Azure VNet injection Workspace Setup Guide (Terraform)

Created by: [Ognjen Pantelic](mailto:ognjen.pantelic@databricks.com) [Gergelj Kiš](mailto:gergelj.kis@databricks.com)// Last reviewed: Sep 22, 2025

## Requirements

* Terraform is installed on your local machine: [[link]](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli#install-terraform)
* Azure CLI is installed on your local machine: [[MAC]](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-macos?view=azure-cli-latest#install-with-homebrew) or [[Windows]](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest&pivots=winget)
* Azure CLI configured with appropriate credentials
* Databricks account created
* Databricks account admin access
* Contributor rights to your Azure subscription (Contributor rights on the resource group level are not sufficient, as Databricks provisioning creates resources in a separate managed resource group, which requires subscription-level access.)

## Before you begin

In this deployment, we define key configuration values, such as subscription ID, resource group location, CIDR block, asset naming, and others, as variables. This keeps our code organized and makes it easy to adjust settings without changing the core infrastructure definitions. You can choose to define these variables directly or reference them from a separate configuration file for better modularity. In this document, we will create a configuration file to store them separately.

Names of the files you will create to hold the code are marked in green\_text.

## 1. Set up Terraform configuration

### Overview:

This step initializes Terraform for your project. Terraform is the Infrastructure-as-Code (IaC) tool used to define and provision all the resources needed for your Databricks workspace on Azure.

Open your terminal and start typing:

| mkdir databricks-az-terraform  cd databricks-az-terraform |
| --- |

This will create our folder that will hold all the files we plan to create.

### Authentication Setup for Terraform

To successfully deploy infrastructure using Terraform, you must be authenticated with valid Azure credentials. Terraform uses the azurerm provider to provision resources, and it relies on your Azure identity to do so.

## Options to provide Azure credentials:

### Option 1. Interactive user login (for users)

| az login |
| --- |

This command opens a browser for user authentication, and it is commonly referred to as U2M (User-to-machine) authentication. This command is sufficient for all operations in this document.

### Option 2. Service principal login (for automation, CI/CD)

Choose this option if you want to deploy the Terraform script to a Git repository and integrate it into your CI/CD processes after completing this guide. It is the recommended approach for automation in non-interactive environments such as pipelines or scripts.

Steps to Create a Service Principal via Azure CLI:

1. Log in to Azure via Azure CLI

| az login |
| --- |

This command opens a browser to authenticate your Azure user account.

2. (Optional) Choose the Target Subscription

If you have multiple subscriptions, set your target subscription:

| az account set --subscription "<subscription-id>" |
| --- |

You can find your subscription ID with:

| az account show |
| --- |

3. Create the Service Principal

Use the following command to create a service principal, specifying the name, role, and scope:

| az ad sp create-for-rbac --name "<sp-name>" --role <role> --scopes /subscriptions/<subscription-id> |
| --- |

* <sp-name>: Desired service principal name.
* <role>: e.g. Contributor, Reader, Owner.
* <subscription-id>: Your Azure Subscription ID.

The command outputs JSON with appId, password, and tenant.

Important: Save the password (client secret) immediately; you cannot retrieve it later.

4. Use the Newly Created SP Credentials

You can now use the output values:

* appId for the username
* password as the client secret
* tenant as the tenant ID

For authentication in automation (like CI/CD or scripts), use:

| az login --service-principal -u <appId> -p <password> --tenant <tenant> |
| --- |

For more information on creating a Service Principal, visit: <https://learn.microsoft.com/en-us/cli/azure/azure-cli-sp-tutorial-1?view=azure-cli-latest&tabs=bash>

## 2. Configure Terraform providers

Overview:

This step sets up the Terraform providers. Providers are plugins that allow Terraform to interact with specific APIs, such as Azure and Databricks.

What’s happening?

The providers.tf file specifies the azurerm provider (to manage Azure resources) and the Databricks provider (to interact with Databricks APIs). It also configures the region and account details required for deployment.

