

IaC with the focus on Terraform

1 Need of IaC in IT industry

Think about coding the infrastructure just like any program, not heard about it? Don't worry this essay will explain how this concept of Infrastructure as a Code (IaC) is getting popular across IT. This essay consists of two major sections, starting from the introduction and comparison of hot tools that facilitates IaC, that is part one, we will go deep into the Terraform implementation in part two, using Azure cloud just as an example.

1.1 Traditional Approach to Infrastructure Management

Earlier there were servers which had to be set up manually by a separate team on-premise that took months of time. The task involved stacking servers on the server racks, configuration of them on the requirements specific to the operating system used and application being hosted.



All this process is time consuming and painful. Firstly, it involves specialized engineers for storage, network engineers, system administrators etc. Secondly, the cost of setup grows subsequently. Ensuring high availability takes x2 effort. Maintaining backup physical server racks and their maintenance is too much of effort. Thirdly, scaling up in case of increased load you may need extra servers, those may not be used for long when the traffic decreases. In this

case the servers will stay unused taking up considerable air-conditioning and maintenance costs.

Emergence of cloud computing solved most of these problems. But here also you need to spin up cloud instances and other dependent components instantly to address the concerns of agility, load balancing and high availability. Therefore, the need comes for something that could be instantly brought up in a couple of minutes without involving any human error ensuring a secure environment this problem was resolved by IaC.

1.2 What is Infrastructure as Code?

As the name suggests; it is the concept of managing the operations environment in the same way applications and code are managed for release. It's a quest to move away from the traditional approach of managing infrastructure and be able to manage the operations infrastructure by following the same modern rules and procedures that govern code development. That means that the core best practices of DevOps—like version control, virtualized tests, and continuous monitoring—are applied to the underlying code that governs the creation and management of the infrastructure.

1.3 Benefits of IaC

1.3.1 Change Tracking & low risk

As we are coding our infrastructure in a source control system. Hence, we get the benefit of change tracking. It's a big advantage when you are dealing with a large team and working in shifts on multiple servers. Then the consistency is to be maintained. With source control in case of any outage in producing there is nothing more beneficial than tracking the changes in infrastructure in GIT.

1.3.2 Increased Site Reliability

In most of the companies they use a legacy system of running monthly scans on the infrastructure for vulnerabilities by the security or a site reliability team. These health checks can't give hundred percent of accuracy and also it might interfere with human errors while execution. These issues are tackled very well by these configuration management tools like Ansible, Chef etc.

1.3.3 Experimenting while having expenses in control

Cloud technologies have many benefits financially that adopt the policy of pay as you use. Moreover, in combination with IaC enables developers to experiment the infrastructure by

scaling up and down the same replica of the production servers for regression testing without spending time in repetitive manual tasks of infrastructure handling. This can be done by a limited number of DevOps or system administrators who are responsible to handle the infrastructure as it all lies inside the script. Once the task is done all the virtual instances can be destroyed within a couple of minutes. Hence, saving the company's cost.

1.3.4 Integration with other automated flows

Many have heard about those famous CI/CD pipelines; these pipelines can be used together with the IaC tools. For instance, think of a scenario when an event infrastructure is created followed by a CI/CD pipeline that is application specific code build and deploy. This is where IT industries are investing in.

2 Classification of IaC tools

IaC tools can be categorized according to the following properties:

2.1 Type

IaC tools can be classified into two groups according to type: Configuration management and Provisioning. Configuration management tools are designed to install and manage software on existing servers. This can be the installation of packages, starting of services, placing scripts or config files on the instance, etc. Provisioning tools are designed to provision the servers themselves. By provisioning, it means the first time the infrastructure is being built. However, these categories are not mutually exclusive. Some configuration tools can for example do to some degree provisioning functions and vice versa. The focus on configuration management or provisioning refers to the original purpose the tool was built for or what kind of task the tool can do better.

2.2 Infrastructure paradigm

Infrastructure can be classified as mutable or immutable. Mutable means further updates are installed on the existing servers and the changes will happen in-place. In immutable infrastructure every change is the deployment of a new server. A new version is created, and the old versions are destroyed. Mutable paradigm faces a problem known as configuration drift. As those modifications and changes happen, the configuration of the applications and infrastructure changes. Over time the configuration installed and that currently running become different making bug detection difficult.

