

Infraestructura para TI parcial 2do corte: despliegue automático de una infraestructura en la nube

Universidad Tecnológica de Bolívar

Andrés Felipe Pico Zúñiga Carlos Alberto Salcedo Rodríguez
Andrés Juan Ramírez Torres

2023-10-11

Índice

1	Introducción	2
1.1	Objetivos	2
1.2	Herramientas a usar	2
1.3	Consideraciones importantes	2
2	Preparación	3
2.1	Estructura general	3
2.2	Funciones específicas	3
3	Configuración	4
3.1	Backend	4
3.1.1	Base de datos	4
3.1.2	Aplicación web	6
3.2	Ansible	9
3.2.1	Estructura general	9
3.2.2	Funciones específicas	9
3.3	Terraform	10
3.3.1	Funciones específicas	10
3.4	Azure CLI	11
4	Despliegue	12
4.1	<code>terraform init</code>	12
4.2	<code>terraform validate</code>	12
4.3	<code>terraform apply</code>	13
4.4	Resultados	14
5	Pruebas	15
5.1	API Web	15
5.1.1	Funcionamiento general	15
5.1.2	Conexión con la base de datos	15
5.2	Máquina virtual con la base de datos	18
5.2.1	SSH	18
5.2.2	Docker	19
5.2.3	Puertos	19
5.2.4	MySQL	20
5.2.5	Firewall	20

1 Introducción

El presente documento explica el proceso de preparación, desarrollo y despliegue de una infraestructura en la nube. El proyecto consiste en una API Web en Node.js que interactúa con una base de datos MySQL. El fin principal es tener un repositorio con scripts que permitan realizar este proceso de manera automática, cumpliendo así el paradigma de infraestructura como código (IaC).

1.1 Objetivos

En base a lo anterior, se deduce que los principales objetivos son:

- Desplegar la API en un App Service.
- Desplegar la base de datos en una máquina virtual.
- Automatizar este proceso.

1.2 Herramientas a usar

Primeramente se necesita un lugar en el que poder realizar el despliegue de la infraestructura, para esto se usará la nube pública de Azure. Asimismo, también se aplicarán otras herramientas más que permitan la completa automatización de este proceso:

- Terraform: para automatizar el despliegue de los recursos (máquina virtual y App Service).
- Azure CLI: para poder acceder a Azure desde Terraform.
- Ansible: para automatizar la configuración y ejecución de la base de datos en la máquina virtual.
- Docker: para ejecutar un contenedor en el que estará la base de datos.

1.3 Consideraciones importantes

Además de lo ya explicado, se tendrán en cuenta ciertos requerimientos:

- Por seguridad, solo se expondrán los puertos estrictamente necesarios para el funcionamiento de la infraestructura.
- El App Service debe estar configurado para usar HTTPS.
- Únicamente se debe tener acceso a la base de datos a través de la App Service, bloqueando cualquier otro tipo de conexión.
- El acceso por SSH a la máquina virtual solo lo tendrá la máquina controladora que ejecuta Terraform y Ansible, bloqueando las demás conexiones.

2 Preparación

En primera instancia, cabe aclarar que para una mejor organización se trabajará todo en un repositorio centralizado de git, de modo que se pueda ver claramente el paso a paso del proceso general.

2.1 Estructura general

El proyecto esta estructurado según la siguiente jerarquía de directorios:

```
## ../
## +-- docs
## \-- iac
##     +-- ansible
##     +-- backend
##     |   +-- database
##     |   \-- nodejs-express-mysql
##     \-- terraform
```

2.2 Funciones específicas

La función de cada directorio está definida tal que:

- **docs:** documentación del proyecto (incluyendo este mismo documento).
- **iac:** infraestructura en general, tanto aplicación y base de datos como scripts para automatizar el despliegue.
 - **backend:** archivos de la base de datos y la aplicación.
 - * **database:** archivos para generar y correr imagen de la BD.
 - * **node-express-mysql:** archivos de la aplicación Web.
 - **ansible:** configuración de Ansible.
 - **terraform:** configuración de Terraform.

3 Configuración

Una vez explicada la base general del proyecto, se explicará cada módulo de éste dentro de la carpeta `iac` punto por punto.

3.1 Backend

Dentro de la carpeta `backend` encontramos la configuración de la base de datos y la aplicación Web.

