Terraforming with Azure for absolute beginners

4/2/2022

Relevancy to System Administration

This weekend I attended a meetup called "Terraforming with Azure for Absolute Beginners" by a group called Mirav Academy. They provide a lot of free IT training on meetup.com on DevOps, QA, and Cloud platforms. In class, we learned how system administrators' job often includes maintaining, developing, and designing cloud infrastructure. Some ways they do this is through AWS CLI like we do in class. With Cloud platform CLIs, we are kind of coding the infrastructure using scripting or commands, so a term for this type of tool was developed: Infrastructure as Code.

Infrastructure as Code(IOC) is code that deploys infrastructure resources onto different platforms as opposed to a user interface. This makes provisioning consistent and predictable. For instance, deploying 100 servers through the UI can result in human errors and loss of time, so we can use IOC to deploy the 100 servers instead, reducing the risk of errors and saving time.

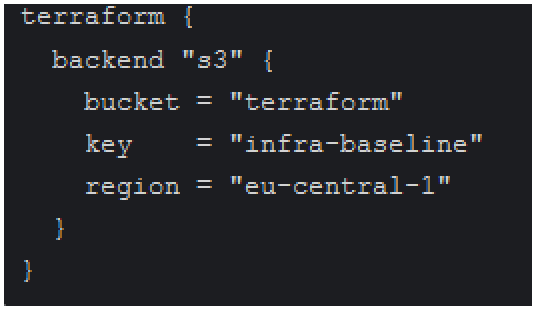
IOC tools can be subdivided into three categories. Configuration management tools which include technologies such as Ansible and Puppet which help design and install software, maintain version control, and idempotency. The next category is Server Templating Tools. This category includes Docker and Vagrant which include pre-installed software/dependencies/, virtual machines and Docker images, and immutable infrastructure. The third category is provisioning tools such as Terraform and Cloud Formation. These are used to deploy servers, databases, or docker images.

Summary of meetup

This meetup introduced Terraforming. Terraform is another type of IOC tool that uses human readable code to deploy servers so users don't have to go through the overhead of learning a specific cloud platform's CLI. It lets you build, change, and version infrastructure safely and efficiently. This includes low level components such as compute instances, storage, and networking and high level components like DNS servers. The benefit of Terraforming over a platform's CLI is that it allows for greater control and collaboration over deploying infrastructure. Other benefits include:

- Everything is done through code
- Enables DevOps
    - It can track version control for better collaboration and visibility
- Declarative infrastructure
    - Parameters are passed from the user to explain exactly what they want in their infrastructure
- Speed, cost, and reduced risk
    - There is less human intervention during deployment so there are less security flaws, errors, and more time is saved
- Supported by multiple cloud providers
    - GCP, AWS, Azure, and more
- Idempotency
- Consistent Infrastructure

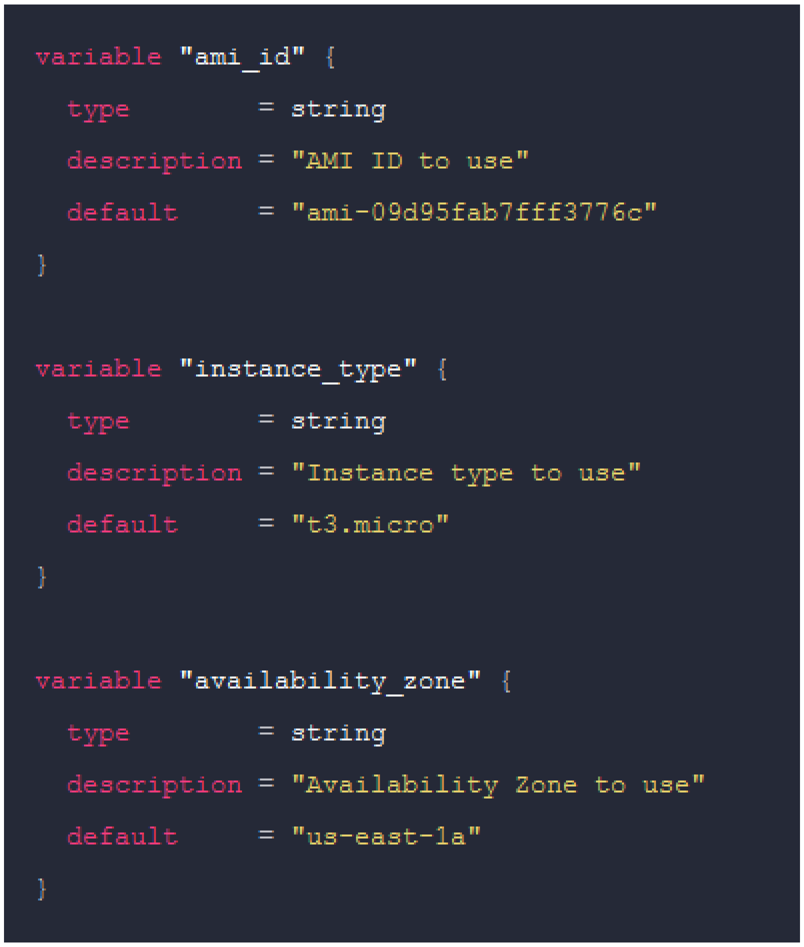
Terraforming uses a domain specific language called HashiCorp Configuration Language(HCL) and has an extension of .tf. Here is an example of HCL creating an s3 bucket in eu-central-1:



```
terraform {
  backend "s3" {
    bucket = "terraform"
    key    = "infra-baseline"
    region = "eu-central-1"
  }
}
```