2.3 Language

Language of IaC tools can be classified into two groups: Imperative and Declarative.

Declarative allows the user to specify what the configuration should be and let the system figure out the steps to take. Imperative approach requires the user to define the steps to take to create and configure resources.

2.4 State storage and distribution of updates

We can group IaC tools as Client/Server or Client only according to how they manage state and distribute updates. Some IaC tools require that you run a *master server* for storing the state of your infrastructure and distributing updates. Commands are issued to the master server, and the master server either pushes the updates out to all the other servers, or those servers pull the latest updates down from the master server on a regular basis. Client only IaC tools do not require any additional Infrastructure.

2.5 Agent software requirements

Some IaC tools require the installation of agent software on each server to be configured. The agent typically runs in the background on each server and is responsible for installing the latest configuration management updates. Some tools do not require the installation of any extra agents. They are typically already installed as part of the infrastructure that is being used.

2.6 Cloud platform

Some IaC are platform specific, especially the proprietary ones. However, many IaC tools are open source and can be used for any cloud platform. The advantage with platform agnostic tools is that you can manage a heterogeneous environment with the same workflow in a situation where you have different cloud and platforms to support various platforms.

3 Terraform as IaC tool

Terraform is one of the most popular IaC tools out there. Terraform is an all platform, open source, and mainly a provisioning tool but can also be used to manage the existing infrastructure. It follows a declarative approach to IaC and this means configuration files only describe the desired state and Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. Terraform is Client only by default. However, terraform can be configured to communicate with cloud providers using the cloud provider's API and in this sense the API servers can be thought of as

the master servers, except they don't require any extra infrastructure or any extra authentication mechanisms. One interesting thing about Terraform is that its plan command allows you to see what changes you are about to apply before you apply them. Let us look at the scenario of infrastructure having a VM, Kubernetes cluster and they are networked together using Virtual Private Cloud (VPC). To implement such an infrastructure the following steps are followed:

3.1 Code

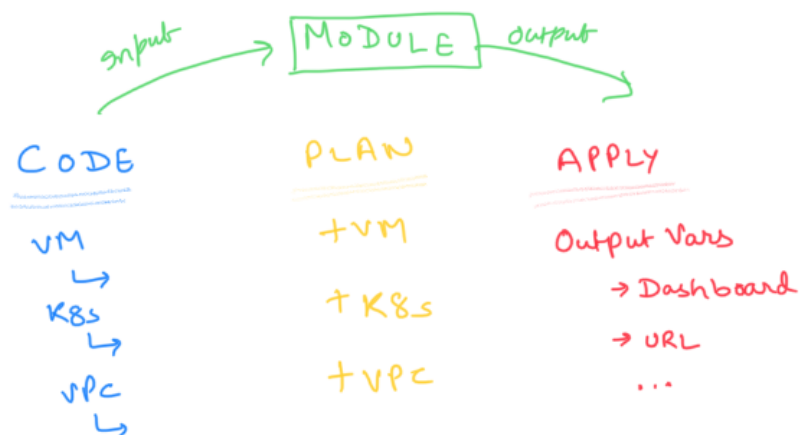
To start with this we need to have a terraform file, we must write about the VM - mentioning the name networking, data centers etc., a Kubernetes cluster- related arguments and a VPC to connect the and have a network.

3.2 Plan

Run terraform plan from desired state to what needs to be created. Once all has been checked.

3.3 Apply

It is another terraform CLI command (terraform apply), terraform calls the cloud provider real API and local API token to output auto-generated variables on the way like generating Kubernetes dashboard URL or URL to access our application.



As seen from this example, we don't have to maintain any agent. Hence, reducing the overhead of its maintenance and security. We will delve into these steps later when we implement some infrastructure in Azure cloud using Terraform.

4 Other popular IaC tools

4.1 Amazon CloudFormation

AWS CloudFormation is for users working Amazon Web Services (AWS) Cloud. It provides users with a simple way to create and manage a collection of Amazon Web Services (AWS) resources by provisioning and updating them in a predictable way. The infrastructure is modelled within JSON or YAML. Once the configuration is defined in the template, CloudFormation handles the rest of the tasks e.g building and rebuilding infrastructure, security and scaling.

