TERRAFORM AKS GUIDE

**Project structure**

**1- modules**: represent here in this layout the Terraform modules (general re-used functions) . In this lab, we have basically 4 modules:   
– aks\_cluster: the main unit providing the AKS service  
– aks\_identities: the cluster identity unit that manage the cluster service principal   
– aks\_network: Create the cluster Virtual Network and subnetwork on Azure  
**–** log\_analytics: Formally Azure Operational Insight is the unit that manages logs and cluster health checks  
  
**2- Deployment:** is the main function of this layout, responsible of the AKS Kubernetes cluster deployment on Azure.  
In **main.tf** we define the Terraform modules already created in /modules sub-folder with the appropriate inputs defined in **variables.tf** or in a terraform.tfvars file

### Terraform modules

With Terraform, you can put a bunch of code inside of a Terraform module and reuse that module in multiple places throughout your code. Instead of having the same code copy/pasted in the staging and production environments, you’ll be able to have both environments reuse code from the same module.  
This is a big deal. Modules are the key ingredient to writing reusable, maintainable, and testable Terraform code.

Every Terraform configuration has at least one module, known as its root module (the /deployment in this lab context), which consists of the resources defined in the .tf files in the main working directory.

#### ****Terraform module structure****

In this lab we have a well defined structure of the TF Modules. Let’s go through the aks\_identities module as an example:

1. main.tf: the aks cluster resources are packaged in the main.tf file
2. variables.tf: In Terraform, modules can have input parameters, too. To define them, you use a mechanism input variables.
3. output.tf: In Terraform, a module can also return values. Again, this is done using a mechanism: output variables.

#### ****Initialization****

Provider initialization is one of the actions of terraform init.   
Running this command will download and initialize any providers that are not already initialized

terraform init

## Set up Azure storage to store Terraform state

Terraform tracks state locally via the terraform.tfstate file.   
This pattern works well in a single-person environment.   
However, in a multi-person environment, Azure Storage is used to track the tf state.

In this section, we will see how to do the following tasks:

* Retrieve storage account information (account name and account key)
* Create a storage container into which Terraform state information will be stored.

1. In the Azure portal, select All services in the left menu.
2. Select Storage accounts.
3. On the Storage accounts tab, select the name of the storage account into which Terraform is to store state. For example, you can use the storage account created when you opened Cloud Shell the first time. The storage account name created by Cloud Shell typically starts with cs followed by a random string of numbers and letters. Take note of the storage account you select. This value is needed later.
4. On the storage account tab, select Access keys.
5. Make note of the key1 key value. (Selecting the icon to the right of the key copies the value to the clipboard.)
6. Using your terminal, create a container in your Azure storage account. Replace the placeholders with appropriate values for your environment.
7. az storage container create -n tfstate --account-name <YourAzureStorageAccountName> --account-key \<YourAzureStorageAccountKey>

## Create AKS using Terraform

In this section, you see how to use the terraform init command to create the resources defined the configuration files you created in the previous sections.

1. In your local terminal, initialize Terraform. Replace the placeholders with appropriate values for your environment

cd src/deployment

terraform init -backend-config="storage\_account\_name=<YourAzureStorageAccountName>" -backend-config="container\_name=tfstate" -backend-config="access\_key=<YourStorageAccountAccessKey>" -backend-config="key=codelab.microsoft.tfstate"

The terraform init command displays the success of initializing the backend and provider plug-in:

2. Export your service principal credentials. Replace the placeholders with appropriate values from your service principal

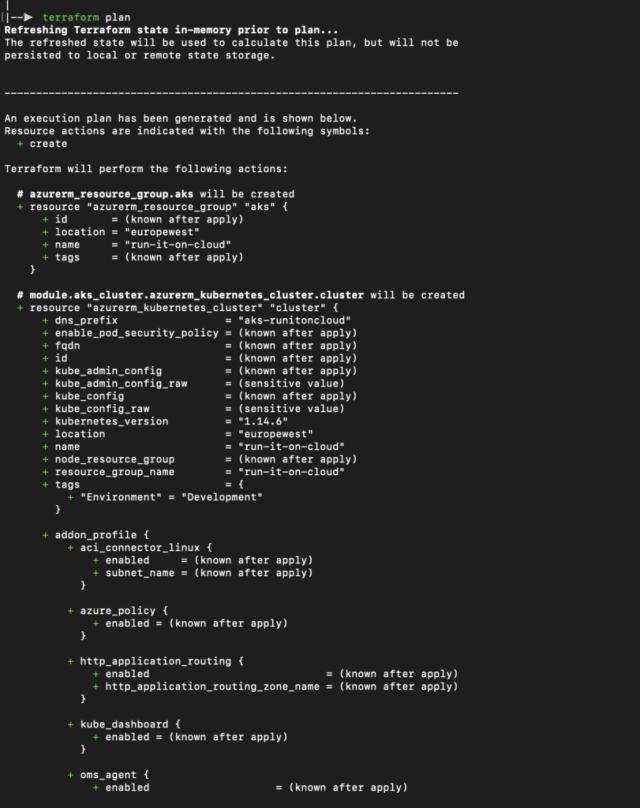
export TF\_VAR\_client\_id=<service-principal-appid>

