Terraform

Getting Started

Release 2022-09-17-1332

Instructor

Sean P. Kane

@spkane

@superorbital_io

SUPERORBITAL

Follow Along Guide

Textual Slides

Katacoda Online Sandbox

https://learning.oreilly.com/scenarios/devops-tools-sandbox/9781098126469/

Prerequisites (1 of 2)

- A recent computer and OS
  - Recent/Stable Linux, macOS, or Windows 10+
  - Reliable and fast internet connectivity
- Hashicorp Terraform

Prerequisites (2 of 2)

- A graphical web browser
- A text editor
- A software package manager
- Git client
- General comfort with the command line will be helpful.
- [optional] tar, wget, curl, jq, SSH client

A Note for Powershell Users

Terminal commands reflect the Unix bash shell. PowerShell users will need to adjust the commands.

- Unix Variables
  - `export MY_VAR=test`
  - `echo ${MY_VAR}`
- Windows 10+ Variables (powershell)
  - `$env:my_var = "test"`
  - `Get-ChildItem Env:my_var`

Translation Key

**** - Unix Shell Line Continuation
` - Powershell Line Continuation (sort of)

${MY_VAR} - Is generally a place holder in the slides.

A Note About Proxies & VPNs

Proxies can interfere with some activities if they are not configured correctly and VPNs can increase audio and video streaming issues in class.

- Terraform
- Docker
- Docker-Compose

Instructor Environment

- **Operating System**: macOS (v12.6.X+)
- **Terminal**: iTerm2 (Build 3.X.X+) - https://www.iterm2.com/
- **Shell Prompt Theme**: Starship - https://starship.rs/
- **Shell Prompt Font**: Fira Code - https://github.com/tonsky/FiraCode
- **Text Editor**: Visual Studio Code (v1.X.X+) - https://code.visualstudio.com/

Terraform Definition

- **ter·ra·form**
- */ˈterəˌfôrm/*
  - (verb) to alter a planet for the purpose of sustaining life

Todo Definition

- As a helpful `reminder`, I keep a written list of things that I need `to do` later.
- **ri-mahyn-der**
- */rɪˈmaɪn dər/*
  - (noun) a person or thing that serves to remind.

Hashicorp Terraform

- Terraform is a tool that makes it possible to document and automate the creation, modification, and destruction of almost anything that can be managed by an API.
- This means that it is finally conceivable to automate the management of everything that your software stacks needs to actually run in any environment, including cloud resources, DNS entries, CDN configuration, and much more.

Installing Terraform

- Download:
  - https://www.terraform.io/downloads.html
- Unarchive and copy into your executable $PATH

Code Setup

```
$ cd ${HOME}
$ mkdir class
$ cd ${HOME}/class
$ git clone https://github.com/spkane/todo-for-terraform \
    --config core.autocrlf=input
$ cd todo-for-terraform
```

Exploring Terraform

```
$ cd terraform-infrastructure
```

- Note: Students **CAN NOT** run `terraform` in the `terraform-infrastructure` directory. This is for instructor demonstration.

The Setup

- `terraform init`

The Plan

- `terraform plan`

The Apply

- `terraform apply`
  - If all looks good, answer: `yes`

The Outputs

- `terraform output`

The State File

- `terraform state list`

HCL & JSON

- Hashicorp Configuration Language v2
  - https://github.com/hashicorp/hcl/tree/hcl2
- HCL is a JSON-compatible configuration language written by Hashicorp to be machine and human friendly.
- HCL is intended to provide a less-verbose JSON style configuration language that supports comments, while also providing humans with a language that is easier to approach than YAML.

