# Signing in and authentication

* Signing in:
  + I can sign in using github account [mbulka44@gmail.com](mailto:mbulka44@gmail.com)
  + I can sign in using my Microsoft account [mbulka44@gmail.com](mailto:mbulka44@gmail.com)
* I am authenticating using the Microsoft Authenticator app, ‘bulka’ - [‘admin@bulka895.onmicrosoft.com’](mailto:‘admin@bulka895.onmicrosoft.com’) account.

# Azure CLI

## Installation

* When installing cli using winget it is downloading an installer in the C:\Users\mbulk\AppData\Local\Temp\WinGet\Microsoft.AzureCLI.2.71.0. I need to go there and click on it in order to install it. I think that it will be easier to use another option for installing.

# Terraform

* Terraform installation and basic tutorial: <https://developer.hashicorp.com/terraform/tutorials/azure-get-started/infrastructure-as-code>

## 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.

# Azure VM creation

## SSH connection

In order to connect to the VM we create using Terraform we can use SSH. In that case we need to use command:

Ssh -i path\_to\_private\_key\_file username@server\_public\_ip

We need to know server’s username, ip and we need to have the private key.

**SSH connection username:**

When creating an Azure VM using Terraform, the admin\_username argument indicates the username which will be used to connect to that VM through SSH:

**SSH connection ip:**

IP address needed to connect to the VM using SSH can be obtained using the azurerm\_linux\_virtual\_machine.<my\_vm\_name>.public\_ip\_address variable, where we replace <my\_vm\_name> with the name of the VM resource in Terraform.

So for example we can create a VM using this command:

resource "azurerm\_linux\_virtual\_machine" "my\_terraform\_vm" {

...

}

And then create the output containing our VM’s IP:

output "public\_ip\_address" {

value = azurerm\_linux\_virtual\_machine.my\_terraform\_vm.public\_ip\_address

}

**SSH connection private key**

We need the private key saved on our local computer in order to connect to the created VM.

When we create a pair of public and private key using this code:

resource "azapi\_resource\_action" "ssh\_public\_key\_gen" {

type = "Microsoft.Compute/sshPublicKeys@2022-11-01"

resource\_id = azapi\_resource.ssh\_public\_key.id

action = "generateKeyPair"

method = "POST"

response\_export\_values = ["publicKey", "privateKey"]

}

Then we save this private key in the local file using this code:

resource "local\_file" "private\_key" {

content = azapi\_resource\_action.ssh\_public\_key\_gen.privateKey

filename = "${path.module}/id\_rsa"

file\_permission = "0600"

}

It will create a new file ./id\_rsa containing the private key in the folder where we run Terraform. Now we can connect to the VM using this command in terminal:

Ssh -i ./id\_rsa username@server\_public\_ip

We can also save the ssh key in the default location on Windows. In that case we set up the filename parameter as follow:

Filename = ‘C:\Users\user\_name\.ssh’

If we save our private key in that default location we can connect to the server using this key without providing the private key path in the command:

Ssh username@server\_public\_ip

Sometimes connecting through SSH might not work when private key is not saved in the default location.

# Azure pipelines

## Self hosted agent set up on Linux VM

Creating a self hosted agent is described here:

- [link](https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/linux-agent?view=azure-devops&tabs=IP-V4) – Microsoft documentation

- [link](https://www.youtube.com/watch?v=Hy6fne9oQJM) – Youtube video

In short we need to:

1. Create an agent pool in devOps website (this can be done from our local machine).
2. Go to that agent pool and click on ‘New agent’. That will display instructions what needs to be done in order to install an agent. We can follow those instructions on our Linux server in order to install an agent there.

We need to choose an authentication option. Below is an example of installing self hosted agent using personal access token.

That token might be generated by clicking on a user icon in the top right hand corner on devOps website and clicking on ‘Personal access tokens’.

We can store presonal access token in the .tfvars file. Then we can use that value in other terraform files:

Terraform.tfvars file:

Azure\_pipelines\_token = <token>

Variables.tf file:

Variable azure\_pipelines\_token {

Type = string

Decsription = „...”

}

Below are commands we need to execute on the Linux server in order to install a self hosted agent:

* wget -P Downloads <https://vstsagentpackage.azureedge.net/agent/4.254.0/vsts-agent-linux-x64-4.254.0.tar.gz>
* mkdir myagent && cd myagent
* tar zxvf ~/Downloads/vsts-agent-linux-x64-4.254.0.tar.gz
* ./config.sh \ # configure the agent

--unattended \

--url <https://dev.azure.com/organization-name> \

--auth pat \

--token <token> \

--pool „name\_of\_the\_pool” \

--agent „name\_of\_the\_agent”

* ./run.sh # start the agent

We can use the ‘Unattended config’ in order to set up agent automatically. It is described in the Microsoft documentation.

We need to add the –unattended flag and other flags mentioned in the documentation to the ./config.sh command.

## Using self hosted agent in the CI/CD pipeline

In order to use our self hosted agent in the pipeline we need to specife the pool name in the YAML file, the ‘pool’ value. That is a name of the agent pool created on the website. It is in the ‘Agent pools’ section.