Quickstart: Use Terraform to create an Azure Recovery Services vault

Article • 02/05/2025

In this quickstart, you learn how to use Terraform to create an Azure resource group, an Azure Recovery Services vault, and a backup policy to share files in Azure. The Azure Site Recovery service can help your business applications to stay online during planned and unplanned outages. Specifically, Site Recovery uses replication, failover, and recovery to manage on-premises machines and Azure virtual machines during disaster recovery. These Site Recovery methods can contribute to your business continuity and disaster recovery strategy.

An Azure Recovery Services vault is a storage entity in Azure that houses data such as backups and recovery points that can protect and manage your data. You create the vault first and then the backup policy for sharing files, which specifies when and how often backups should occur, plus the retention period. This setup can help to ensure that your data is backed up consistently and can be easily restored if needed.

Terraform enables the definition, preview, and deployment of cloud infrastructure. Using Terraform, you create configuration files using HCL syntax. The HCL syntax allows you to specify the cloud provider - such as Azure - and the elements that make up your cloud infrastructure. After you create your configuration files, you create an *execution plan* that allows you to preview your infrastructure changes before they're deployed. Once you verify the changes, you apply the execution plan to deploy the infrastructure.

- ✓ Create an Azure resource group with a unique name.
- ✓ Define local variables for the SKU name and tier.
- ✓ Create an Azure Recovery Services vault in the resource group.
- ✓ Create a backup policy to share files in the resource group.
- ✓ Output the names of the Recovery Services Vault and the backup policy for sharing files.

Prerequisites

- Create an Azure account with an active subscription. You can create an account for free.
- Install and configure Terraform.

Implement the Terraform code

ⓘ **Note**

The sample code for this article is located in the Azure Terraform GitHub repo . You can view the log file containing the test results from current and previous versions of Terraform .

See more articles and sample code showing how to use Terraform to manage Azure resources.

1. Create a directory in which to test and run the sample Terraform code, and make it the current directory.

2. Create a file named `main.tf`, and insert the following code:

Terraform

```
# Create Resource Group
resource "random_pet" "rg_name" {
  prefix = var.resource_group_name_prefix
}

resource "azurerm_resource_group" "rg" {
  location = var.resource_group_location
  name     = random_pet.rg_name.id
}

locals {
  skuName = "RS0"
  skuTier = "Standard"
}

# Create Recovery Services Vault
resource "azurerm_recovery_services_vault" "vault" {
  name                = var.vaultName
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  sku                 = local.skuName
}

# Create Backup Policy for File Share
resource "azurerm_backup_policy_file_share" "policy" {
  name                = "vaultstorageconfig"
  resource_group_name = azurerm_resource_group.rg.name
  recovery_vault_name = azurerm_recovery_services_vault.vault.name

  backup {
    frequency = "Daily"
    time      = "23:00"
  }

  retention_daily {
    count = 10
  }
}
```

3. Create a file named `outputs.tf`, and insert the following code:

Terraform

```
output "recovery_services_vault_name" {
  value = azurerm_recovery_services_vault.vault.name
}

output "backup_policy_file_share_name" {
  value = azurerm_backup_policy_file_share.policy.name
}
```

4. Create a file named `providers.tf`, and insert the following code:

Terraform

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>3.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~>3.0"
    }
  }
}

provider "azurerm" {
  features {}
}
```

5. Create a file named `variables.tf`, and insert the following code:

Terraform

```
variable "resource_group_location" {
  type        = string
  default     = "eastus"
  description = "Location of the resource group."
}

variable "resource_group_name_prefix" {
  type        = string
  default     = "rg"
  description = "Prefix of the resource group name that's combined with a
random ID so name is unique in your Azure subscription."
}

variable "vaultName" {
  description = "Name of the Recovery Services Vault."
  type        = string
  default     = "examplevault"
}
```

Initialize Terraform

Run terraform init to initialize the Terraform deployment. This command downloads the Azure provider required to manage your Azure resources.

Console

```
terraform init -upgrade
```

Key points:

- The `-upgrade` parameter upgrades the necessary provider plugins to the newest version that complies with the configuration's version constraints.

Create a Terraform execution plan

Run terraform plan to create an execution plan.

Console

```
terraform plan -out main.tfplan
```

Key points:

- The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
- The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.

Apply a Terraform execution plan

Run terraform apply to apply the execution plan to your cloud infrastructure.

Console

```
terraform apply main.tfplan
```

Key points:

- The example `terraform apply` command assumes you previously ran `terraform plan -out main.tfplan`.
- If you specified a different filename for the `-out` parameter, use that same filename in the call to `terraform apply`.
- If you didn't use the `-out` parameter, call `terraform apply` without any parameters.

Verify the results

Azure CLI

1. Get the Azure resource group name.

   Console

   ```
   resource_group_name=$(terraform output -raw resource_group_name)
   ```

2. Get the Azure Recovery Services vault name.

   Console

   ```
   recovery_services_vault_name=$(terraform output -
   recovery_services_vault_name)
   ```

3. Get the Azure Recovery Services vault backup policy file share name.

   Console

   ```
   backup_policy_file_share_name=$(terraform output -
   backup_policy_file_share_name)
   ```

4. Run az backup vault show to view the Azure Recovery Services vault.

   Azure CLI

```
az backup vault show --name $recovery_services_vault_name --resource
group $resource_group_name
```

Clean up resources

When you no longer need the resources created via Terraform, do the following steps:

1. Run terraform plan and specify the `destroy` flag.

   Console

   ```
   terraform plan -destroy -out main.destroy.tfplan
   ```

   Key points:

   - The `terraform plan` command creates an execution plan, but doesn't execute it. Instead, it determines what actions are necessary to create the configuration specified in your configuration files. This pattern allows you to verify whether the execution plan matches your expectations before making any changes to actual resources.
   - The optional `-out` parameter allows you to specify an output file for the plan. Using the `-out` parameter ensures that the plan you reviewed is exactly what is applied.

2. Run terraform apply to apply the execution plan.

   Console

   ```
   terraform apply main.destroy.tfplan
   ```

Troubleshoot Terraform on Azure

Troubleshoot common problems when using Terraform on Azure.

Next steps

See more articles about Azure Recovery Services vault