Core Components

- Terraform & Backends
- Providers
- Variables
- Resources
- Data Sources
- Outputs
- State File

Examine the Server

- `. ./bin/ip_vars.sh`
- `ssh -i $HOME/.ssh/oreilly_aws -o IdentitiesOnly=yes ubuntu@${todo_ip}`
- `sudo systemctl status todo-list`
- `exit`
- `cd ..`

Test the Todo API

```
$ curl -i http://todo-api.spkane.org:8080/
$ curl -i http://todo-api.spkane.org:8080/ -X POST \
    -H 'Content-Type: application/spkane.todo-list.v1+json' \
    -d '{"description":"go shopping","completed":false}'
# In Windows Powershell try this for the POST command:
#   curl.exe -i -X POST `
#   -H 'Content-Type: application/spkane.todo-list.v1+json' `
#   --% `
#   -d "{\"description\":\"go shopping\",\"completed\":false}" `
#   http://127.0.0.1:8080/
```

More Todo API testing

```
$ curl -i http://todo-api.spkane.org:8080/
$ curl -i http://todo-api.spkane.org:8080/1 -X DELETE \
    -H 'Content-Type: application/spkane.todo-list.v1+json'
$ curl -i http://todo-api.spkane.org:8080/
```

The Todo Provider

- Open in your web browser:
  - https://registry.terraform.io/providers/spkane/todo

Copy the Code

- `cd $HOME/class/todo-for-terraform`
- `mkdir -p tf-code`
- `cp -a terraform-tests tf-code`
- `cd ./tf-code/terraform-tests`

Defining Variables

- Open `variables.tf`
  - This is where we define variables we will use in the terraform code.

Using the Todo Provider

- Open `main.tf`
  - Configure the todo provider
    - Change `host = "127.0.0.1"` to `host = "todo-api.spkane.org"`
  - Create 5 new todos
  - Read 1 existing todo as a data source
  - Create 5 more new todos based on the data source

Defining Outputs

- Open `outputs.tf`
  - Prints the IDs for all of the new todos

Prepare the Data

- We need a todo with ID 1 to read in as an example data source:

```
curl -i http://todo-api.spkane.org:8080/
curl -i http://todo-api.spkane.org:8080/ -X POST \
    -H 'Content-Type: application/spkane.todo-list.v1+json' \
    -d '{"description":"go shopping","completed":false}'
```

Apply Terraform Code

- `terraform init`
- `terraform apply`
  - **Plan**: 10 to add, 0 to change, 0 to destroy.
    - If all looks good, answer: `yes`

Examine the Outputs

- `terraform output`
- You may notice that your IDs are likely not in order. This is because, by default terraform creates many of the resources in parallel and we have many students using the server at the same time.

Examine todo.test1[0]

- Examine the state from one of the resulting todos
  - `terraform state list`
  - `terraform state show todo.test1[0]`

```
# todo.test1[0]:
resource "todo" "test1" {
    completed   = false
    description = "0-1 test todo"
    id          = "6"
}
```

The Complete State File

```
$ terraform state pull > \
    $HOME/class/state.json
$ less $HOME/class/state.json
$ rm $HOME/class/state.json
```

The Real Object

- From the output of the last command, grab the ID and use it at the end of this command.
- `curl -i http://todo-api.spkane.org:8080/6`

```
HTTP/1.1 200 OK
Date: Wed, 01 Jan 2020 20:13:45 GMT
Content-Type: application/spkane.todo-list.v1+json
Content-Length: 59
Connection: keep-alive

[{"completed":false,"description":"0-1 test todo","id":6}]
```

Updating Objects

- Change the 2 `count = 5` lines to read `count = 4`
- Add `(updated)` to the end of the first description string.

Code With Edits

```
resource "todo" "test1" {
  count = 4
  description = "${count.index}-1 test todo (updated)"
  completed = false
}

resource "todo" "test2" {
  count = 4
  description = "${count.index}-2 test todo (linked to ${data.todo.foreign.
  completed = false
}
```

Examine The Current State

```
$ terraform state show todo.test1[0]
$ terraform state show todo.test1[4]
```

Apply The Updates

- `terraform apply`
  - **Plan**: 0 to add, 4 to change, 2 to destroy.
    - If all looks good, answer: `yes`

Re-examine The State

- `terraform state show todo.test1[0]`
  - The description should now be updated.
- `terraform state show todo.test1[4]`
  - This should give you an error since it has now been deleted.
- `terraform state show todo.test1[3]` will work however, since we only have 4 todos now.

