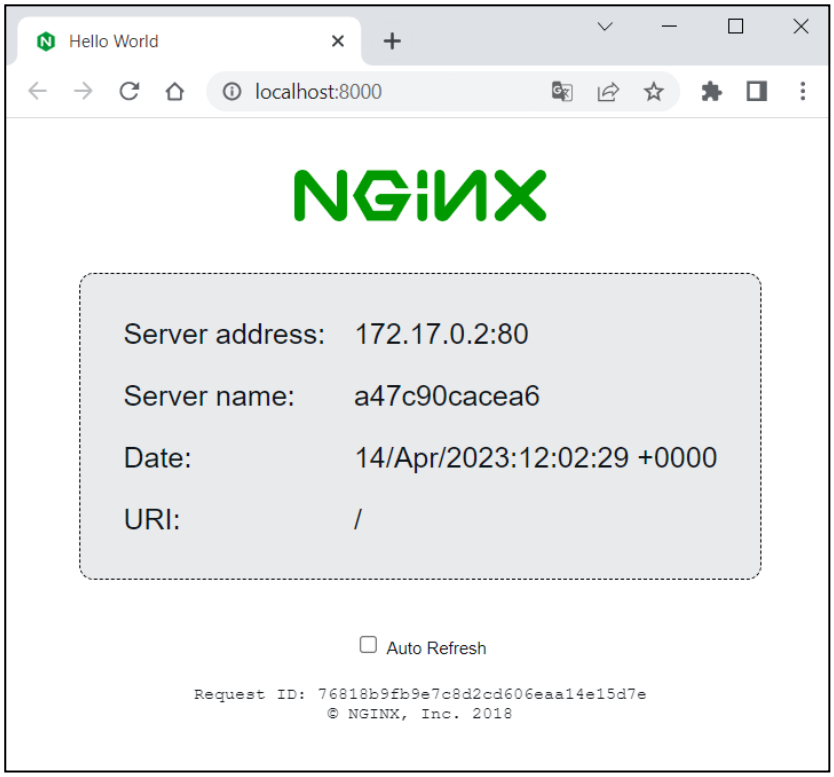
Lab: Infrastructure as Code

Lab assignment for the "Containers and Clouds" course @ SoftUni.

1. NGINX Docker Container

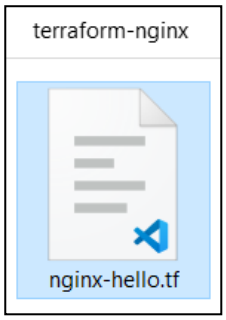
It is time to **create our first infrastructure**. We will provision an **NGINX Docker container** using **Terraform**.



For this, you should have **Docker running**.

Step 1: Write the Configuration

First, **create a directory** that will keep your **Terraform configuration files**. Do it in a way you like – using **File Explorer** or a **terminal**. Then, **create a file** with the **.tf extension** in the **folder**, where we will **define the infrastructure** that we want:



Open the file in an editor of your choice and **let's write the configuration** for the **NGINX container**.

Start by adding a **terraform {} block**, which **contains Terraform settings**, including the **required providers** Terraform will use to **provision your infrastructure**.

```
nginx-hello.tf ●
D: > SoftUni > terraform-nginx > nginx-hello.tf
1   terraform {
2
3   }
```

Each **`Terraform` module** must **declare which providers it requires**, so that **`Terraform` can install and use them**. **Provider requirements** are declared in a **`required_providers {}`** block in the **`terraform`** one and consist of a **local name**, a **source location**, and a **version constraint**.

In our case, we need **`Docker`**, so we should **install a provider for it**. **Providers are installed** from the Terraform Registry by default and the one we need is called "**`kreuzwerker/docker`**". (https://registry.terraform.io/providers/kreuzwerker/docker/latest). Use it like this:

```
nginx-hello.tf ×
D: > SoftUni > terraform-nginx > nginx-hello.tf
1   terraform {
2     required_providers {
3       docker = {
4         source  = "kreuzwerker/docker"
5         version = "~> 3.0.1"
6       }
7     }
8   }
9
```

Provider's `local name` is its **unique identifier within this module** (in our case "**`docker`**"), the **`source`** defines the **global source address for the provider** you intend to use and the **`version` constraint specifies which subset of available provider versions the module is compatible with**.

