Provision an AKS cluster (Azure)

Terraform

Reference this often? Create an account to bookmark tutorials. The Azure Kubernetes Service (AKS) is a fully managed Kubernetes service for deploying, managing, and scaling containerized applications on Azure.

In this tutorial, you will deploy a 2 node AKS cluster on your default VPC using Terraform then access its Kubernetes dashboard.

Warning! If you're not using an account that qualifies under the Azure free tier, you may be charged to run these examples. The most you should be charged should only be a few dollars, but we're not responsible for any charges that may incur.

Why deploy with Terraform?

While you could use the built-in Azure provisioning processes (UI, CLI) for AKS clusters, Terraform provides you with several benefits:

- Unified Workflow - If you are already deploying infrastructure to Azure with Terraform, your AKS cluster can fit into that workflow. You can also deploy applications into your AKS cluster using Terraform.

- Full Lifecycle Management - Terraform doesn't only create resources, it updates, and deletes tracked resources without requiring you to inspect the API to identify those resources.

- Graph of Relationships - Terraform understands dependency relationships between resources. For example, an Azure Kubernetes cluster needs to be associated with a resource group, Terraform won't attempt to create the cluster if the resource group failed to create.

Prerequisites

The tutorial assumes some basic familiarity with Kubernetes and kubectl but does not assume any pre-existing deployment.

It also assumes that you are familiar with the usual Terraform plan/apply workflow. If you're new to Terraform itself, refer first to the Getting Started tutorial.

For this tutorial, you will need

- an Azure account
- a configured Azure CLI
- kubectl

Configured Azure CLI
kubectl

In order for Terraform to run operations on your behalf, you must install and configure the Azure CLI tool. To install the Azure CLI, follow these instructions or choose a package manager based on your operating system.

macOS install with Homebrew
Windows install with Chocolatey

You can also use the package manager homebrew to install the Azure CLI.

```
$ brew install azure-cli
```

After you've installed the Azure CLI, login into Azure by running:

```
$ az login
```

Set up and initialize your Terraform workspace

In your terminal, clone the following repository. It contains the example configuration used in this tutorial.

```
$ git clone https://github.com/hashicorp/learn-terraform-provision-aks-cluster
```

You can explore this repository by changing directories or navigating in your UI.

```
$ cd learn-terraform-provision-aks-cluster
```

In here, you will find three files used to provision the AKS cluster.

1. aks-cluster.tf provisions a resource group and an AKS cluster. The default_node_pool defines the number of VMs and the VM type the cluster uses.

```
resource "azurerm_kubernetes_cluster" "default" {

  name                = "${random_pet.prefix.id}-aks"
  location            = azurerm_resource_group.default.location
  resource_group_name = azurerm_resource_group.default.name
  dns_prefix          = "${random_pet.prefix.id}-k8s"
  kubernetes_version  = "1.26.3"

  default_node_pool {
    name            = "default"
    node_count      = 2
```

```
    vm_size         = "Standard_D2_v2"
    os_disk_size_gb = 30
  }

  service_principal {
    client_id     = var.appId
    client_secret = var.password
  }

  role_based_access_control_enabled = true

  tags = {
    environment = "Demo"
  }
}
```

27. variables.tf declares the appID and password so Terraform can use reference its configuration

28. terraform.tfvars defines the appId and password variables to authenticate to Azure

29. outputs.tf declares values that can be useful to interact with your AKS cluster

30. versions.tf sets the Terraform version to at least 0.14 and defines the required_provider block

Create an Active Directory service principal account

There are many ways to authenticate to the Azure provider. In this tutorial, you will use an Active Directory service principal account. You can learn how to authenticate using a different method here.