Prepare to Import

- Create a new todo by hand:

```
$ curl -i http://todo-api.spkane.org:8080/ -X POST \
    -H 'Content-Type: application/spkane.todo-list.v1+json' \
    -d '{"description":"Imported Todo","completed":false}'
```

- Note the ID in your output (*13* in this example):

```
{"completed":false,"description":"Imported Todo","id":13}
```

Modify The Code

- In `main.tf` add:

```
resource "todo" "imported" {
  description = "Imported Todo"
  completed = false
}
```

Run a Plan

- `terraform plan`
  - You should see: **Plan**: 1 to add, 0 to change, 0 to destroy.

```
  # todo.imported will be created
  + resource "todo" "imported" {
      + completed   = false
      + description = "Imported Todo"
      + id          = (known after apply)
    }
```

Import a Pre-Existing Todo

- Import the ID of the Todo that you just created.
  - `terraform import todo.imported[0] 13`

Re-run the Plan

- `terraform plan`
  - You should see
    - **No changes. Infrastructure is up-to-date.**

Rename a Resource

- In `main.tf`:
  - change the line `resource "todo" "imported" {` to read `resource "todo" "primary" {`
- Run `terraform plan`
  - You should see
    - **Plan**: 1 to add, 0 to change, 1 to destroy.
- This would delete one todo and create a new one.
  - This is not what we want.

Manipulating State

- `terraform state mv todo.imported todo.primary`
- Run `terraform plan`
  - You should see
    - **No changes. Infrastructure is up-to-date.**
- By moving the state of the existing resource to the new name, everything lines back up properly.

Terraform Modules

- Blocks of re-useable Terraform code w/ inputs and outputs
- `cd ..`
- `cp -a ../__modules .`
- `cd __modules/todo-test-data`

Module Variables

- Open `variables.tf`
  - This files defines all the variables that the module uses and any default values.

The Main Module Code

- Open `main.tf`
  - This files will allow us to easily create two set of todos matching our specific requirements.

Module Outputs

- Open `outputs.tf`
  - If you think of modules like functions then outputs are return values.

Prepare to Use the Module

- `cd ../../terraform-tests/`
- Open `main.tf`

Utilize the Module (1 of 2)

- Add:

```
module "series-data" {
    source       = "../__modules/todo-test-data"
    number       = 5
    purpose      = "testing"
    team_name    = "oreilly"
    descriptions = ["my first completed todo", "my second completed todo",
                    "my third completed todo", "my fourth completed todo",
                    "my fifth completed todo"
                   ]
}
```

Utilize the Module (2 of 2)

- Open `outputs.tf`
- Add:

```
output "first_series_ids" {
  value = module.series-data.first_series_ids
}

output "second_series_ids" {
  value = module.series-data.second_series_ids
}
```

Apply the Module

- `terraform init`
  - Initialize/download the module.
- `terraform apply`
  - If all looks good, answer: `yes`

Destroy the Todos

- `terraform destroy`
  - **Plan**: 0 to add, 0 to change, 9 to destroy.
    - If all looks good, answer: `yes`

Real World Infrastructure

- `cd terraform-infrastructure`

Terraform & Backends

- Open `main.tf`
  - Terraform block
    - Define high-level requirements for this associated HCL. Terraform and provider version, etc.
  - Backend block
    - Define where remote state is stored and any information required to read and write it.

Providers (1 of 2)

- Providers
  - Individual plugins that enable terraform to properly interact with an API.
  - These can range between Hashicorp's officially supported providers to custom providers written by a single developer.

