Configuring DataSunrise with Terraform Template on Microsoft Azure Cloud

Instruction Manual

March, 2022

Table of Contents

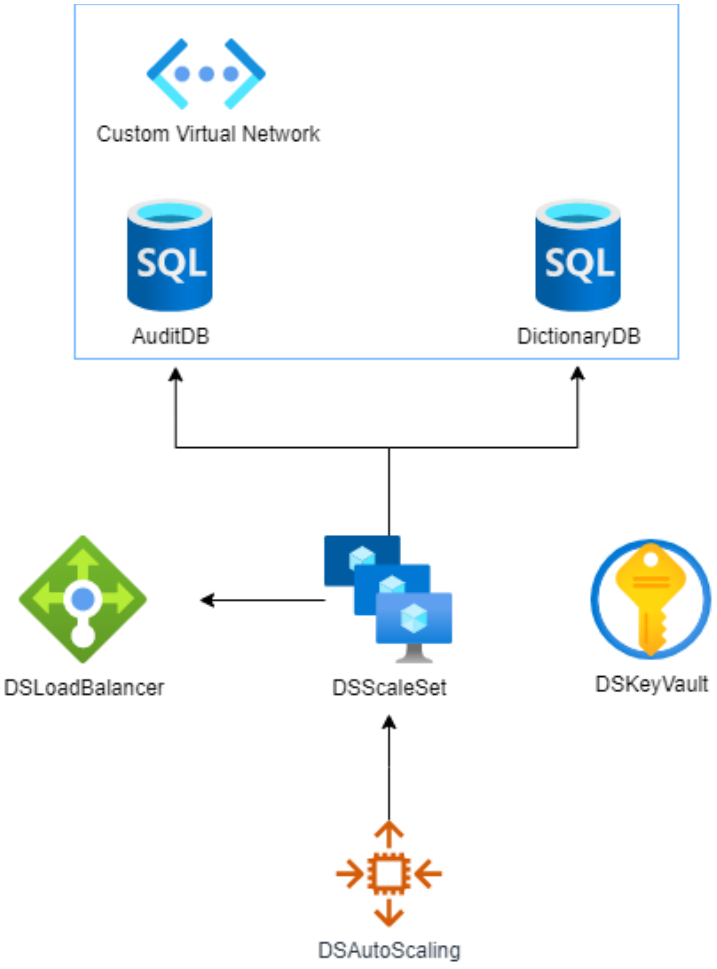
1. Introduction....3
1.1 Description and structural scheme of a Terraform....3
2. Prerequisites....4
3. Deploying a Terraform Template....4
4. Connecting to the DataSunrise Web UI....8
5. Summary....9
6. Debugging....9

1. Introduction

Manual deployment of a High Availability (HA) configuration requires accurate settings implementation with all the dependencies applied correctly. Moreover, the product installation process might take much more time by also including the maintenance of a reliable and stable solution to run the production environment. To avoid all the possible issues and eliminate the presuming inconveniences, DataSunrise provides the dedicated script for HA infrastructure deployment within the Microsoft Azure service based on a Terraform template. The deployment process is automated and does not require any manual adjustments.

1.1 Description and structural scheme of a Terraform

The following picture displays the most important objects created by Terraform as they are listed in the template, creation order is parallel, considering objects' dependencies (marked with arrows).



1. Custom Virtual Network (Microsoft.Network/ virtualNetworks): Virtual Network where the subnets are stored. Needed for free communication between Azure resources;

2. DictionaryDb (Microsoft.DBforPostgreSQL/servers): Azure database instance used to store DataSunrise settings (Dictionary);

3. AuditDB (Microsoft.DBforPostgreSQL/servers): Azure database instance used to store DataSunrise's audit journal and other journals (Audit Storage);

4. DSLoadBalancer (Microsoft.Network/loadBalancers): Load Balancer;

5. DSScaleSet (Microsoft.Compute/virtualMachineScaleSets): Virtual Machine Scale Set, includes configuration of a failover cluster;

6. DSKeyVault (Microsoft.KeyVault/vaults): Key Vault which is used to store credentials for ARM resources;

7. DSAutoScaleSettings (Microsoft.Insights/Autoscalesettings): Auto scaling settings that define how new VMs of the failover cluster are deployed.

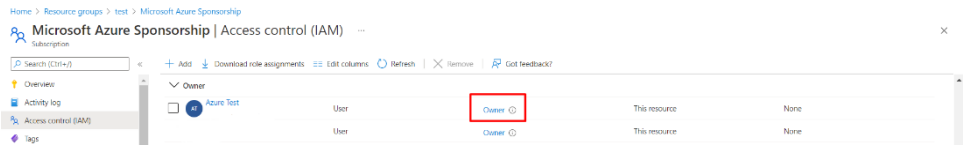
Note: there are different dependent resources that will be created as well. Describing them here will not bring any useful information.