Next, the **`provider {}` block configures the specified provider**, in this case "**`docker`**". **Configure Docker** to **connect to Docker daemon** and **interact with Docker containers and images** on a **`Windows` host** using a **named pipe**: "**`npipe:////./pipe/docker_engine`**":

```
10  provider "docker" {
11    host = "npipe:////.//pipe//docker_engine"
12  }
13
```

Now we need to **pull the** "**`nginxdemos/hello`**" **image** from **`Docker` Hub** to later create a container with it. To do this, you need to **create a `Terraform` resource**.

Resource blocks are used to **define components of your infrastructure**. They have **two strings before the block**: the **resource type** and the **resource name**. Together, the resource type and resource name form a **unique ID for the resource**.

Use the "**`docker_image`**" **`Terraform` resource** and "**`nginx`**" **for a resource name**. Also, **provide the name of the `Docker` Hub image** you want to use:

```
14  resource "docker_image" "nginx" {
15    name = "nginxdemos/hello"
16  }
17
```

At last, **create a resource for the Docker container**, using the **image we defined as a resource** (with its **unique resource ID** as **image name**). Also, you should give a **name to the container** and **map ports**:

```
resource "docker_container" "nginx" {
  image = resource.docker_image.nginx.name
  name  = "nginx_hello"

  ports {
    internal = 80
    external = 8000
  }
}
```

As you can see, the **resource blocks can also contain arguments** which you use to **configure the resource**.

Now your **`Terraform` configuration is ready**. **Save the file** and see how to use it in the next steps.

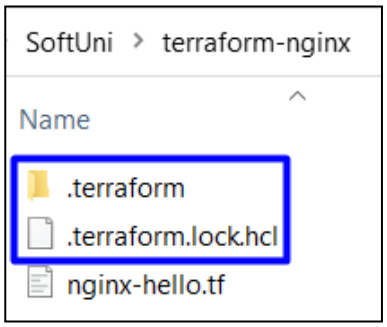
Step 2: Initialize the Directory

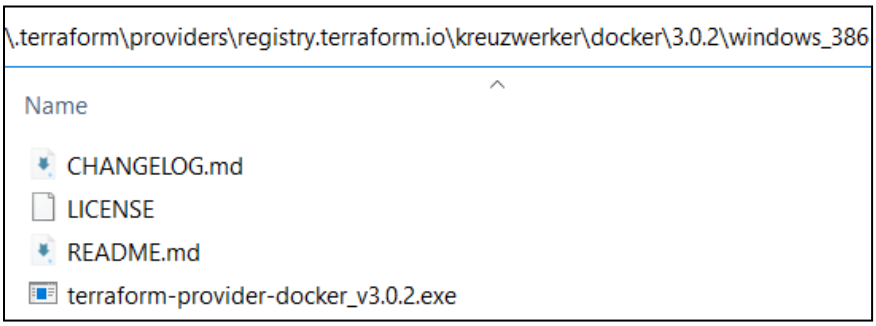
When you have a **configuration**, you need to **initialize a configuration directory**, which **downloads and installs the providers** defined in the configuration. Do it with the **`terraform init` command** in the **`Terraform` configuration directory**:

```
PS D:\SoftUni\terraform-nginx> terraform init

Initializing the backend...

Initializing provider plugins...
- Finding kreuzwerker/docker versions matching "~> 3.0.1"...
- Installing kreuzwerker/docker v3.0.2...
- Installed kreuzwerker/docker v3.0.2 (self-signed, key ID BD080C4571C6104C)

Partner and community providers are signed by their developers.
If you'd like to know more about provider signing, you can read about it here:
https://www.terraform.io/docs/cli/plugins/signing.html

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
```

`Terraform` downloads the `docker` provider and **installs it in a hidden subdirectory** of your current working directory, named **`.terraform`**. It also **creates a lock file** named **`.terraform.lock.hcl`** which **specifies the exact provider versions used**, so that you can control when you want to update the providers used for your project:





Then, before we **create the container**, let's **format and validate the configuration** we have written.

Step 3: Format and Validate the Configuration

The **`terraform fmt` command automatically updates configurations** in the current directory for readability and consistency:

```
PS D:\SoftUni\terraform-nginx> terraform fmt
```

`Terraform` will print out the names of the files it modified, if any. In our case, our configuration was formatted well.