4.2 Azure Resource Manager

Azure Resource Manager (ARM) is the deployment and management service for Azure. It provides a management layer that enables users to create, update, and delete resources in Azure accounts. The configuration is stored in a JSON file called Resource Manager template. In this JSON file users define the resource group, its resources, their properties and any dependencies. This allows an identical copy of the application to easily be created so it can be deployed in testing, staging, production or in an additional geography to allow the service to scale out. ARM is usually managed through the centralized GUI portal, but for customers with advanced needs, it also supports Azure PowerShell, Azure CLI, Azure REST APIs, and client SDKs.

4.3 Google Cloud Deployment Manager (Google CDM)

This is the automation tool from Google. It allows users to specify all the resources needed for the application in a declarative format using templates written in yaml or Python or Jinja2. In the templates, users parameterize the configuration and this template can be reused for common deployment paradigms such as a load balanced, auto-scaled instance group. Google CDM also supports previews; which means rather than committing changes directly, you can sneak an advanced overview of the impact deployments and changes will have. The feature allows for human errors to be avoided and to strengthen and stabilize your infrastructure as a whole.

4.4 Puppet

The three IaC tools described so far are tailored for specific cloud environments. Puppet is for any cloud environment. Puppet uses a declarative, model-based approach to IT

automation. Users model the infrastructure and Puppet translates the model into operating system and software setup instructions. It uses Ruby-based DSL as the primary language for defining the desired end state of the infrastructure. Puppet does frequent polling to check that the configuration is in the desired state (every 30 mins by default). Unlike other tools that do the job when triggered, Puppet is always on to prevent configuration drift.

4.5 Ansible

Like Puppet, Ansible can use a declarative model-based approach and it is for any cloud environment. Ansible also supports an imperative approach requiring the user to define the steps to take to create and configure resources. Ansible is a general-purpose tool. Apart from building infrastructure it can be used for deploying and configuring applications. It has a very simple configuration language, yaml in the form of Ansible Playbooks, that almost approaches plain English. Ansible is a popular choice for automating provisioning of Docker containers and Kubernetes deployments.

4.6 Chef

All platform configuration tool and uses both imperative and declarative approaches. The machine setup is described in a Chef recipe. Collections of recipes are stored in a cookbook that relates to a single task, but can have a number of different server configurations involved e.g web application with a database, will have two recipes, one for each part, stored together in a cookbook. The recipes are written in Ruby but the domain specific language used by Chef is designed to be able to be understood by everyone.

5 Comparison of most popular IaC tools available

The table below summarizes important differences between IaC tools:

Tool	Platform	Type	Infrastructure	Agent	Language	Architecture
Amazon Cloud Formation	Amazon Cloud	Config and Provisioning	Immutable	No	Declarative	Client only
Azure Resource Manager	Azure	Config and Provisioning	Immutable	No	Declarative	Client only
Google CDM	Google Cloud	Config and Provisioning	Immutable	No	Declarative	Client only
Terraform	Any	Provisioning	Immutable	No	Declarative	Client only
Puppet	Any	Config	Mutable	Yes	Declarative	Client/Server
Ansible	Any	Config	Mutable	No	Declarative and Imperative	Client only
Chef	Any	Config	Mutable	Yes	Declarative and Imperative	Client/Server

As seen from the table above Terraform is the only open-source tool that is platform agnostic, declarative, builds immutable infrastructure, agentless and client only. The importance of these characteristics is more highlighted below:

5.1 Open Source

Terraform is backed by large communities of contributors who build plugins to the platform. Regardless of which cloud provider you use, it's easy to find plugins, extensions, and professional support. This also means Terraform evolves quickly, with new benefits and improvements added consistently.

5.2 Platform agnostic

You can use Terraform it with *any* cloud services provider. As seen, the other IaC tools are designed to work with a single cloud provider.

5.3 Immutable infrastructure

Each change to the environment, the current configuration is replaced with a new one that accounts for the change, and the infrastructure is reprovisioned. Even better, previous configurations can be retained as versions to enable rollbacks if necessary or desired. This clearly integrates well with DevOps practices.

5.4 Agentless

There is no overhead of installing and maintaining the agent software.

