Terraform Remote State Azure Lab 1.12

.

Expected outcome :
Store Backend State in Azure Storage Account

One of the things that HashiCorp Terraform requires to work is the ability to save and store the Terraform State Management (`.tfstate`) file. By default, the state management file is stored locally where the Terraform CLI is run. While this does provide the necessary functionality to use Terraform for infrastructure deployments, at a minimum, it's not ideal for larger deployments using CI/CD (Continuous Integration and Continuous Deployment) deployment pipelines or in other shared environments. This is the reason that Terraform supports the configuration of the Backend settings that tell it where to store / retrieve the state management file. This article will show you how to store the Terraform State Management (`.tfstate`) file in a Microsoft Azure Storage Account.
There are a couple steps necessary to configuring the Terraform Backend to store the state management file in an Azure Storage Account. Let's take a look at these steps and how to configure the Terraform Backend to get this done!

Steps

Step 1: Create Azure Storage Account

The first step to configuring Terraform Backend to store the state management file in Azure Storage is to create an Azure Storage Account and a container in the account that will store the files.

The Azure Storage Account and container can easily be created using the Azure Portal. However, the following is the basic Azure CLI commands that you can use to create the Azure Storage Account and container needed:

```
# create Azure resource group

az group create --name tf-state-rg  --location uksouth



# create Azure storage account

az storage account create --name tfstatesaXX  --location uksouth   --resource-group tf-state-rg



# create Container in Azure storage account

az storage container create --accountname tfstatesaXX--name tfstate --public-access off  --account-key <account-key>
```

These Azure CLI commands perform the following steps:

1. Create an Azure Resource Group to organize the Storage Account within. This will generally be a resource group just for this Azure Storage Account since it will be secure and managed separately from any other resources.
2. Create the Azure Storage Account to use.
3. Create the Container where the state file will be stored. In this example, the container was named `tfstate`.

Notice how I'm not saying to create the backend Azure Storage Account with Terraform? The reason for this is that Terraform needs a place to store state files in order to function. Due to this reason, the backend Azure Storage Account needs to exist before Terraform projects can be deployed while using it as their backend. This is a sort of circular dependency of Terraform backend configuration, so it's just easier to manually create the Azure Storage Account, or do it via Azure CLI script.

Also, keep in mind that you'll want to secure the backend Azure Storage Account separately from other resources in your Azure Subscription. The Terraform state

files will likely contain keys, passwords and other secrets depending on your usage of Terraform providers and code. Storing these secrets in the Terraform state (`.tfstate`) file is just a necessity of how Terraform state works. As a result, you'll probably want to secure the Terraform Backend Azure Storage Account so that your Terraform deployments are the only process that can read and write to them. This extra security will prevent any other users from being able to access any secrets within the state files.

Step 2: Configure Terraform Backend to use Azure Storage Account

To use the Azure Storage Account as the Backend for storing the Terraform state files, you will need to create and configure a `backend` block. When configuring the `backend` block, Terraform will then know to use this configuration for the backend state file storage instead of the default local storage.
The following is a basic example of configuring the `backend` block to use an Azure Storage Account as was created previously, and the `azurerm` provider accordingly:

```
terraform {
 backend "azurerm" {
   resource_group_name  = "tf-state-rg"
   storage_account_name = "tfstatesaXX"
   container_name       = "tfstate"
   key                  = "terraform.tfstate"
 }
}
```

This `backend` block specifies `azurerm` as the type of backend to use. This essentially tells Terraform to use the backend state storage provided by the `azurerm` provider.
The Terraform `backend` block is also configured using the following arguments:

- `resource_group_name` – This specifies the Azure Resource Group where the Azure Storage Account is located in the Azure Subscription. In this example, the resource group is named `tf-state-rg`.

- `storage_account_name` – This is the name of the Azure Storage Account to use for the backend state storage. In this example, the Azure Storage Account is named `tfstatesaXX`.
- `container_name` – This is the name of the Container within the Azure Storage Account where the state files will be stored. In this example, the container is named `tfstate`.
- `key` – The Key specifies the file name to use for the Terraform State (`tfstate`) file. In this example, we're naming the file `terraform.tfstate`.

Also, the `backend` block example is not defining any Storage Account Key or other authentication method. The reason for this is it will automatically use the same Azure Subscription credentials that are being authenticated for the `azurerm` provider.
The Azure Subscription credentials are passed to Terraform and used automatically by the backend configuration when you use the following methods of authenticating Terraform with Azure:

- **Azure CLI** – When you use the `az login` command to login to Azure from the terminal prior to running the Terraform `plan` or `apply` commands then the Terraform `azurerm` provider will automatically grab the credentials from the Azure CLI for it to authenticate with Azure and access the Storage Account.
- **Azure Service Principal** – When you configure the `azurerm` provider in your Terraform code to use a Service Principal credentials (client id and secret) then the `backend` configuration will automatically use those same credentials to authenticate with Azure and access the Storage Account.

Usually, the `backend` block is placed in a file named `backend.tf` in your Terraform project, but it could also be placed in the `main.tf` or any other `.tf` file in your project as needed.

Step 3: Run Terraform Commands

Once the Terraform `backend` block is configured, you can then just run any Terraform CLI commands as normal. Instead of reading and writing the state (`.tfstate`) files from local storage, it will now be configured to read and write state to the Azure Storage Account that is configured in the `backend` block.

```
terraform init
```

```
terraform plan

terraform apply
```

Summary

Configuring a Terraform project to use an Azure Storage Account as the backend storage for the state (`.tfstate`) files provides several benefits, including: team collaboration on running Terraform deployments, better version control of deployment state, and disaster recovery capabilities in Azure Storage for storing and recovering state files when necessary. Using a shared or cloud based `backend` configuration is an essential step when setting up enterprise ready Terraform deployments, as well as configuring effective CI/CD pipelines for managing Terraform deployments.

Keep in mind that if you are using Terraform Cloud, then this isn't necessary as your `.tfstate` files will be stored in Terraform Cloud instead.