Providers (2 of 2)

- In this example we are using the `aws` and `ns1` providers.
  - https://github.com/terraform-providers/terraform-provider-aws
  - https://github.com/ns1-terraform/terraform-provider-ns1

Variables

- Open `variables.tf`
  - Defines all the variables that you will be using and their default values.
- You will get errors if you use variables that are not defined in this file.

Data Sources

- Open `data.tf`
- Using output as input
  - Remote Terraform State
  - APIs
  - Scripts
    - Open `bin/local-ip.sh`
  - etc

Building Infrastructure

- `key-pairs.tf`
- `backend.tf`
- `frontend.tf`
- `security-groups.tf`

Backend Service

- Open `key-pairs.tf`
  - SSH public key for system access
- Open `backend.tf`
  - Server Instance w/ basic provisioning
  - Setup of `todo` backend service
- The files in `./files` support the system provisioning.

Frontend Infrastructure

- Open `frontend.tf`
  - S3 bucket (file share) for Load Balancer Logs
    - Security Policy for access to S3 bucket
  - Load Balancer for backend `todo` service
    - Listener
    - Target Group
    - Target Group Attachment
  - DNS record for load balancer

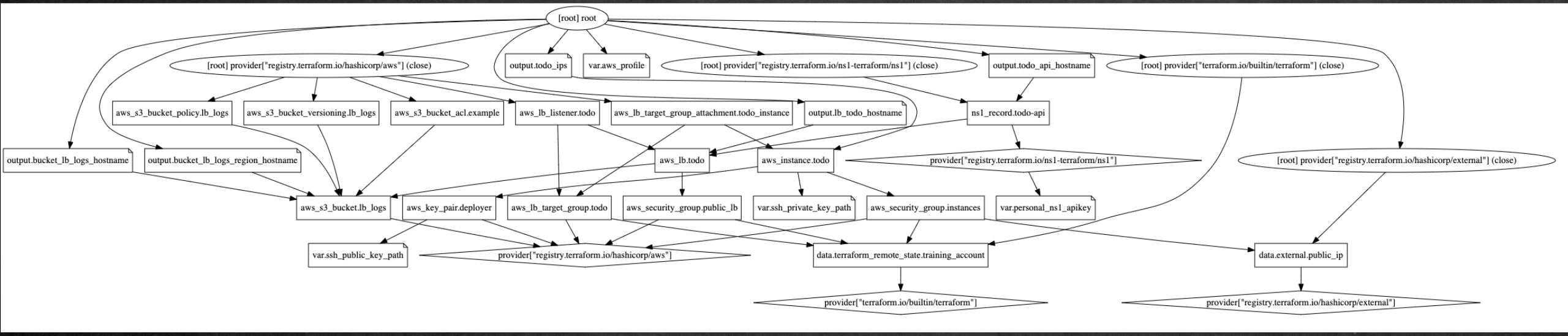
Firewall Security

- Open `security-groups.tf`
  - SSH to the backend server
  - Traffic between load balancer and `todo` service

Outputs

- Open `outputs.tf`
  - Human and computer-readable data

The Graph



Destroy the Infrastructure

- `terraform destroy`
  - If all looks good, answer: `yes`

What We Have Learned

- How to install Terraform
- The primary use case for Terraform
- How to install a provider and what they are for
- Creating, reading, updating, and deleting objects
- Reading data sources & importing existing objects
- Making & using modules
- What the Terraform state is
- and more...

Additional Reading

Terraform: Up & Running

Terraform Documentation

Additional Learning Resources

https://learning.oreilly.com/

Student Survey

Please take a moment to fill out the class survey linked to from the bottom of the ON24 audience screen.

O'Reilly and I value your comments about the class.

Thank you!

Any Questions?

[Sean P. Kane]

Providing stellar Kubernetes engineering and workshops.

https://superorbital.io/contact/