Similar to the UI and CLI, you can specify regions, tags, and other attributes.

You can also introduce variables like the following and use them throughout your infrastructure code.

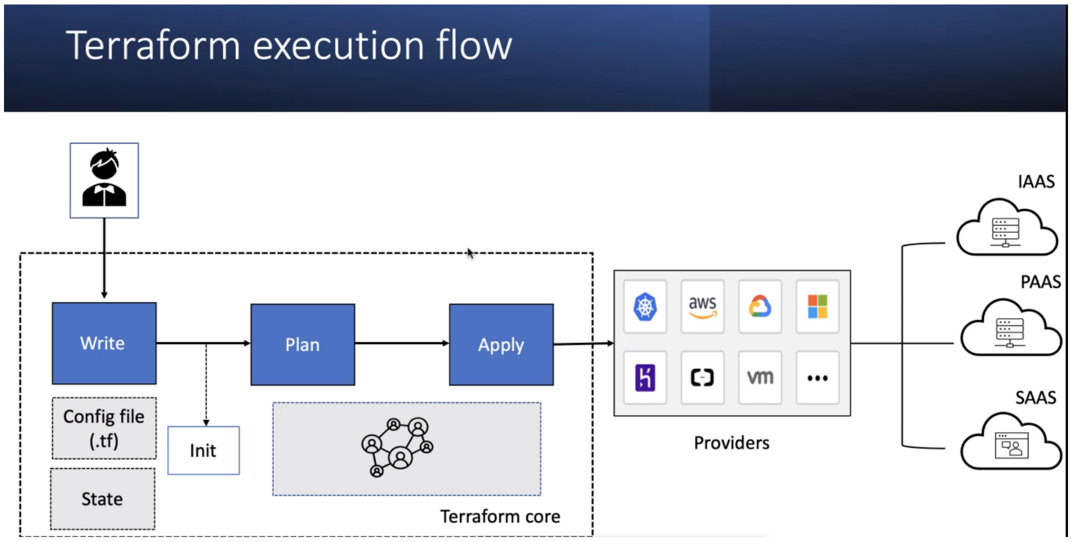


```
variable "ami_id" {
  type        = string
  description = "AMI ID to use"
  default     = "ami-09d95fab7fff3776c"
}

variable "instance_type" {
  type        = string
  description = "Instance type to use"
  default     = "t3.micro"
}

variable "availability_zone" {
  type        = string
  description = "Availability Zone to use"
  default     = "us-east-1a"
}
```

This is an example of the workflow to execute Terraform code they introduced:



1. Write
    a. User writes config files to specify the type of resources that they want deployed
2. Init
    a. Syncs project with cloud provider
3. Plans
    a. Test run configuration files/code, nothing is deployed
    b. Allows users to see what is going to be deployed before committing changes
4. Apply
    a. Deploys infrastructure to cloud with specified provider
    b. State files then contain the difference between your local project and what is actually deployed

Takeaways

Terraforming is an absolutely brilliant technology. Imagine all the fine tuning you have over your data in normal code through data structures, expressions, and organization. Now you can have a similar control over your infrastructure. To onboard someone onto your project, they can clone the config files from your repository and start editing and making changes. It allows people to collaborate on infrastructure similar to how developers would collaborate on code. Developers can also version their infrastructure meaning they can roll back to a certain version of their infrastructure or merge with another project's configuration. I've attached the official website for Terraforming and the group's youtube channel with recordings of their previous meetups and sessions.

Sources

- Terraform Website
    - https://terraform.io
    - Azure documentation
        - https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs
    - AWS documentation
        - https://registry.terraform.io/providers/hashicorp/aws/latest/docs
- Youtube Channel with previous meetups
    - https://www.youtube.com/channel/UCbHwYg-usZfXhfvod0WYTyA/videos