First, you need to create an Active Directory service principal account using the Azure CLI. You should see something like the following.

```
$ az ad sp create-for-rbac --skip-assignment
{
  "appId": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",
  "displayName": "azure-cli-2019-04-11-00-46-05",
  "name": "http://azure-cli-2019-04-11-00-46-05",
  "password": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",
  "tenant": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
}
```

Update your terraform.tfvars file

Replace the values in your terraform.tfvars file with your appId and password. Terraform will use these values to authenticate to Azure before provisioning your resources. Your terraform.tfvars file should look like the following.

```
# terraform.tfvars
appId    = "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
password = "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
```

Initialize Terraform

After you have saved your customized variables file, initialize your Terraform workspace, which will download the provider and initialize it with the values provided in your terraform.tfvars file.

```
$ terraform init
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Reusing previous version of hashicorp/random from the dependency lock file
- Installing hashicorp/azurerm v3.67.0...
- Installed hashicorp/azurerm v3.67.0 (signed by HashiCorp)
- Installing hashicorp/random v3.5.1...
- Installed hashicorp/random v3.5.1 (signed by HashiCorp)

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.
```

Provision the AKS cluster

In your initialized directory, run terraform apply and review the planned actions. Your terminal output should indicate the plan is running and what resources will be created.

Note

If you get an error that the VM size of Standard_D2_v2 is not allowed in your subscription, you may have reached a resource limit. Refer to the AKS VM size restrictions and region availability documentation for more information.

```
$ terraform apply
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  ## ...

Plan: 3 to add, 0 to change, 0 to destroy.
```

```
## ...
```

You can see this terraform apply will provision an Azure resource group and an AKS cluster. Confirm the apply with a yes.

This process should take approximately 5 minutes. Upon successful application, your terminal prints the outputs defined in aks-cluster.tf.

```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

kubernetes_cluster_name = light-eagle-aks
resource_group_name = light-eagle-rg
```

Configure kubectl

Now that you've provisioned your AKS cluster, you need to configure kubectl.

Run the following command to retrieve the access credentials for your cluster and automatically configure kubectl.

```
$ az aks get-credentials --resource-group $(terraform output -raw resource_group_name) --name $(terraform output -raw kubernetes_cluster_name)
Merged "light-eagle-aks" as current context in /Users/dos/.kube/config
```

The resource group name and Kubernetes Cluster name correspond to the output variables showed after the successful Terraform run.

Access Kubernetes Dashboard

To verify that your cluster's configuration, visit the Azure Portal's Kubernetes resource view. Azure recommends using this view

over the default Kubernetes dashboard, since the AKS dashboard add-on is deprecated for Kubernetes versions 1.19+.

Run the following command to generate the Azure portal link.

```
$ az aks browse --resource-group $(terraform output -raw resource_group_name) --name $(terraform output -raw kubernetes_cluster_name)
Kubernetes resources view on https://portal.azure.com/#resource/subscriptions/aaaaa/resourceGroups/light-eagle-rg/providers/Microsoft.ContainerService/managedClusters/light-eagle-aks/workloads
```

Go to the URL in your preferred browser to view the Kubernetes resource view.

Clean up your workspace

Congratulations, you have provisioned an AKS cluster, configured kubectl, and visited the Kubernetes dashboard.

If you'd like to learn how to manage your AKS cluster using the Terraform Kubernetes Provider, leave your cluster running and continue to the Kubernetes provider tutorial.

Note

This directory is only used to provision a AKS cluster with Terraform. By keeping the Terraform configuration for provisioning a Kubernetes cluster and managing a Kubernetes cluster resources separate, changes in one repository don't affect the other. In addition, the modularity makes the configuration more readable and enables you to scope different permissions to each workspace.
If not, remember to destroy any resources you create once you are done with this tutorial. Run the destroy command and confirm with yes in your terminal.

```
$ terraform destroy
```

Next steps

For more information on the AKS resource, visit the Azure provider documentation.

For steps on how to manage Kubernetes resources your AKS cluster or any other already created Kubernetes cluster, visit the Kubernetes provider tutorial.

To use run triggers to deploy a Kubernetes Cluster, Consul and Vault on Google Cloud, visit the Deploy Consul and Vault on a Kubernetes Cluster using Run Triggers tutorial.