02. Getting Started

Contents

- Preparing an AWS account to use with Terraform
- Installing Terraform
- Initializing Working Directories
- Configuring Terraform CLI
- Terraform CLI Overview

Preparing an AWS account

Requirements

- Note : In this course, student accounts are created with AWS IAM Identity Center (formerly AWS SSO – Single Sign On)
  - A separate PDF document will explain how to use Terraform on this environment
- Requirements to manage / provision AWS infrastructure with Terraform:
  - AWS Account(s)
  - IAM User (or role!) with permissions for the actions to be performed (by Terraform)
    - Terraform uses the AWS Go SDK to perform its operations on AWS
  - Terraform with AWS Provider - with several options to authenticate with AWS
    - AWS provider installed when performing "terraform init"
- Permissions / IAM – general info
  - AWS IAM Documentation
  - Excellent practical resource (AWS re:invent 2019 – Getting Started with IAM Identity)

Creating AWS Account

- Note: Not applicable to CTA course – kept here for completeness
- Standalone Account
  - Create Account -
  - Root user -> Create IAM user with administrator permissions
  - Admin IAM user -> Create IAM user or role with restricted permissions for Terraform (principle of minimum privilege)
- AWS Organizations (our case in this course)
  - Create "member" Account from Organizations management account
  - Recommended: use roles to work member accounts
  - Alternative: create IAM user(s) in member account
- (course specific) Set the default region for the console to "eu-west-1"

AWS Provider - Authentication

- Terraform uses the AWS Go SDK to communicate with AWS (refresh, provision, etc.).
  - This communication requires authentication, with an AWS IAM user and two parameters that identify and authenticate it:
  - aws_access_key_id and aws_secret_access_key

- Terraform AWS provider documentation specifies several methods for Terraform to authenticate with AWS
  - Static credentials – hard coded access and secret keys (not recommended)
  - Environment Variables for the access and secret keys (used e.g. with Terraform Cloud)
  - Shared credentials / configuration file - use AWS CLI credentials file to obtain the access and secret keys (used in this course – see next slides)
  - CodeBuild, ECS and EKS Roles -- if running Terraform inside these AWS services
  - EC2 Instance Metadata Service (IMDS, and IMDSv2) - if running Terraform inside an EC2 instance

AWS CLI Named Profiles and Credentials

- Terraform will attempt to use the AWS user profile specified in the "providers" block when performing API calls to AWS. How to create that profile?
- For example, assume we will use a specific IAM user (e.g. "tfadmin1") as the profile for TF to connect to AWS
- Add "tfadmin1" to the config file:
  - ~/.aws/config (Linux or Mac)
  - %USERPROFILE%\.aws\credentials (Windows)
  - Optionally, set region and other parameters
  - Optionally add this profile as [default]
- Generate credentials for IAM user "tfadmin1" in AWS console (aws_access_key_id and aws_secret_access_key) and copy to the credentials file, under a [tfadmin1] profile
  - ~/.aws/credentials file (Linux or Mac)
  - %USERPROFILE%\.aws\credentials (Windows)
  - Optionally add as [default]

```
provider "aws" {
  region = var.region
  profile = "tfadmin1"    ## will normally use variables
  default_tags {
    tags = {
      environment = var.environment
      project = var.project
      created_by = "terraform"
    }
  }
}
```

```
$cat ~/.aws/config
(…)
[profile tfadmin1]
region = eu-west-1
output = yaml
```

```
$cat ~/.aws/credentials
(…)
[tfadmin1]
aws_access_key_id = Ayyyyyyyyyyyyyyyyyyyyyyyy
aws_secret_access_key = Xyyyyyyyyyyyyyyyyyyyyyyyyyy
```