5.5 Client only

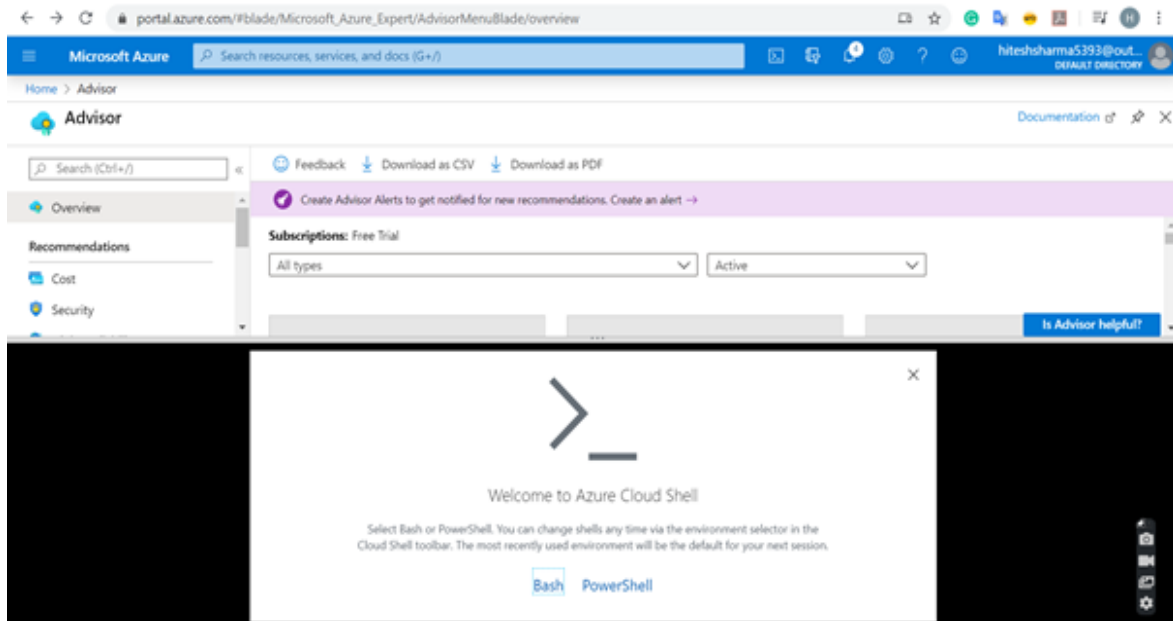
When using Terraform there is no need of extra infrastructure to maintain the master configuration server.

6 Terraform in Action

To show the power of Terraform in one of the leading cloud environments seems necessary to wrap up this essay. The screenshot shows the implementation of cloud infrastructure that involves setup of a VM with necessary public IP binding, configuring network security group firewall and network interface.

Following screenshots will guide you through the implementation. It tells how much effort it takes for implementing Terraform in Azure cloud:

1. Following figure shows the Azure portal home page and when we open its cloud shell.

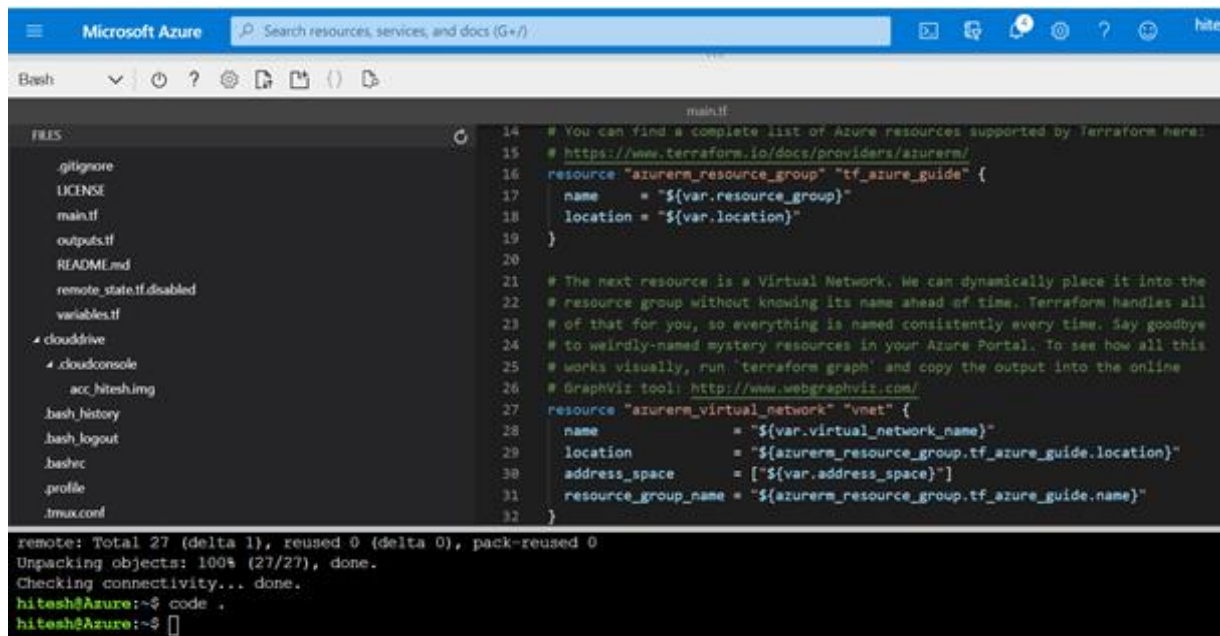


2. Once the account is initialized and membership is authenticated. Terraform scripts written in HashiCorp Configuration Language (HCL), are cloned from the repository.

```
Bash
hitesh@Azure:~$ terraform --version
Terraform v0.12.23

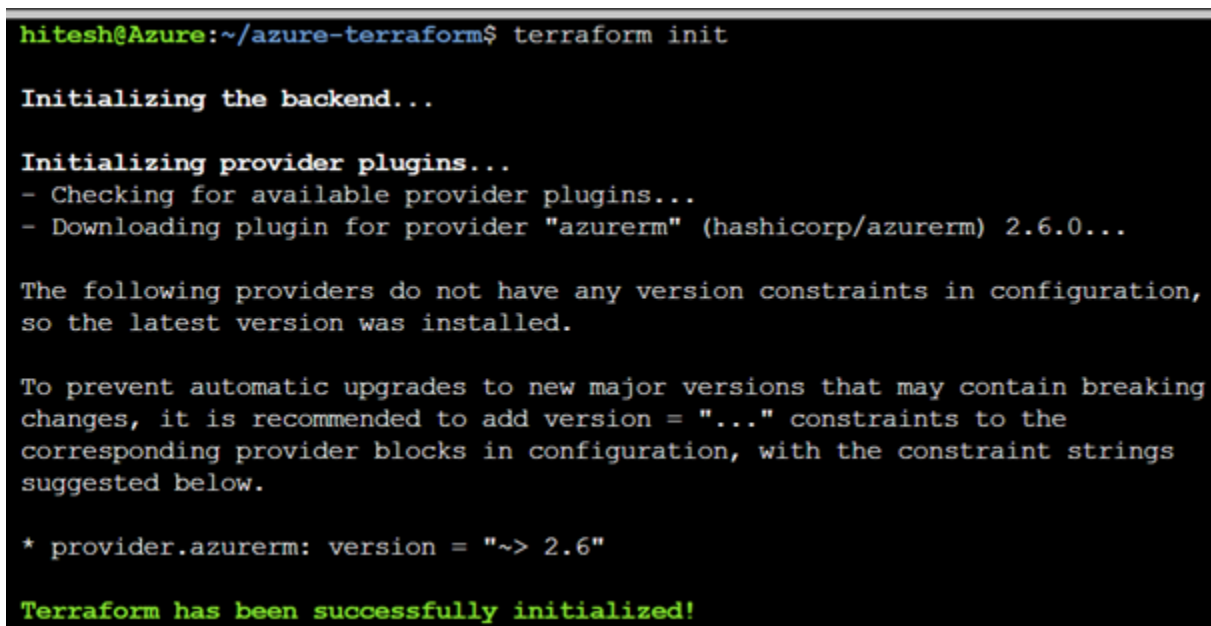
Your version of Terraform is out of date! The latest version
is 0.12.24. You can update by downloading from https://www.terraform.io/downloads.html
hitesh@Azure:~$ git clone https://github.com/hittesharma/azure-terraform.git
Cloning into 'azure-terraform'...
remote: Enumerating objects: 27, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 27 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (27/27), done.
Checking connectivity... done.
hitesh@Azure:~$ code .
```