*Obs: Make sure you use a Databricks Terraform provider version >=1.25.0*

providers.tf

| terraform {  required\_providers {  databricks = {  source = "databricks/databricks"  version = ">=1.25.0"  }  azurerm = {  source = "hashicorp/azurerm"  }  }  }  provider "azurerm" {  subscription\_id = var.azure\_subscription\_id  features {}  } |
| --- |

## 

* var.azure\_subscription\_id: Azure subscription identifier used by the azurerm provider.

## 3. Create the Azure resource group

An Azure resource group is a container that holds related resources for a solution (Workspace) and allows them to be managed as a unit. It simplifies organizing, deploying, updating, and deleting resources collectively according to their lifecycle. Resource groups make it easier to coordinate changes and control access for all grouped resources.

azure.tf

| resource "azurerm\_resource\_group" "this" {  name = var.resource\_group\_name  location = var.location  tags = var.tags  } |
| --- |

* var.resource\_group\_name: Name assigned to the Azure resource group.
* var.location: Azure region where the resource group will be deployed.
* var.tags: Key-value pairs for categorizing and organizing the resource group with tags.

## 4. Create Required Network Resources

Overview:

Databricks workspaces on Azure require a properly configured Virtual network (VNet). Depending on your specific use case, you can either:

1. Use an existing VNet that meets Databricks’ requirements.
2. Create a new VNet with the necessary subnets and configurations.

This step explains what is required for both scenarios and how Terraform can help you achieve it. The key here is understanding why these configurations are needed and tailoring them to your environment.

### General Requirements for VNet

Before proceeding, ensure your VNet meets the following requirements:

* The VNet must be in the same Azure region and subscription as the Databricks workspace.
* The address space for the VNet must use a CIDR block between /16 and /24.
* Two dedicated subnets are required within the VNet: a container (private) subnet and a host (public) subnet.
* The subnets must not be shared with other Azure resources or Databricks workspaces.

### Option A: Create a New VNet

For creating a new VNet for Databricks, Terraform makes it easy to provision all necessary resources. Here's what you'll create:

1. A new Virtual Network.
2. One container (private) subnet.
3. One host (public) subnet.
4. Azure network security groups.
5. Necessary associations between the subnets and the NSG.

network.tf

| # This file creates a new VNet for the Databricks workspace  # Don’t use this file if you already have a VNet you want to use  locals {  network\_prefix = var.workspace\_name  }  resource "azurerm\_virtual\_network" "this" {  name = "${local.network\_prefix}-vnet"  location = azurerm\_resource\_group.this.location  resource\_group\_name = azurerm\_resource\_group.this.name  address\_space = [var.cidr]  tags = var.tags  }  resource "azurerm\_network\_security\_group" "this" {  name = "${local.network\_prefix}-nsg"  location = azurerm\_resource\_group.this.location  resource\_group\_name = azurerm\_resource\_group.this.name  tags = var.tags  }  resource "azurerm\_subnet" "public" {  name = "${local.network\_prefix}-public-subnet"  resource\_group\_name = azurerm\_resource\_group.this.name  virtual\_network\_name = azurerm\_virtual\_network.this.name  address\_prefixes = [cidrsubnet(var.cidr, 2, 0)]  delegation {  name = "databricks"  service\_delegation {  name = "Microsoft.Databricks/workspaces"  actions = [  "Microsoft.Network/virtualNetworks/subnets/join/action",  "Microsoft.Network/virtualNetworks/subnets/prepareNetworkPolicies/action",  "Microsoft.Network/virtualNetworks/subnets/unprepareNetworkPolicies/action"]  }  }  }  resource "azurerm\_subnet\_network\_security\_group\_association" "public" {  subnet\_id = azurerm\_subnet.public.id  network\_security\_group\_id = azurerm\_network\_security\_group.this.id  }  resource "azurerm\_subnet" "private" {  name = "${local.network\_prefix}-private-subnet"  resource\_group\_name = azurerm\_resource\_group.this.name  virtual\_network\_name = azurerm\_virtual\_network.this.name  address\_prefixes = [cidrsubnet(var.cidr, 2, 1)]  delegation {  name = "databricks"  service\_delegation {  name = "Microsoft.Databricks/workspaces"  actions = [  "Microsoft.Network/virtualNetworks/subnets/join/action",  "Microsoft.Network/virtualNetworks/subnets/prepareNetworkPolicies/action",  "Microsoft.Network/virtualNetworks/subnets/unprepareNetworkPolicies/action"]  }  }  }  resource "azurerm\_subnet\_network\_security\_group\_association" "private" {  subnet\_id = azurerm\_subnet.private.id  network\_security\_group\_id = azurerm\_network\_security\_group.this.id  } |
| --- |