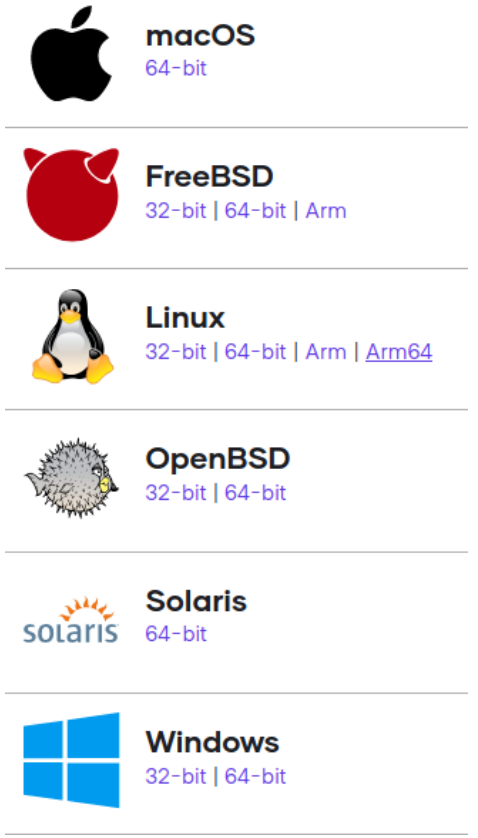
Installing Terraform

Terraform Deployment Options

- Terraform CLI (Standalone)
  - Installation : Linux, Windows, Mac
  - Configuration and tuning
- Terraform Cloud (same software as Terraform Enterprise)
  - *"Terraform Cloud is an application that helps teams use Terraform together. It manages Terraform runs in a consistent and reliable environment, and includes easy access to shared state and secret data, access controls for approving changes to infrastructure, a private registry for sharing Terraform modules, detailed policy controls for governing the contents of Terraform configurations, and more."* (HashiCorp description)
- Terraform Enterprise: self-hosted version of Terraform Cloud
  - On-Premises
    - On Line
    - Air-Gapped
  - Cloud Providers – AWS, Azure, GCP (Hashicorp provides Reference Architectures)
  - VMware
- (Azure Specific) Terraform also available on CloudShell for Bash or Powershell
  - CloudShell already includes terraform binary (may be a few weeks delayed from latest)
- Third Party Provisioning Environments (

Terraform CLI – Installation Overview

- Terraform packaged as single binary file: download, unzip to a directory included in path, and execute
- Download link in Terraform web site
  - Direct link to latest version
  - Contains links to checksums, changelog and older releases
- Available for
  - Windows (386, Amd64)
  - Linux:
    - Generic binaries (32-bit, 64-bit and Arm/Arm64)
    - Distribution packages (RHEL/CentOS yum/DNF and Debian/Ubuntu/apt)
  - Mac
  - FreeBSD
  - OpenBSD
  - Solaris
- Installation tutorials in HashiCorp "learn" site



Terraform CLI Installation - Linux

- Main distribution method: zipped binary files – unzip and move to destination directory
- Select / Create directory for installation (e.g. /usr/local/bin)
  - If not done earlier, ensure that installation directory is part of environment PATH variable
  - E.g. add path to .profile: `export PATH=$PATH:/selected/terraform/path`
- Extract 'terraform' executable from zipped file to installation directory
- No special privileges required – terraform can be installed to and run from a directory under the user's home directory
- Some users prefer using their Linux distribution package managers for smoother integration into their config management strategies. This involves adding Terraform-specific repository to the package manager. Distribution specific docs at:
  - Debian/Ubuntu (apt): https://www.terraform.io/docs/cli/instal/apt.html
  - RedHat (yum) : https://www.terraform.io/docs/cli/install/yum.html
- Test installation: run "terraform –version" & "terraform –help"

```
rafa@rp3:~$ terraform version
Terraform v1.2.4
on linux_amd64
rafa@rp3:~$ terraform -help
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure
```

Terraform CLI Installation - Windows

- Download zip file
- Define / Create folder for installation (e.g. c:\binutils)
- Extract .exe to installation folder
- If not done earlier, ensure that installation folder is in execution path
  - See instructions in Windows 10 documentation or in this StackOverflow link

Terraform CLI Overview

- Root of CLI documentation
  - Google or Duckduckgo search of commands such as "terraform cli show" normally yields the actual documentation page among the first search results.
- Autocompletion for bash / zsh environments available (see next slide)

```
rafa@rp3:~$ terraform -h
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init          Prepare your working directory for other commands
  validate      Check whether the configuration is valid
  plan          Show changes required by the current configuration
  apply         Create or update infrastructure
  destroy       Destroy previously-created infrastructure

All other commands:
  console       Try Terraform expressions at an interactive command prompt
  fmt           Reformat your configuration in the standard style
  force-unlock  Release a stuck lock on the current workspace
  get           Install or upgrade remote Terraform modules
  graph         Generate a Graphviz graph of the steps in an operation
  import        Associate existing infrastructure with a Terraform resource
  login         Obtain and save credentials for a remote host
  logout        Remove locally-stored credentials for a remote host
  output        Show output values from your root module
  providers     Show the providers required for this configuration
  refresh       Update the state to match remote systems
  show          Show the current state or a saved plan
  state         Advanced state management
  taint         Mark a resource instance as not fully functional
  test          Experimental support for module integration testing
  untaint       Remove the 'tainted' state from a resource instance
  version       Show the current Terraform version
  workspace     Workspace management

Global options (use these before the subcommand, if any):
  -chdir=DIR    Switch to a different working directory before executing the
                given subcommand.
  -help         Show this help output, or the help for a specified subcommand.
  -version      An alias for the "version" subcommand.
```

