# Materials to learn from

* High level overview - [www.youtube.com](https://www.youtube.com/watch?v=l5k1ai_GBDE)
* Install and start using Terraform - [developer.hashicorp.com](https://developer.hashicorp.com/terraform/tutorials/azure-get-started/infrastructure-as-code)
* Detailed tutorial - [www.youtube.com](https://www.youtube.com/watch?v=V53AHWun17s)

# Terraform setup

* Terraform installation and basic tutorial: [developer.hashicorp.com](https://developer.hashicorp.com/terraform/tutorials/azure-get-started/infrastructure-as-code)
* In order to allow terraform to create service principals we need to create a service principal with the ‘owner’ role which will be used by the Terraform.

Creating such a service principal and assigning it to Terraform to use it is described here: [developer.hashicorp.com](https://developer.hashicorp.com/terraform/tutorials/azure-get-started/azure-build) in the ‘Authenticate using the Azure CLI’ section.

But in that section we are creating a service principal with the ‘Contributor’ role and we need to change it into ‘Owner’

Also it is useful to add some name to the created service principal, for example ‘Terraform’. We can do this by using the ‘az ad sp create-for-rbac’ command with the ‘--name’ parameter.

Additionaly we need to assign Microsoft Entra ‘Application Administrator’ role to that service principal. It is described here how to do this: [docs.azure.cn](https://docs.azure.cn/en-us/entra/identity/role-based-access-control/manage-roles-portal?tabs=admin-center)

# Commands

* Init Terraform in a folder (we need to do this before we run a Terraform file located in that folder):
  + cd directory\_name
  + terraform init
* Create and destroy resources defined in the Terraform file:
  + terraform apply # create or modify resources
  + terraform destroy # destroy resources
* Create and destroy resources defined in the Terraform file with execution plan:
  + terraform plan -out main.tfplan # create execution plan
  + terraform apply # create or modify resources
  + terraform plan -destroy -out main.destroy.tfplan # create execution plan
  + terraform apply main.destroy.tfplan
* Get output value:
  + Terraform output -raw output\_name

# Creating outputs

In terraform we can create outputs. In order to do that we need to write such a command in out .tf file:

output "output\_name" {

value = ‘output\_value’

}

Further sections of this document explains how we can use outputs.

# Using outputs in modules

In Terraform I can create modules. Every module is a directory containing multiple .tf files. For example we might have a structure like this:

terraform-modules/

├── main.tf # Parent module

├── variables.tf

├── outputs.tf

└── modules/

└── s3\_bucket/ # Child module

├── main.tf

├── variables.tf

└── outputs.tf

So here we are creating a child module called ‘s3\_bucket’.

This module can create output values. They are defined in the modules/s3\_bucket/outputs.tf file. This file can look for example like that:

modules/s3\_bucket/outputs.tf :

output "bucket\_arn" {

value = aws\_s3\_bucket.this.arn

}

In order to access this module’s output in the parent module we need to define that module at first in the main.tf file in the parent module:

Main.tf file:

module "my\_s3\_bucket" {

source = "./modules/s3\_bucket"

bucket\_name = "my-unique-bucket-name-1234"

environment = "dev"

}

So here we are creating a module called ‘my\_s3\_bucket’.

Now in the parent module we can use output from the child module by using command:

module.my\_s3\_bucket.bucket\_arn.

We can use that output from the child module for example in output of the parent module. In that case we can create the output of the parent module in the file called output.tf:

Output.tf file:

output "s3\_bucket\_arn" {

value = module.my\_s3\_bucket.bucket\_arn

}

# Accessing outputs in terminal

I can access terraform outputs in terminal. At first we create output in the output.tf file:

output "output\_name" {

value = ‘output\_value’

}

And we can access it in terminal by using command:

Terraform output -raw output\_name

# Saving outputs in a file on local computer

In order to save an output in a file on local computer where we are runnin terraform commands we can use the following command:

#save all the outputs in the text file.

terraform output > outputs.txt

Or to save in JSON:

terraform output -json > outputs.json

Save only a specific output value:

terraform output -raw my\_output\_name > my\_output.txt

# Execute bash script on the VM created using Terraform

When we are creating a VM using Terraform we can also execute a bash script on that VM right after creating it also using Terraform.

We use for that the azurerm\_virtual\_machine\_extension resource. In the settings block, commandToExecute value we are entering our bash script to execute.

If we want to pass a mulitline script we can create a bash script file and use the following syntax:

settings = <<SETTINGS

{

"script": "${filebase64("custom\_script.sh")}"

}

SETTINGS

Where custom\_script.sh is a bash script saved in our Terraform folder.

# Modules and providers

Sometimes in the module we need to specify providers even though we have those providers defined in the root module.