3. The resource group can be clearly seen in the picture, they are necessary in Azure for implementing cloud infrastructure. They are logical collections of virtual machines, storage accounts, virtual networks, web apps, databases, and database servers. Similarly, other commands for virtual machines, security groups etc. were scripted in HCL to provision the environment.



```
Microsoft Azure Search resources, services, and docs (G+/) hite
Bash
FILES
.gitignore
LICENSE
main.tf
outputs.tf
README.md
remote_state.tf.disabled
variables.tf
clouddrive
cloudconsole
acc_hiteshimg
.bash_history
.bash_logout
.bashrc
.profile
.tmux.conf
main.tf
14 # You can find a complete list of Azure resources supported by Terraform here:
15 # https://www.terraform.io/docs/providers/azurerm/
16 resource "azurerm_resource_group" "tf_azure_guide" {
17   name     = "${var.resource_group}"
18   location = "${var.location}"
19 }
20
21 # The next resource is a Virtual Network. We can dynamically place it into the
22 # resource group without knowing its name ahead of time. Terraform handles all
23 # of that for you, so everything is named consistently every time. Say goodbye
24 # to weirdly-named mystery resources in your Azure Portal. To see how all this
25 # works visually, run 'terraform graph' and copy the output into the online
26 # GraphViz tool: http://www.webgraphviz.com/
27 resource "azurerm_virtual_network" "vnet" {
28   name                = "${var.virtual_network_name}"
29   location             = "${azurerm_resource_group.tf_azure_guide.location}"
30   address_space       = ["${var.address_space}"]
31   resource_group_name = "${azurerm_resource_group.tf_azure_guide.name}"
32 }
remote: Total 27 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (27/27), done.
Checking connectivity... done.
hitesh@Azure:~$ code .
hitesh@Azure:~$
```

4. terraform init command performs several different initialization steps in order to prepare a working directory for use. Terraform will attempt to initialize that configuration. Terraform init understands what the terraform is and what it contains. Like it has recognized it is for Azure. Also, downloaded the plugin to connect with Azure using APIs.



```
hitesh@Azure:~/azure-terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "azurerm" (hashicorp/azurerm) 2.6.0...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

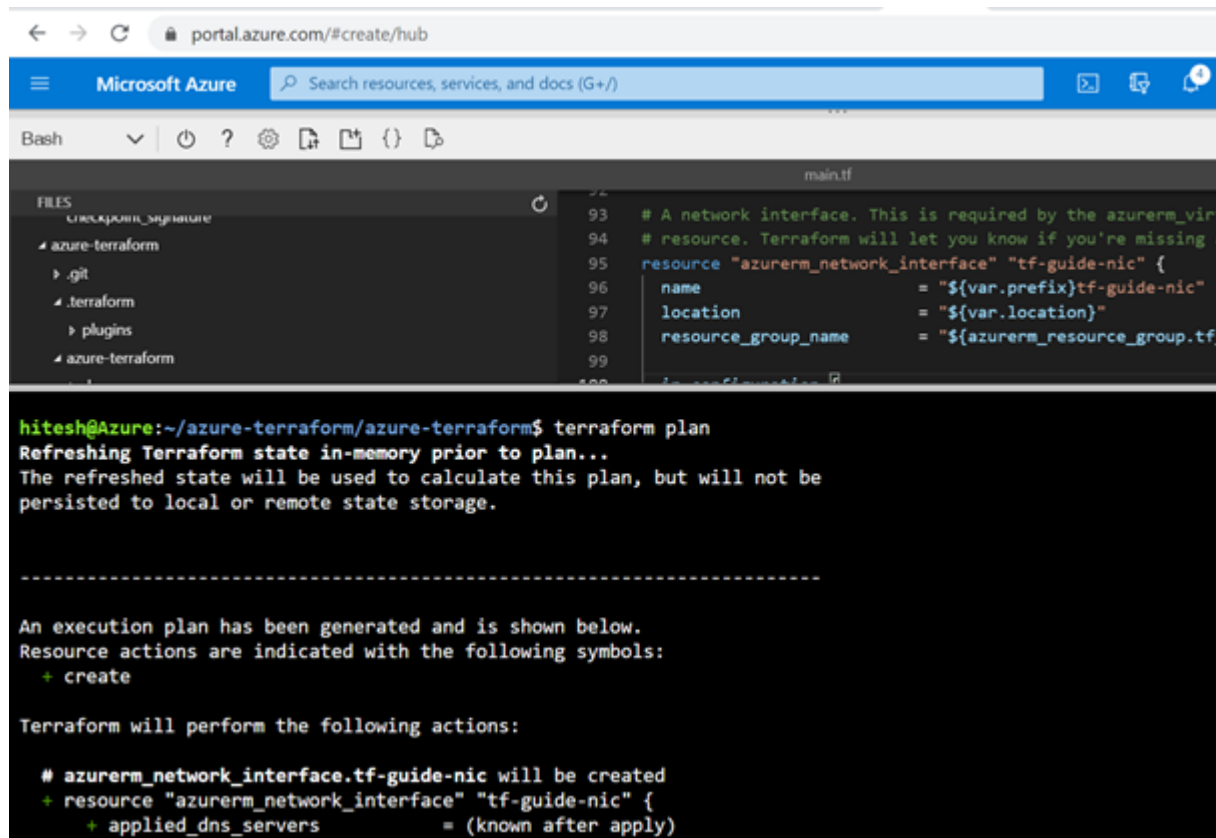
To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.azurerm: version = "~> 2.6"

Terraform has been successfully initialized!
```