* var.workspace\_name: Contains the Databricks workspace name, used to prefix resource names.
* var.cidr: Specifies the main CIDR block for the virtual network and subnet address calculations.
* var.tags: Holds tags applied to resources for management and identification.
* local.network\_prefix: Stores the workspace name locally to construct resource names.
* azurerm\_resource\_group.this.location: Provides the Azure region where resources are deployed.
* azurerm\_resource\_group.this.name: Specifies the name of the Azure resource group for network resources.
* azurerm\_virtual\_network.this.name: Contains the virtual network's name for subnet associations.
* azurerm\_network\_security\_group.this.id: Provides the security group's ID for subnet security associations.
* azurerm\_subnet.public.id: Represents the public subnet's ID for the security association.
* azurerm\_subnet.private.id: Represents the private subnet's ID for the security association.
* cidrsubnet(var.cidr, 2, 0): Generates the CIDR for the public subnet from the main address space.
* cidrsubnet(var.cidr, 2, 1): Generates the CIDR for the private subnet from the main address space.

Documentation/Examples:

1. [Adb-vnet-injection example on GitHub](https://github.com/databricks/terraform-databricks-examples/blob/main/examples/adb-vnet-injection/vnet.tf)
2. [Provisioning Azure Databricks](https://registry.terraform.io/providers/databricks/databricks/latest/docs/guides/azure-workspace)
3. [Deploy Azure Databricks in your Azure virtual network (VNet injection)](https://learn.microsoft.com/en-us/azure/databricks/security/network/classic/vnet-inject)

### Option B: Use an Existing VNet

If you already have a VNet that satisfies the above requirements, you can use it for your Databricks workspace deployment.

network\_existing\_vnet.tf

| # This file uses an existing VNet for the Databricks workspace  # Don’t use this file if you are creating a new VNet  locals {  network\_prefix = var.workspace\_name  }  data "azurerm\_virtual\_network" "existing" {  name = var.vnet\_name  resource\_group\_name = var.vnet\_resource\_group\_name  }  data "azurerm\_resource\_group" "vnet\_resource\_group" {  name = var.vnet\_resource\_group\_name  }  resource "azurerm\_network\_security\_group" "this" {  name = "${local.network\_prefix}-nsg"  location = data.azurerm\_resource\_group.vnet\_resource\_group.location  resource\_group\_name = data.azurerm\_resource\_group.vnet\_resource\_group.name  tags = var.tags  }  resource "azurerm\_subnet" "public" {  name = "${local.network\_prefix}-public-subnet"  resource\_group\_name = data.azurerm\_resource\_group.vnet\_resource\_group.name  virtual\_network\_name = data.azurerm\_virtual\_network.existing.name  address\_prefixes = [cidrsubnet(var.cidr, 2, 0)]  delegation {  name = "databricks"  service\_delegation {  name = "Microsoft.Databricks/workspaces"  actions = [  "Microsoft.Network/virtualNetworks/subnets/join/action",  "Microsoft.Network/virtualNetworks/subnets/prepareNetworkPolicies/action",  "Microsoft.Network/virtualNetworks/subnets/unprepareNetworkPolicies/action"]  }  }  }  resource "azurerm\_subnet\_network\_security\_group\_association" "public" {  subnet\_id = azurerm\_subnet.public.id  network\_security\_group\_id = azurerm\_network\_security\_group.this.id  }  resource "azurerm\_subnet" "private" {  name = "${local.network\_prefix}-private-subnet"  resource\_group\_name = data.azurerm\_resource\_group.vnet\_resource\_group.name  virtual\_network\_name = data.azurerm\_virtual\_network.existing.name  address\_prefixes = [cidrsubnet(var.cidr, 2, 1)]  delegation {  name = "databricks"  service\_delegation {  name = "Microsoft.Databricks/workspaces"  actions = [  "Microsoft.Network/virtualNetworks/subnets/join/action",  "Microsoft.Network/virtualNetworks/subnets/prepareNetworkPolicies/action",  "Microsoft.Network/virtualNetworks/subnets/unprepareNetworkPolicies/action"]  }  }  }  resource "azurerm\_subnet\_network\_security\_group\_association" "private" {  subnet\_id = azurerm\_subnet.private.id  network\_security\_group\_id = azurerm\_network\_security\_group.this.id  } |
| --- |

## 5. Create Databricks Workspace resources:

We are now creating a resource for the Databricks workspace.