Terraform CLI – Autocomplete (bash / zsh)

- Documentation under "basic CLI features/Shell Tab-completion"
- Available for *bash* or *zsh*
- Install with *terraform -install-autocomplete*
- After installation, it is necessary to restart your shell or to re-read its profile script before completion will be activated.
- Remove with *terraform -uninstall-autocomplete*

```
rafa@rp3:~$ terraform
apply          fmt            import         output         refresh        test           workspace
console        force-unlock   init           plan           show           untaint
destroy        get            login          providers      state          validate
env            graph          logout         push           taint          version
rafa@rp3:~$ terraform state
list             pull             replace-provider  show
mv               push             rm
```

Initializing Working Directories

CLI: *terraform init*

- *terraform init* documentation
- The terraform init command is used to initialize a working directory containing Terraform configuration files.
- This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control.
- It is safe to run this command multiple times.
- After running *terraform init*, terraform scans the .tf files for provider information. It then downloads the required providers (say, *azurerm*) to specific subdirectories under directory *.terraform*
  - This behavior can be partially modified by specifying caching for plugins (see next sub-section on CLI configuration file)
- See example with basic configuration with AWS EC2 instances and Google Cloud Compute Instance.

Configuring Terraform CLI

Terraform CLI configuration file

- CLI configuration file configures per-user settings of the CLI behavior.
- Default locations:
  - Windows: terraform.rc in %APPDATA directory
  - Other systems: .terraformrc (in user's home directory)
- Or as specified in TF_CLI_CONFIG_FILE environment variable
- Details in terraform documentation
- Most settings refer to Terraform Cloud/Enterprise
- Most useful setting for Terraform CLI : "plugin_cache_dir" (next slide)

Optimizing disk space - Plugin Cache Directory

```
rafa@rp3:notes$ cat ~/.terraformrc
plugin_cache_dir   = "$HOME/.terraform.d/plugin-cache"
```

- Among the terraform CLI configuration options in file ~/.terraformrc
- If "plugin_cache_dir" is set to a specific directory it enables plugin caching
- From Documentation
  - By default, *terraform init* downloads plugins into a subdirectory of the working directory so that each working directory is self-contained. As a consequence, if you have multiple configurations that use the same provider then a separate copy of its plugin will be downloaded for each configuration.
  - Given that provider plugins can be quite large (on the order of hundreds of megabytes), this default behavior can be inconvenient for those with slow or metered Internet connections. Therefore Terraform optionally allows the use of a local directory as a shared plugin cache, which then allows each distinct plugin binary to be downloaded only once.

More on Terraform CLI

Terraform CLI – Command List

- Main documentation: https://www.terraform.io/docs/cli/index.html
- Main commands seen in relevant sections of the course

- **apply**
- **console**
- **destroy**
- env
- **fmt**
- force-unlock
- get
- graph
- **import**
- **init**
- login
- logout
- output
- **plan**
- providers
- providers lock
- providers mirror
- providers schema
- push Unsupported
- **refresh**
- **show**

- **state list**
- state mv
- state pull
- state push
- state replace-provider
- state rm
- state show
- **taint** Deprecated
- test Experimental
- **untaint** Deprecated
- **validate**
- **version**
- workspace list
- workspace select
- workspace new
- workspace delete
- workspace show