2. Prerequisites

There are some entities that should exist in your Azure environment before you start the deployment process. Here is the list of required items:

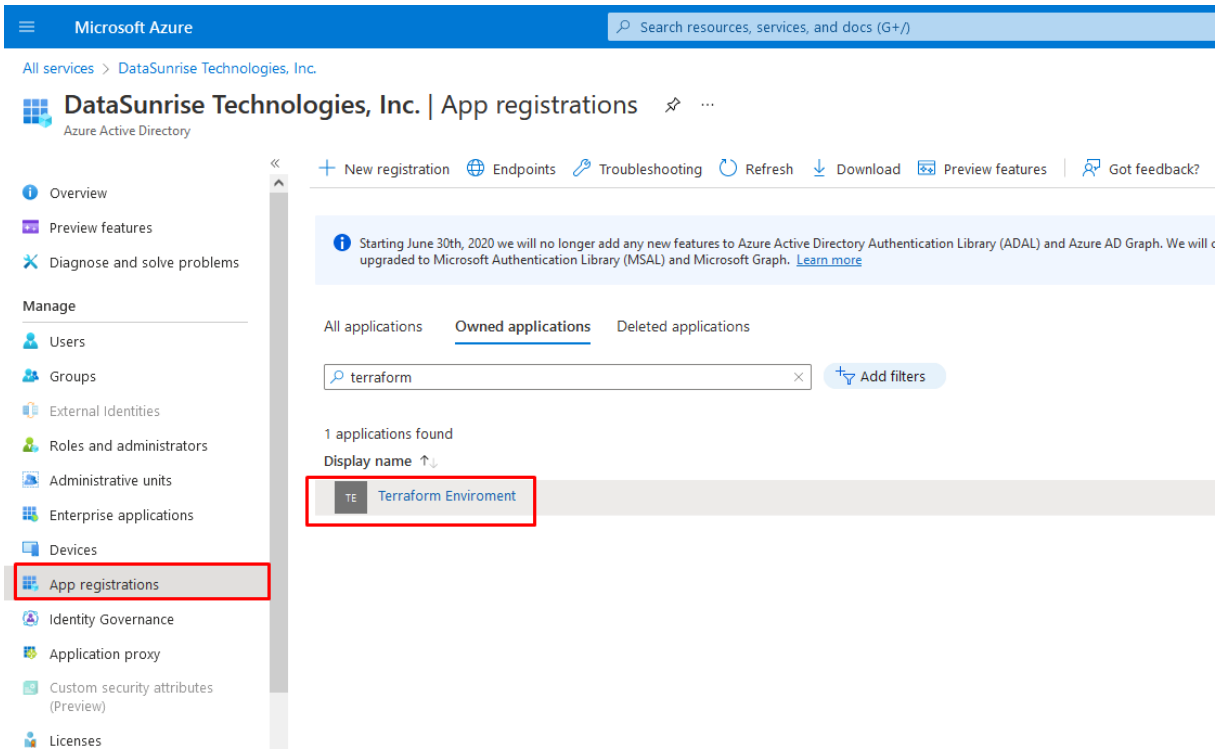
- AZ CLI is installed;
- Downloaded/Installed Terraform CLI;
- Storage account with an accessible container with DataSunrise build inside or any place from where you can download the DS build;

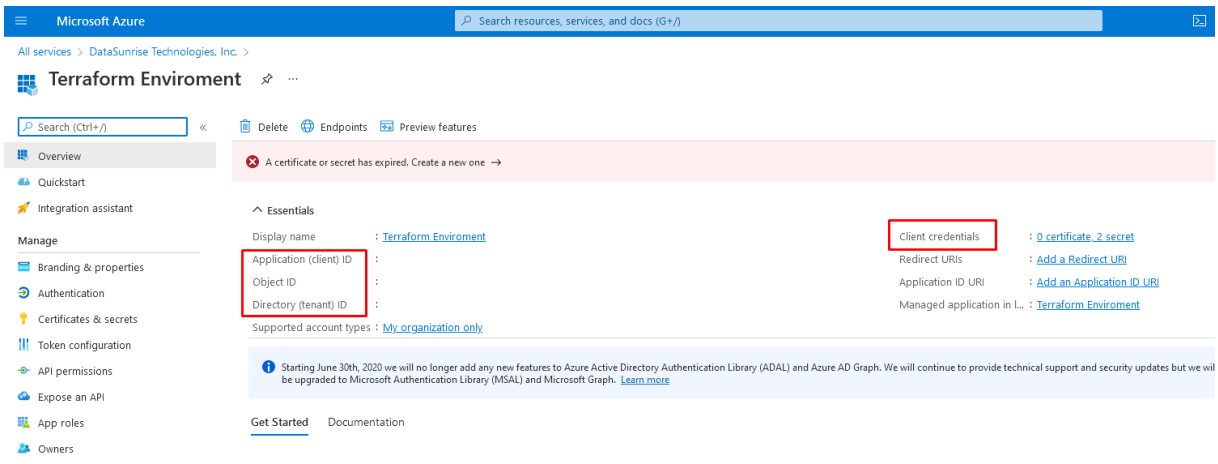
Important: your Azure account subscription should be assigned to the Owner role to enable usage of the Managed Identity resource to connect to Azure CLI and to deploy required Azure resources for successful DataSunrise implementation. You can check it in **Subscriptions -> <your-subscription> -> Access control -> Role assignments**:



3. Deploying a Terraform Template

1. Register or select an existing app and provide *"CLIENT_ID", "CLIENT_SECRET", "SUBSCRIPTION_ID", "TENANT_ID":*





2. To deploy DataSunrise in HA configuration, Download terraform scripts from the repository
3. Open file *terraform.tfvars* and replace *xxx xxx xxx* there with values that correspond to your environment:

```
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# DataSunrise Cluster for Microsoft Azure Cloud
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Please replace xxxxxxxxx with values that correspond to your environment
# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

# --------------------------------------------------------------------------------
# Virtual Machine Configuration
# --------------------------------------------------------------------------------
prefix                          = "xxxxxxxxx"
#description = "Name that will be used as the prefix to the resources' names that will be created by the Terraform script (only in lower case, not more than 15 symbols

location                        = "xxxxxxxxx"
#description = "The Azure Region in which all resources should be created."

admin_Username                  = "xxxxxxxxx"
#default = "dsuser"
#description = "VM User Account Name. For example, dsuser"

admin_Password                  = "xxxxxxxxx"
#description = "VM User Password"

vmcount                         = "xxxxxxxxx"
#default = "1"
#description = "Count of instances of the VM being created."

vmSize                          = "xxxxxxxxx"
#description = "Virtual Machine type for DataSunrise instance. Depends on the Location and Availability Set"

# --------------------------------------------------------------------------------
# DataSunrise Configuration
# --------------------------------------------------------------------------------

link_To_DS_Build                = "xxxxxxxxx"
#description = "Url of the DataSunrise distribution `.run` package. #Make sure that this URL will be accessible from your PC"

DS_Admin_Password               = "xxxxxxxxx"
#description = "DataSunrise admin's password. The password must contain at least 8 characters, lower and upper case, numbers and special characters."

DS_License_Key                  = "xxxxxxxxx"
#description = "!!!Important!!! for correct key substitution, if there are double quotes in the key, then it is necessary to make a concatenation using a backslash (\"

# --------------------------------------------------------------------------------
```

4. Open console and set these environment variables:

```
Set   ARM_CLIENT_ID="SET-CLIENT-KEY-HERE"

Set  ARM_CLIENT_SECRET="SET-CLIENT-SECRET-HERE"

Set  ARM_SUBSCRIPTION_ID="SET-SUBSCRIPTION-ID-HERE"

Set  ARM_TENANT_ID="SET-TENANT-ID-HERE"
```

Important: on Windows, you need to copy the downloaded executable file, *terraform.exe*, to the *scripts* folder.

5. Run the *terraform init* command in the console

   The *terraform init* command is used to initialize a working directory containing Terraform configuration files. This is the first command that should be ran after setting up a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

```
C:\azuretf>terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/template from the dependency lock file
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Using previously-installed hashicorp/template v2.2.0
- Using previously-installed hashicorp/azurerm v2.99.0
- Using previously-installed hashicorp/random v3.1.2

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

6. Run the *terraform plan* command

The *terraform plan* command creates an execution plan which lets you preview the changes that Terraform plans to make to your infrastructure. By default, when Terraform creates a plan it does the following:

- Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date;
- Compares the current configuration to the prior state and notes any differences;
- Proposes a set of change actions that should, if applied, make the remote objects match the configuration.

```
Plan: 0 to add, 1 to change, 0 to destroy.

  Warning: Argument is deprecated

    with azurerm_key_vault.DS_Key_Vault,
    on main.tf line 83, in resource "azurerm_key_vault" "DS_Key_Vault":
    83:   soft_delete_enabled       = true

  Azure has removed support for disabling Soft Delete as of 2020-12-15, as such this field is no longer configurable
  and can be safely removed. This field will be removed in version 3.0 of the Azure Provider.

  (and 7 more similar warnings elsewhere)


Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if
you run "terraform apply" now.