You can also make sure your **configuration is syntactically valid and internally consistent** by using the **`terraform validate`** command:

```
PS D:\SoftUni\terraform-nginx> terraform validate
Success! The configuration is valid.
```

Validation was successful and our **configuration is valid**. Now it is time to apply it.

Step 4: Create Infrastructure

Use the **`terraform apply`** command to **execute the configuration file and apply it** to **create the wanted infrastructure**. Terraform will **print a similar output**:

```
PS D:\SoftUni\terraform-nginx> terraform apply

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # docker_container.nginx will be created
  + resource "docker_container" "nginx" {
      + attach                                      = false
      ...
Plan: 2 to add, 0 to change, 0 to destroy.
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value:
```

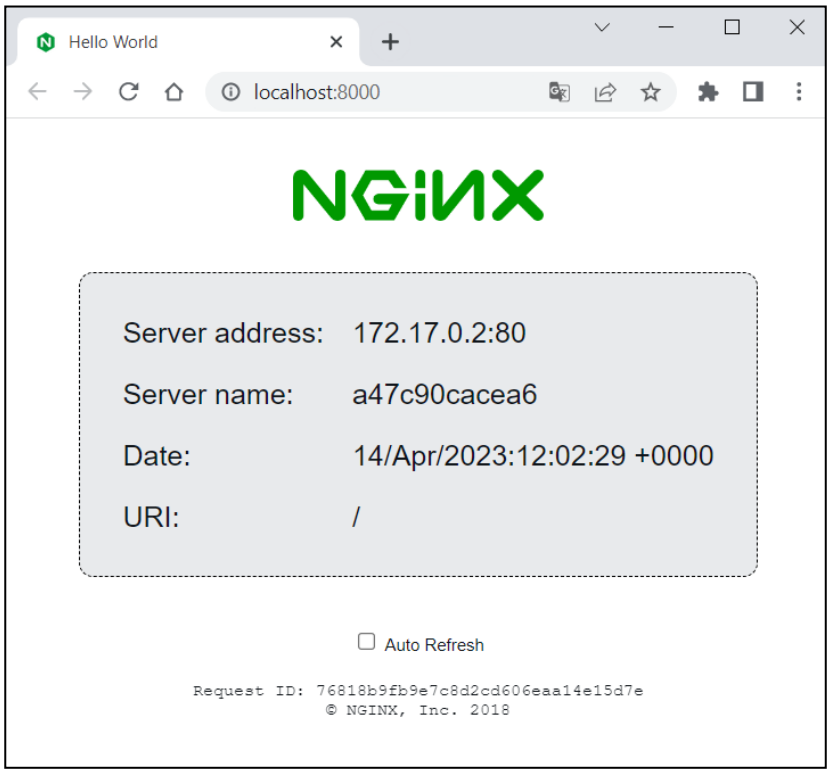
Before it applies any changes, **`Terraform` prints out the execution plan** which **describes the actions Terraform will take** in order to **change your infrastructure to match the configuration**. Terraform will now pause and **wait for your approval** before proceeding.

In this case the **plan is acceptable**, so **type** "**yes**" at the **confirmation prompt to proceed**:

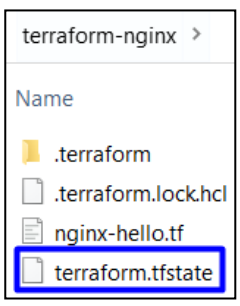
```
  Enter a value: yes

docker_image.nginx: Creating...
docker_image.nginx: Creation complete after 0s [id=sha256:f1f55236c9e2897e3cb18a07cf
0cd5d5f3d54aaecfbfabdd081aa73f95bb9090nginxdemos/hello]
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 1s [id=a47c90cacea67dae3d7f058c3207c
4acb4f46ab321b147b6c1442080e8aed959]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Terraform has created the infrastructure you want, which includes **2 resources** – the **container image** and the **container** itself. You can **look at them** in **Docker Desktop**:

You can also **visit** http://localhost:8080/ to **validate that the NGINX container is started and working**:



You can also **inspect the state of the resources that Terraform manages**, as it wrote data into the **terraform.tfstate state file** when you **applied your configuration**:



Inspect the current state using the **`terraform show` command**:

```
PS D:\SoftUni\terraform-nginx> terraform show
# docker_container.nginx:
resource "docker_container" "nginx" {
    attach                                      = false
    command                                     = [
        "nginx",
        "-g",
        "daemon off;",
    ]
    container_read_refresh_timeout_milliseconds = 15000
    ...
# docker_image.nginx:
resource "docker_image" "nginx" {
    id          = "sha256:f1f55236c9e2897e3cb18a07cf0cd5d5f3d54aaecfbfabdd081aa73f95
bb9090nginxdemos/hello"
    image_id    = "sha256:f1f55236c9e2897e3cb18a07cf0cd5d5f3d54aaecfbfabdd081aa73f95
    ...
```

You have your **`Terraform` configuration and resources**. Now let's see how to **modify them**.

Step 5: Update Configuration

To **change your configuration**, make changes to the **`.tf` configuration file**. For the example, let's **change the external port** of the container **from `8000` to `5000`**:

nginx-hello.tf

D: > SoftUni > terraform-nginx > nginx-hello.tf

```
1  > terraform {···
8    }
9
10 > provider "docker" {···
12   }
13
14 > resource "docker_image" "nginx" {···
16   }
17
18   resource "docker_container" "nginx" {
19     image = resource.docker_image.nginx.name
20     name  = "nginx_hello"
21
22     ports {
23       internal = 80
24       external = 5000
25     }
26   }
```

Save the file and **apply the changed configuration**:

```
PS D:\SoftUni\terraform-nginx> terraform apply
docker_image.nginx: Refreshing state... [id=sha256:f1f55236c9e2897e3cb18a07cf0cd5d5f
3d54aaecfbfabdd081aa73f95bb9090nginxdemos/hello]
docker_container.nginx: Refreshing state... [id=a47c90cacea67dae3d7f058c3207c4acb4f4
6ab321b147b6c1442080e8aed959]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # docker_container.nginx must be replaced
-/+ resource "docker_container" "nginx" {
      + bridge                                    = (known after apply)
      ~ command                                   = [
          - "nginx",
          - "-g",
          - "daemon off;",
      ...
      ~ ports {
          ~ external = 8000 -> 5000 # forces replacement
            # (3 unchanged attributes hidden)
        }
    }

Plan: 1 to add, 0 to change, 1 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

docker_container.nginx: Destroying... [id=a47c90cacea67dae3d7f058c3207c4acb4f46ab321
b147b6c1442080e8aed959]
docker_container.nginx: Destruction complete after 1s
docker_container.nginx: Creating...
docker_container.nginx: Creation complete after 1s [id=ca9c6be7617b9238757537970b1e7
11e489e7e10548814077f39e1327556ca38]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

The **Docker provider** knows that it **cannot change the port of a container** after it has been created, so **`Terraform` destroys the old container** and **creates a new one**.

And you **have your new container**, accessible on http://localhost:5000/:

As you saw, you can **change the `Terraform` configuration and resources** easily in a way you want. However, **don't forget to examine `Terraform` execution plans carefully before applying them**, so that you don't perform incorrect or dangerous actions.

Step 6: Destroy Configuration

Once you **no longer need infrastructure**, you may want to **destroy it**. Use the **`terraform destroy` command** to **terminate resources** managed by your **Terraform project**.

Destroy the resources you created:

```
PS D:\SoftUni\terraform-nginx> terraform destroy
docker_image.nginx: Refreshing state... [id=sha256:f1f55236c9e2897e3cb18a07cf0cd5d5f
3d54aaecfbfabdd081aa73f95bb9090nginxdemos/hello]
docker_container.nginx: Refreshing state... [id=ca9c6be7617b9238757537970b1e711e489e
7e10548814077f39e1327556ca38]

...
Plan: 0 to add, 0 to change, 2 to destroy.

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes

docker_container.nginx: Destroying... [id=ca9c6be7617b9238757537970b1e711e489e7e1054
8814077f39e1327556ca38]
...
Destroy complete! Resources: 2 destroyed.
```

`Terraform` destroys resources in a **suitable order** to **respect dependencies**. You can **verify that the NGINX image and container don't exist anymore** in **`Docker Desktop`**.