5. terraform plan command is used to create an execution plan. It performs a refresh and determines what actions are necessary to achieve the desired state specified in the configuration files. It is used for validation of whether the execution plan for a

set of changes matches our expectations without making any changes to real resources or to the state. For example, terraform plan could run before committing a change to your repository, to ensure that it will behave as per our expectations. In other words, I will say it does a dry run.



```
portal.azure.com/#create/hub

Microsoft Azure Search resources, services, and docs (G+/)

Bash

FILES
  checkpoint_signature
  azure-terraform
    .git
    .terraform
      plugins
  azure-terraform

main.tf
93 # A network interface. This is required by the azurerm_virtual_machine resource. Terraform will let you know if you're missing a dependency for this resource.
94 # resource. Terraform will let you know if you're missing a dependency for this resource.
95 resource "azurerm_network_interface" "tf-guide-nic" {
96     name                       = "${var.prefix}tf-guide-nic"
97     location                   = "${var.location}"
98     resource_group_name        = "${azurerm_resource_group.tf-guide-rg.name}"
99     ip_configuration {
100         name               = "ipconfig1"
101         subnet_id           = azurerm_subnet.tf-guide-subnet.id
102         private_ip_address  = "${var.private_ip_address}"
103         public_ip_address_id = azurerm_public_ip.tf-guide-pip.id
104     }
105 }

hitesh@Azure:~/azure-terraform/azure-terraform$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

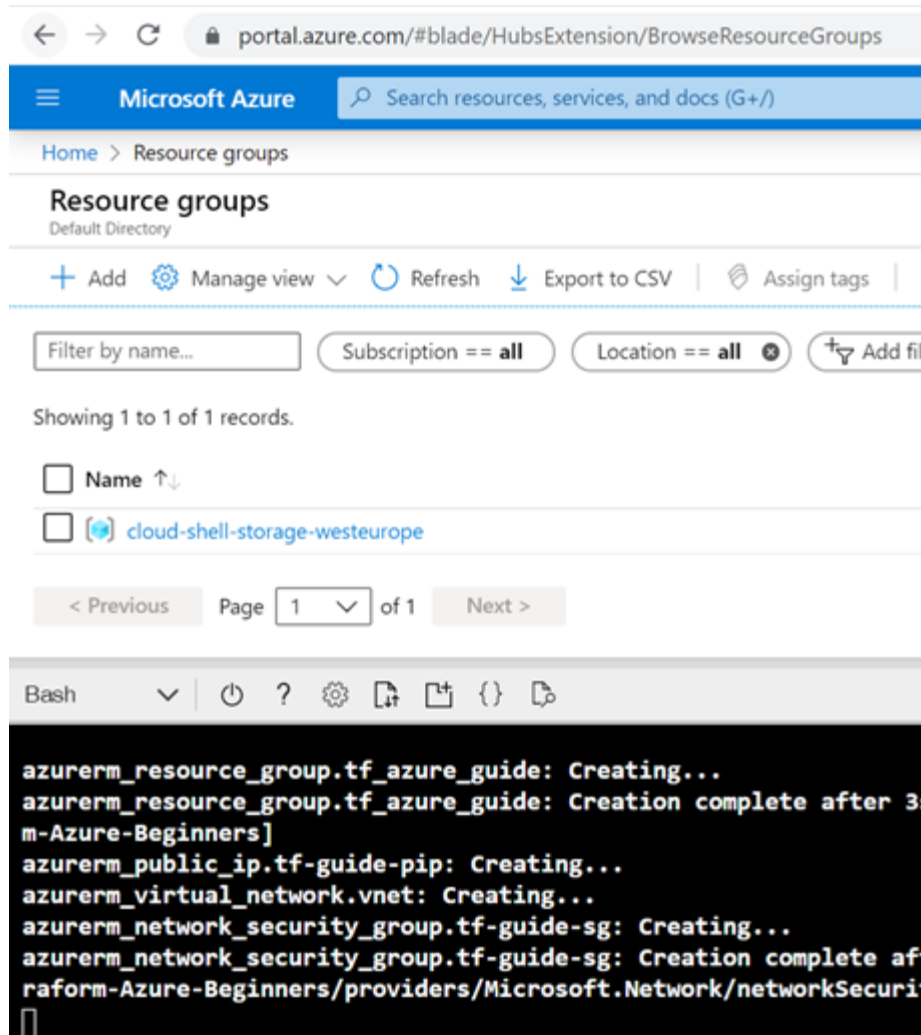
-----

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# azurerm_network_interface.tf-guide-nic will be created
+ resource "azurerm_network_interface" "tf-guide-nic" {
+   applied_dns_servers      = (known after apply)
+   ip_configuration {
+     name               = "ipconfig1"
+     subnet_id           = azurerm_subnet.tf-guide-subnet.id
+     private_ip_address  = "${var.private_ip_address}"
+     public_ip_address_id = azurerm_public_ip.tf-guide-pip.id
+   }
}
```