C:\azuretf>
```

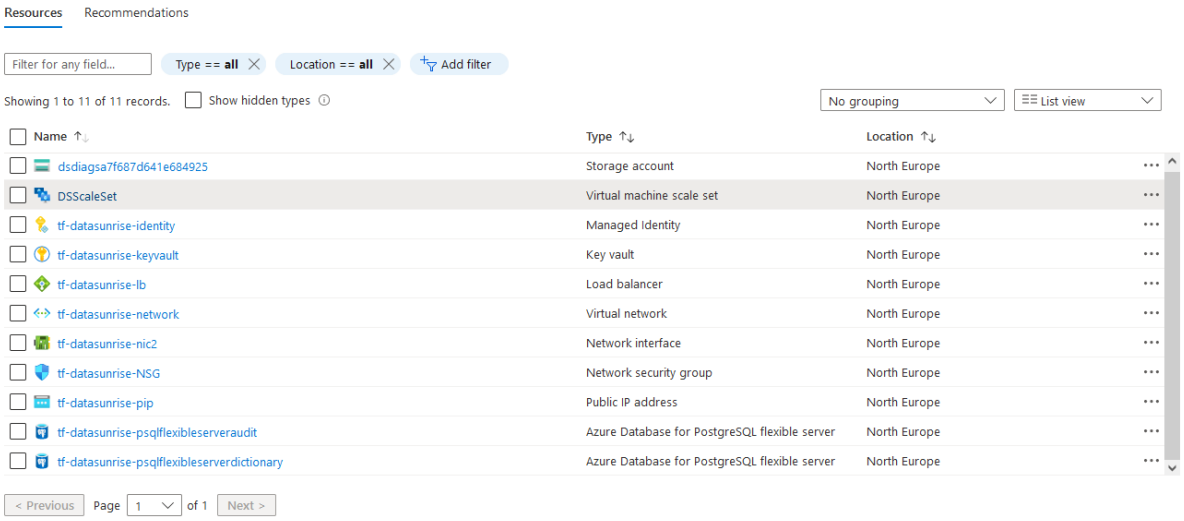
7. Run the *terraform apply* command

The *terraform apply* command executes the actions proposed in a Terraform plan.

After executing *terraform apply*, the resource creation takes about 5-10 minutes, and further configuration and installation of additional libraries take about 15-20 minutes.

You can see all the created resources in the created resource group with the name that was set in the *terraform.tfvars* – prefix

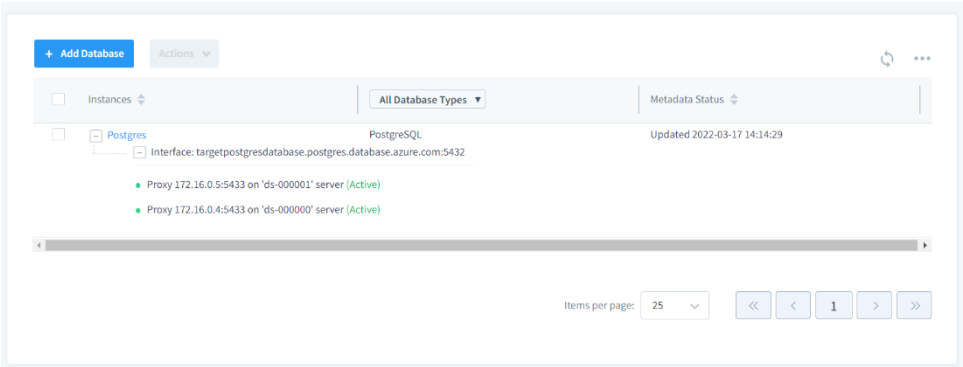
All the resources are successfully deployed and listed inside the resource group:



4. Connecting to the DataSunrise Web UI

To connect to the DataSunrise's Web Console, use *LoadBalancer public IP address*. It will automatically connect to one of the configured nodes. You also need to use this IP address to connect to your target database through a proxy and to connect to your Virtual Machine using SSH.

The Target DB will be automatically added to the DataSunrise's settings:



5. Summary

DataSunrise Terraform template is successfully deployed. DataSunrise Configuration and Audit databases are configured on a PostgreSQL server. Public Load Balancer IP address is configured to access the Web Console, target databases and to connect via SSH.

HA configuration is set up. Autoscaling settings are applied depending on the amount of the alive DataSunrise servers. Virtual machine contributor role is assigned to the virtual machine scale set to access the Azure CLI.

To delete the created environment, just enter the *terraform destroy* command

The *terraform destroy* command is a convenient way to destroy all remote objects managed by a particular Terraform configuration.

While you would not typically want to destroy long-lived objects in a production environment, Terraform is sometimes used to manage ephemeral infrastructure for development purposes, in which case you can use *terraform destroy* to conveniently clean up all of those temporary objects once you are finished with your work.

6. Debugging

If your server is unavailable for a long time, you can connect to the created virtual machine via SSH using the username and password specified in the *terraform.tfvars* file. Having connected, run the command *sudo cat /var/log/cloud-init-output.log* and see the reason for incorrect installation of the server.