This Terraform resource block provisions an Azure Databricks workspace with a premium SKU inside a specified resource group and virtual network, while associating it with security groups, subnets, and storage.

Note the presence of a commented block related to Customer Managed Keys (CMK). Databricks encrypts objects at rest, and by providing a CMK, this encryption can be done with the provided key. Managed services encryption encrypts workspace data and Databricks objects, while the Managed disk encryption encrypts the local storage of the Azure VMs. This guide does not address the configuration of such CMK keys; it only exposes the parameters where they can be applied.

tldr: By default, your data is encrypted using keys managed by the cloud provider.

For more information about using CMK, please refer to the following documentation:

<https://learn.microsoft.com/en-us/azure/databricks/security/keys/customer-managed-keys>

databricks.tf

| resource "azurerm\_databricks\_workspace" "this" {  name = var.workspace\_name  resource\_group\_name = azurerm\_resource\_group.this.name  location = azurerm\_resource\_group.this.location  sku = "premium"  tags = var.tags  # customer managed key (CMK) configuration  # customer\_managed\_key\_enabled = true  # managed\_services\_cmk\_key\_vault\_id = var.managed\_services\_cmk\_key\_vault\_id  # managed\_services\_cmk\_key\_vault\_key\_id= var.managed\_services\_cmk\_key\_vault\_key\_id  # managed\_disk\_cmk\_key\_vault\_id = var.managed\_disk\_cmk\_key\_vault\_id  # managed\_disk\_cmk\_key\_vault\_key\_id = var.managed\_disk\_cmk\_key\_vault\_key\_id  custom\_parameters {  # if using new vnet  virtual\_network\_id = azurerm\_virtual\_network.this.id  # if using existing vnet  # virtual\_network\_id = data.azurerm\_virtual\_network.existing.id  private\_subnet\_name = azurerm\_subnet.private.name  public\_subnet\_name = azurerm\_subnet.public.name  public\_subnet\_network\_security\_group\_association\_id = azurerm\_subnet\_network\_security\_group\_association.public.id  private\_subnet\_network\_security\_group\_association\_id = azurerm\_subnet\_network\_security\_group\_association.private.id  storage\_account\_name = var.root\_storage\_name  }  depends\_on = [  azurerm\_subnet\_network\_security\_group\_association.public,  azurerm\_subnet\_network\_security\_group\_association.private  ]  } |
| --- |

* var.workspace\_name: Name of the Databricks workspace to be created.
* var.tags: Key-value pairs to organize and categorize the workspace with tags.
* var.managed\_services\_cmk\_key\_vault\_id: Azure Key Vault ID for customer-managed key (CMK) setup of managed services (optional).
* var.managed\_services\_cmk\_key\_vault\_key\_id: Key ID for CMK within the managed services Key Vault (optional).
* var.managed\_disk\_cmk\_key\_vault\_id: Azure Key Vault ID for customer-managed key for managed disks (optional).
* var.managed\_disk\_cmk\_key\_vault\_key\_id: Key ID for CMK within the managed disks Key Vault (optional).
* var.root\_storage\_name: Name of the root storage account associated with the workspace.

**Important**: Please check the comments for the virtual\_network\_id parameter in this block. Different parameters are used depending on whether a new VNet should be created or reuse an existing VNet; keep just the appropriate line in the code.