6. terraform apply scans the current directory of any configuration changes if that had happened. If the entire configuration is new, then it will execute the complete plan. As we are aware of reusability of functions/ methods in programming similarly we can reuse the general terraform templates and call them within another one. This allows to split a big plan into more logical reusable templates within a system. Resource group name and other resources created can be seen in the figure. Terraform is intelligent enough and can deploy various resources in parallel resolving dependencies saving time.



7. In the following figure we can see from the resource group that our infrastructure is ready in a lot less time. Just these seven steps are required to be followed if you have your terraform script ready which is one time work only. As simple as that!

7 Conclusion

Infrastructure as Code is the modern way of managing infrastructure.

Many IT companies can implement stable and reliable environments quickly and at scale, avoiding manual intervention and enforcing consistency by representation of the desired state of infrastructure by scripting it. This in turn helps to develop repeatable infrastructure avoiding any kind of run time errors which are usually caused by missing dependencies or variation in configuration called as configuration drift.

Majority of tech companies across the globe require an opensource, cloud-agnostic provisioning tool that supports immutable infrastructure. Another factor that is good to have is

considering a client-only architecture that can be run easily using declarative approach. Terraform is the only such cloud-agnostic tool that meets all of these!

8 References

- <https://www.terraform.io/intro/index.html> last accessed 27th April 2020 at 06:01 am GMT.
- <https://www.ansible.com/>
- <https://aws.amazon.com/cloudformation/>
- <https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/overview>
- <https://puppet.com/>
- <https://www.chef.io/products/chef-infra/>
- <https://www.ibm.com/cloud/learn/terraform> last accessed 27th April 2020 at 06:01 am GMT.
- <https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code>
- https://en.wikipedia.org/wiki/Infrastructure_as_code
- <https://techbeacon.com/enterprise-it/infrastructure-code-engine-heart-devops>
- <http://www.mynewsdesk.com/dsv-corporate-news/pressreleases/panalpina-powers-digital-data-centers-through-procuring-and-owning-inventory-for-tech-companies-2890866>
- <https://medium.com/cloudnativeinfra/when-to-use-which-infrastructure-as-code-tool-665af289fbde>
- <https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c>