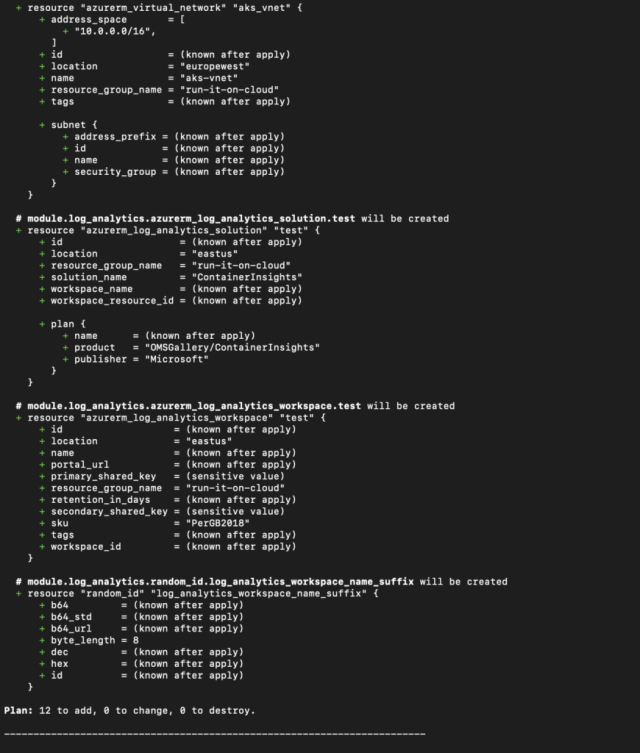
export TF\_VAR\_client\_secret=<service-principal-password>

3. Run the terraform plan command to create the Terraform plan that defines the infrastructure elements.

terraform plan -out out.plan

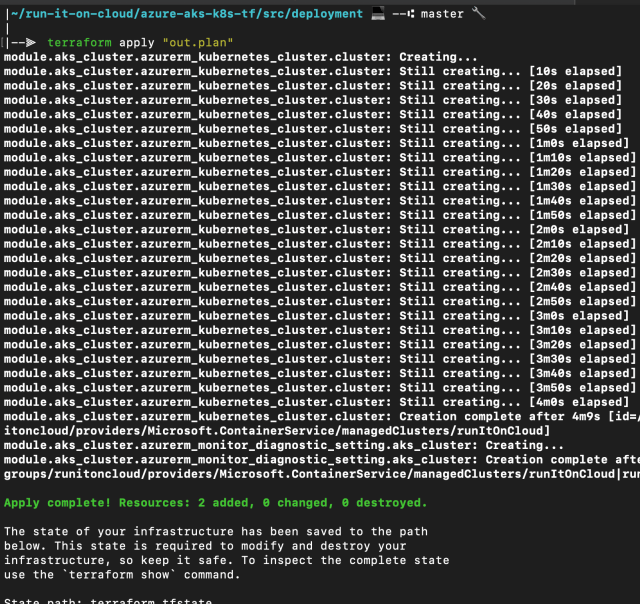
The terraform plan command displays the resources that will be created when you run the terraform apply command.  
There’s an Terraform plan output example:

Terraform plan

Terraform plan output

4. Run the terraform apply command to apply the plan to create the Kubernetes cluster. The process to create a Kubernetes cluster can take several minutes.

terraform apply out.plan



## Test the Kubernetes AKS cluster

To manage a Kubernetes cluster, you use kubectl , the Kubernetes command-line client. If you use Azure Cloud Shell, kubectl is already installed. To install kubectl locally, use the  command to install .

az aks install-cli

To configure kubectl to connect to your Kubernetes cluster, use the az aks get-credentials command. This command downloads credentials and configures the Kubernetes CLI to use them. (make sure to use your own created resource group and AKS names)

az aks get-credentials –resource-group resource\_group\_name – name aks\_cluster\_name