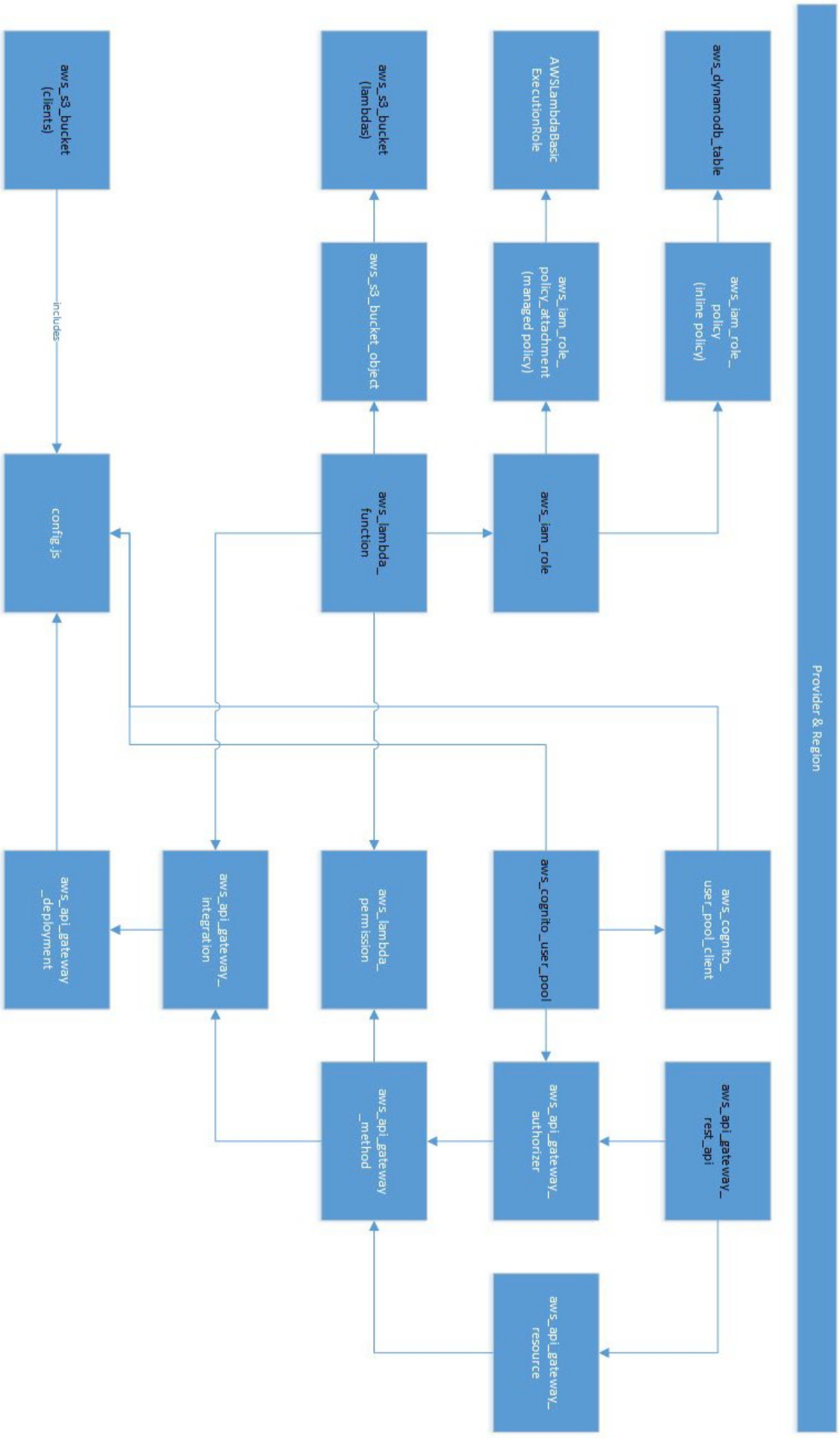
Follow this up by a quick review within AWS. Do all the resources look exactly like the ones you built using the AWS console to build Wild Rydes? (HINT: compare the instructions from Wild Rydes to what you see now. Does each resource configuration match?)

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.

Common Issues

1. I am getting a CORS error when trying to request a unicorn.
    a. Terraform does not add CORS headers to the API Gateway by default. You will need to add these to any method responses and/or integration responses that need them
    b. An OPTIONS method must be defined along with a POST method
2. I am getting a 500 server error or ResourceNotFound error from my lambda
    a. Check that your names are consistent. Did you remember to change the database name defined in requestUnicorn.js to match the one defined in Dynamo.tf?


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