Terraform Deployment failures

Last updated by | Mohammad Abu Hamdieh | Nov 16, 2022 at 10:02 AM PST

Terraform Deployment failures

Resolve Terraform deployment failures

Terraform can be used to manage and deploy your Azure Database for PostgreSQL, which can simplify managing a platform as a service (PaaS).

When deploying Azure databases for PostgreSQL using Terraform, a race condition can be triggered if resources and configurations are not properly deployed. Internal server errors and conflicts may result.

Use the following guidance to help you resolve your issues with Terraform deployment failures.

Troubleshoot deployment failures

Configuration

- When creating a server using Terraform, make sure `depends-on` is used with configuration, firewall, or creating databases, for instance:

  Flexible server: If you're using a default template from a Terraform flexible server, you'll need to add `depends_on` for virtual networks and private DNS zones if the server is created with private access.

```
resource "azurerm_postgresql_flexible_server" "example" {
      name                   = "example-psqlflexibleserver"
      resource_group_name    = azurerm_resource_group.example.name
      location               = azurerm_resource_group.example.location
      version                = "12"
      delegated_subnet_id    = azurerm_subnet.example.id
      private_dns_zone_id    = azurerm_private_dns_zone.example.id
      administrator_login    = "psqladmin"
      administrator_password = "H@Sh1CoR3!"
      zone                   = "1"

      storage_mb = 32768

      sku_name   = "GP_Standard_D4s_v3"
      depends_on = [azurerm_private_dns_zone_virtual_network_link.example]

    }
```

Single server: For a single server, you'll need to add `depends_on` for the private endpoint.

```
resource "azurerm_private_endpoint" "example" {
    name                = "example-endpoint"
    location            = azurerm_resource_group.example.location
    resource_group_name = azurerm_resource_group.example.name
    subnet_id           = azurerm_subnet.endpoint.id

    private_service_connection {
    name                           = "example-privateserviceconnection"
    private_connection_resource_id = azurerm_private_link_service.example.id
    is_manual_connection           = false


    depends_on = [azurerm_postgresql_server.example.name]
  }
```

- If you're doing multiple configurations, for instance, updating a firewall and changing parameters, make sure to add `depends_on` when changing the parameters for the firewall update if the firewall was the first change on the server.

```
resource "azurerm_postgresql_flexible_server_configuration" "example" {
    name      = "backslash_quote"
    server_id = azurerm_postgresql_flexible_server.example.id
    value     = "on"

    depends_on = [azurerm_postgresql_flexible_server_firewall_rule.example]
    }
```

- The parallelism default in Terraform is 10. Try to run Terraform without it: `terraform apply parallelism =1` .
- Avoid relying on implicit dependencies, and utilize explicit dependencies. See the examples under number 1 of this section.
- When configuring high availability, use `ignore-changes` in Terraform to avoid running a failover:

```
resource "azurerm_postgresql_flexible_server" "example" {
  name                   = "server_name"
  resource_group_name    = "terraform"
  location               = "West Europe"
  version                = "12"
  administrator_login    = "adminuser"
  administrator_password = "H@Sh1CoR33"
  sku_name               = "GP_Standard_D2ds_v4"

  storage_mb = 32768

lifecycle {
      ignore_changes = [
        zone,
        high_availability.0.standby_availability_zone
      ]
  }
}
```

Additional troubleshooting

If you're still seeing errors in Terraform deployments after using `depends_on` and disabling parallelism, try the following.

1. Create a server as follows:

```
resource "azurerm_postgresql_flexible_server" "example" {
  name                   = "testflexible"
  resource_group_name    = "testgroup"
  location               = azurerm_resource_group.example.location
  version                = "12"
  administrator_login    = "psqladmin"
  administrator_password = "H@Sh1CoR3!"

  storage_mb = 32768

  sku_name = "GP_Standard_D4s_v3"
}
```

- After the server is created, then add firewall rules and server configuration.

2. You can debug Terraform.

   In PowerShell run:

   ```
   $env:TF_LOG="TRACE"
   In bash/linux shell
   export TF_LOG="TRACE"
   ```