## 6. Declaring variables

In this step, we will create a file containing all the variables used in the aforementioned resources. All values are string types except for tags, which can be a map of string values to support multiple tagging.

variables.tf

| variable "azure\_subscription\_id" {  description = "Your Azure Subscription ID"  type = string  }  variable "resource\_group\_name" {  description = "The name of the resource group"  type = string  }  variable "location" {  description = "The Azure region to deploy the workspace to"  type = string  }  variable "workspace\_name" {  description = "The name of the Databricks workspace"  type = string  }  variable "root\_storage\_name" {  description = "The name of the root storage account"  type = string  }  variable "vnet\_name" {  description = "The name of the virtual network"  type = string  }  variable "cidr" {  description = "The CIDR block of the virtual network"  type = string  default = "10.0.0.0/20"  }  # Required when using an existing VNet  # variable "vnet\_resource\_group\_name" {  # description = "The name of the resource group where the existing VNet is located"  # type = string  # }  variable "tags" {  description = "A map of tags to assign to the resources"  type = map(string)  default = {}  }  # variable "managed\_services\_cmk\_key\_vault\_id" {  # description = "The Key Vault ID of the CMK for managed services encryption"  # type = string  # }  # variable "managed\_services\_cmk\_key\_vault\_key\_id" {  # description = "The CMK ID for managed services encryption"  # type = string  # }  # variable "managed\_disk\_cmk\_key\_vault\_id" {  # description = "The Key Vault ID of the CMK for managed disks encryption"  # type = string  # }  # variable "managed\_disk\_cmk\_key\_vault\_key\_id" {  # description = "The CMK ID for managed disks encryption"  # type = string  # } |
| --- |

## 7. Outputs

Optionally, create a file that contains the configurations to output the workspace ID and the URL to the workspace in the terminal once the workspace is successfully created.

outputs.tf

| output "databricks\_workspace\_id" {  value = azurerm\_databricks\_workspace.this.id  }  output "workspace\_url" {  value = "https://${azurerm\_databricks\_workspace.this.workspace\_url}/"  } |
| --- |

## 8. Assigning values to the variables

Lastly, we will assign values to the variables in this last file.

If you want Terraform to automatically load values for variables from a file, the file must be named either terraform.tfvars, terraform.tfvars.json, or end with .auto.tfvars or .auto.tfvars.json. If your file has a custom name (like random\_name.tfvars), you must provide it explicitly using the -var-file flag when running Terraform commands.

The following is an example of a terraform.tfvars file; replace the original values with your desired ones.

terraform.tfvars

| azure\_subscription\_id = "My\_subscription\_id"  resource\_group\_name = "My\_resource\_group\_name"  location = "westeurope"  workspace\_name = "my-prod"  root\_storage\_name = "myrootstorage"  vnet\_name = "my-vnet"  cidr = "10.0.0.0/20"  tags = {  Creator = "my@email.com"  Project = "Project A"  }  # if using existing VNet  # vnet\_resource\_group\_name = "existing-vnet-rg" |
| --- |

For the supported Azure Databricks locations, please visit:  
<https://learn.microsoft.com/en-us/azure/databricks/resources/supported-regions>

## 9. Save the files

Save all the files you just created before proceeding to the final step.

## 10. Execution

Lastly, in the terminal opened in the folder, execute the following commands:

1. terraform init
2. terraform plan
3. terraform apply

Occasionally, you'll be asked to confirm certain actions; type yes when prompted.

Once the execution finishes, the terminal will output the URL of the created workspace.

**Congratulations!** You now have a production-grade Databricks workspace that adheres to best practices with the potential for future extensibility if needed.

## Terraform template examples\* and more documentation:

\*Note: Keep in mind that the git code is not always up to date. You should use these templates as an example and not directly copy and paste. Please note that the code in the template projects is provided for your exploration only and is not formally supported by Databricks with Service Level Agreements (SLAs). They are provided AS-IS, and we do not make any guarantees of any kind.

* [Deploy with Private Link](https://github.com/databricks/terraform-databricks-examples/tree/main/examples/adb-with-private-link-standard)
* [Security Reference Architecture Template](https://github.com/databricks/terraform-databricks-sra)
  + This is a template that adheres to the best security practices we recommend.
* [Terraform Databricks provider documentation](https://registry.terraform.io/providers/databricks/databricks/latest/docs)
* [Configure a workspace with VNet injection](https://learn.microsoft.com/en-us/azure/databricks/security/network/classic/vnet-inject)