

# Get started

## Build your first Elestio Terraform project

This exemple shows the setup for a Terraform project containing a single PostgreSQL service, and shows off some useful commands to stand up (and destroy) your Elestio infrastructure.

You can check this repository [elestio-terraform-scratch](#) that contain the final code of this guide.

### Prepare the dependencies

- [Sign up for Elestio if you haven't already](#)
- [Get your API token in the security settings page of your account](#)
- [Download and install Terraform](#)

You need a Terraform CLI version equal or higher than v0.14.0.

To ensure you're using the acceptable version of Terraform you may run the following command:

```
terraform -v
```

Your output should resemble:

```
Terraform v0.14.0 # any version >= v0.14.0 is OK
...
```

### Configure your project and services

Terraform files are used to define the structure and configuration of your infrastructure. It is generally a good idea to keep these definitions in **separate files** rather than combining them all in one file.

This section will explain how to organize a basic Terraform project :

## 1. Create and move to an empty folder

Here is an overview of the files we will create together :

```
| - outputs.tf  # Defines the outputs you want terraform extract
| - postgres.tf  # Defines the PostgreSQL service
| - project.tf  # Defines the Elestio project that will contain the PostgreSQL service
| - provider.tf  # Defines the Elestio provider for Terraform
| - secret.tfvars  # Defines the sensitive variables values
| - variables.tf  # Defines the variables required in other .tf files
```

## 2. Create a file `provider.tf` and declare the provider adding the following lines :

```
# provider.tf

terraform {
  required_providers {
    elestio = {
      source = "elestio/elestio"
      version = "0.3.0" # check out the latest version available
    }
  }
}

# Configure the Elestio Provider
provider "elestio" {
  email    = var.elestio_email
  api_token = var.elestio_api_token
}
```

As you can see, the email and API token are assigned to variables.  
You should never put sensitive information directly in `.tf` files.

## 3. Create a file `variables.tf` and declare variables adding the following lines :

```
# variables.tf

variable "elestio_email" {
  description = "Elestio Email"
```

```
type      = string
}

variable "elestio_api_token" {
  description = "Elestio API Token"
  type        = string
  sensitive   = true
}
```

This file does not contain the values of these variables. We will have to declare them in another file.

4. **Create a file `secret.tfvars` and fill it with your values :**

```
# secret.tfvars

elestio_email    = "YOUR-EMAIL"
elestio_api_token = "YOUR-API-TOKEN"
```

**Do not commit with Git this file !** Sensitive information such as an API token should never be pushed.

For more information on how to securely authenticate, please read the [authentication documentation](#).

5. **Create a file `project.tf` and add the following lines :**

```
# project.tf

# Create a Project
resource "elestio_project" "pg_project" {
  name          = "PostgreSQL Project"
  description    = "Contains a postgres database"
  technical_emails = var.elestio_email
}
```

To contain our PostgreSQL service, we will have to create a new project on Elestio. Instead of using the web interface, we can also declare it via terraform.

## 6. Create a file `postgres.tf` and add the following lines :

```
# postgres.tf

# Create a PostgreSQL Service
resource "elestio_postgresql" "pg_service" {
  project_id   = elestio_project.pg_project.id
  server_name  = "pg-service"
  server_type  = "SMALL-1C-2G"
  provider_name = "hetzner"
  datacenter   = "fsn1"
  support_level = "level1"
  admin_email  = var.elestio_email
}
```

Terraform takes care of managing the dependencies and creating the different resources in the right order. As you can see, `project_id` will be filled with the value of the Project Resource that will be created with the previously `project.tf` file.

## 7. Create a file `outputs.tf` and add the following lines :

```
# outputs.tf

output "pg_service_psql_command" {
  value       = elestio_postgresql.pg_service.database_admin.command
  description = "The PSQL command to connect to the database."
  sensitive   = true
}
```

# Apply the Terraform configuration

## 1. Download and install the Elestio provider defined in the configuration :

```
terraform init
```

## 2. Ensure the configuration is syntactically valid and internally consistent:

```
terraform validate
```

### 3. Apply the configuration :

```
terraform apply -var-file="secret.tfvars"
```

Deployment time varies by service, provider, datacenter and server type.

### 4. Voila, you have created a Project and PostgreSQL Service using Terraform !

You can visit the [Elestio web dashboard](#) to see these ressources.

## (Optional) Access to the database

Let's try to connect to the database to see if everything worked well

First, you need to [install psql](#).

After that, run this command :

```
eval "$(terraform output -raw pg_service_psql_command)"
```

**Note:** The command to leave psql terminal is `\q`

## Clean up

Run the following command to destroy all the resources you created:

```
terraform destroy -var-file="secret.tfvars"
```

This command destroys all the resources specified in your Terraform state. `terraform destroy` doesn't destroy resources running elsewhere that aren't managed by the current Terraform project.

Now you've created and destroyed an entire Elestio deployment!

Visit the [Elestio Dashboard](#) to verify the resources have been destroyed to avoid unexpected charges.

---

Revision #16

Created 30 December 2022 13:47:34 by Adam

Updated 3 January 2023 00:35:11 by Adam