Terraform CLI Resources

- A Cloud Guru Terraform Cheatsheet

A CLOUD GURU

Terraform Cheatsheet

LEGEND

Headings are underlined

Commands are in BOLD

Helpful command descriptions are green

Plan, deploy and cleanup infrastructure

1. **terraform apply --auto-approve** Apply changes without being prompted to enter "yes"
2. **terraform destroy --auto-approve** Destroy/cleanup deployment without being prompted for "yes"
3. **terraform plan -out plan.out** Output the deployment plan to plan.out
4. **terraform apply plan.out** Use the plan.out plan file to deploy infrastructure
5. **terraform plan -destroy** Outputs a destroy plan
6. **terraform apply -target=aws_instance.my_ec2** Only apply/deploy changes to the targeted resource
7. **terraform apply -var my_region_variable=us-east-1** Pass a variable via command-line while applying a configuration
8. **terraform apply -lock=true** Lock the state file so it can't be modified by any other Terraform apply or modification action (possible only where backend allows locking)
9. **terraform apply refresh=false** Do not reconcile state file with real-world resources(helpful with large complex deployments for saving deployment time)
10. **terraform apply --parallelism=5** Number of simultaneous resource operations
11. **terraform refresh** Reconcile the state in Terraform state file with real-world resources
12. **terraform providers** Get information about providers used in current configuration

Terraform Workspaces

1. **terraform workspace new mynewworkspace** Create a new workspace
2. **terraform workspace select default** Change to the selected workspace
3. **terraform workspace list** List out all workspaces

Terraform state manipulation

1. **terraform state show aws_instance.my_ec2** Show details stored in Terraform state for the resource
2. **terraform state pull > terraform.tfstate** Download and output terraform state to a file
3. **terraform state mv aws_iam_role.my_ssm_role module.custom_module** Move a resource tracked via state to different module
4. **terraform state replace-provider hashicorp/aws registry.custom.com/aws** Replace existing provider with another
5. **terraform state list** List all the resources tracked in the current state file
6. **terraform state rm aws_instance.myinstace** Unmanage a resource, delete it from Terraform state file

Terraform Import and Outputs

1. **terraform import aws_instance.new_ec2_instance i-abcd1234** Import EC2 instance with id i-abcd1234 into the Terraform resource named "new_ec2_instance" of type "aws_instance"
2. **terraform import 'aws_instance.new_ec2_instance[0]' i-abcd1234** Same as above, imports a real-world resource into an instance of Terraform resource
3. **terraform output** List all outputs as stated in code
4. **terraform output instance_public_ip** List a specific declared output
5. **terraform output -json** List all outputs in JSON format

Terraform CLI tricks

1. **terraform -install-autocomplete** Setup tab auto-completion, requires logging back in

Format and validate Terraform code

1. **terraform fmt** Format code per HCL canonical standard
2. **terraform validate** Validate code for syntax
3. **terraform validate -backend=false** Validate code skip backend validation

Initialize your Terraform working directory

1. **terraform init** Initialize directory, pull down providers
2. **terraform init -get-plugins=false** Initialize directory, do not download plugins
3. **terraform init -verify-plugins=false** Initialize directory, do not verify plugins for Hashicorp signature

Terraform miscellaneous commands

1. **terraform version** Display Terraform binary version, also warns if version is old
2. **terraform get -update=true** Download and update modules in the "root" module

Terraform Console (Test out Terraform interpolations)

1. **echo 'join(",",["foo","bar"])' | terraform console** Echo an expression into terraform console and see its expected result as output
2. **echo '1 + 5' | terraform console** Terraform console also has an interactive CLI just enter "terraform console"
3. **echo "aws_instance.my_ec2.public_ip" | terraform console** Display the Public IP against the "my_ec2" Terraform resource as seen in the Terraform state file

Terraform Graph (dependency graphing)

1. **terraform graph | dot -Tpng > graph.png** Produce a PNG diagram showing relationship and dependencies between Terraform resources in your configuration/code

Terraform Taint/Untaint

1. **terraform taint aws_instance.my_ec2** Taint resource to be recreated on next apply
2. **terraform untaint aws_instance.my_ec2** Remove taint from a resource
3. **terraform force-unlock LOCK_ID** Force-unlock a locked state file, LOCK_ID provided when locking the State file beforehand

Terraform Cloud

1. **terraform login** Obtain and save API token for Terraform cloud
2. **terraform logout** Log out of Terraform Cloud, defaults to hostname app.terraform.io

Lab 01 – Getting Started

- Directory: lab_01_getting_started
- Configure AWS credentials in your PC
- Experiment with provider:
  - terraform init
  - Change provider version requirements
- With and without plugin cache
- Explore the terraform CLI
  - terraform fmt
  - terraform validate
  - terraform plan
  - terraform apply
- Estimated time 1 hour