3.1.1 Base de datos

Toda la configuración de ésta se encuentra dentro del directorio `database`, aquí se creará en primer lugar un script de SQL con la tabla que usará la aplicación y varios datos de prueba, este archivo se llamará `prueba.sql`:

```
CREATE TABLE IF NOT EXISTS `tutorials` (  
  id int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,  
  title varchar(255) NOT NULL,  
  description varchar(255),  
  published BOOLEAN DEFAULT false  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
INSERT INTO `tutorials` (id, title, description, published) VALUES  
  (1, "Node.js Basics", 'Tut#1 Description', false),  
  (2, "Rest APIs", 'Tut#2 Description', false),  
  (3, "Node Rest APIs", 'Tut#3 Description', false),  
  (4, "MySQL database", 'Tut#4 Description', false),  
  (5, "Node Rest Apis with MySQL", 'Tut#5 Description', false);
```

El siguiente paso consiste en crear una imagen de Docker que contenga el servidor de MySQL y ejecute dicho script. Para esto se creará el siguiente `Dockerfile`:

```
FROM mysql  
EXPOSE 3306  
COPY ./test.sql /docker-entrypoint-initdb.d/
```

Ahora bien, para simplificar y posteriormente automatizar las labores tanto de construcción de la imagen como de iniciación del contenedor, se usará un archivo `docker-compose.yml`:

```

services:
  database:
    container_name: 'database'
    build: .
    environment:
      MYSQL_ROOT_PASSWORD: '123456'
      MYSQL_DATABASE: 'appdb'
    restart: always
    network_mode: host
    volumes:
      - 'database:/var/lib/mysql'

volumes:
  database:

```

Por lo que, para construir la imagen inicialmente y ejecutar el contenedor se usa `docker compose up -d`.

```

usuario@reimu: database ♪ master
$ docker compose up -d
[+] Building 2.0s (8/8) FINISHED
=> [database internal] load build definition from Dockerfile
=> => transferring dockerfile: 106B
=> [database internal] load .dockerignore
=> => transferring context: 2B
=> [database internal] load metadata for docker.io/library/mysql:latest
=> [database auth] library/mysql:pull token for registry-1.docker.io
=> CACHED [database 1/2] FROM docker.io/library/mysql@sha256:44056c45e214c26c37b6f244534c6fb5f8a40eacbc28e870a2652b19d7a8a814
=> [database internal] load build context
=> => transferring context: 755B
=> [database 2/2] COPY ./test.sql /docker-entrypoint-initdb.d/
=> [database] exporting to image
=> => exporting layers
=> => writing image sha256:62740862b4c57ecd67e9c6a0c49f98317ee55db4acb76e52ba31e4b7d9489f65
=> => naming to docker.io/library/database-database
[+] Running 3/3
✔ Network database_default Created
✔ Volume "database_database" Created
✔ Container database Started

```

Y se verifica que efectivamente el contenedor se ejecutó con los cambios indicados:

```

usuario@reimu: database ♪ master
$ mysql -u root -h 127.0.0.1 -p123456 -e 'select * from tutorials;' appdb
mysql: [Warning] Using a password on the command line interface can be insecure.
+----+-----+-----+-----+
| id | title                | description          | published |
+----+-----+-----+-----+
| 1  | Node.js Basics      | Tut#1 Description   | 0        |
| 2  | Rest APIs            | Tut#2 Description   | 0        |
| 3  | Node Rest APIs       | Tut#3 Description   | 0        |
| 4  | MySQL database       | Tut#4 Description   | 0        |
| 5  | Node Rest Apis with MySQL | Tut#5 Description   | 0        |
+----+-----+-----+-----+

```

De este modo ya se tendría todo lo necesario para ejecutar el servidor de MySQL en la máquina virtual.

3.1.2 Aplicación web

Siguiendo con la API web, el primer paso es clonar el [repositorio](#) que contiene el código que usaremos para la aplicación:

```
usuario@reimu: backend ♀ master
$ git clone https://github.com/bezkoder/nodejs-express-mysql --quiet && ls nodejs-express-mysql
app  package.json  README.md  server.js
```

Todo este código se encuentra ahora en la carpeta `nodejs-express-mysql`, se ingresa en ella y lo siguiente es instalar las dependencias necesarias para correr la aplicación. No obstante, antes de esto es necesario cambiar una de las dependencias dentro del `package.json`, de modo que se usará el paquete `mysql2` en vez de `mysql`, ya que el primero está más actualizado y presenta mayor seguridad. Una vez hecho esto se hace la instalación con `npm install`:

```
usuario@reimu: nodejs-express-mysql ♀ master
$ npm install

added 72 packages, and audited 73 packages in 2s

8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Hecho esto, es necesario seguir realizando ciertos cambios al código de la aplicación, ahora se cambian los valores de la configuración de conexión al servidor de MySQL para que concuerde con los del contenedor, de modo que el archivo `./app/config/db.config.js` pasaría a ser:

```
module.exports = {
  HOST: "20.172.199.171",
  USER: "root",
  PASSWORD: "123456",
  DB: "appdb"
};
```

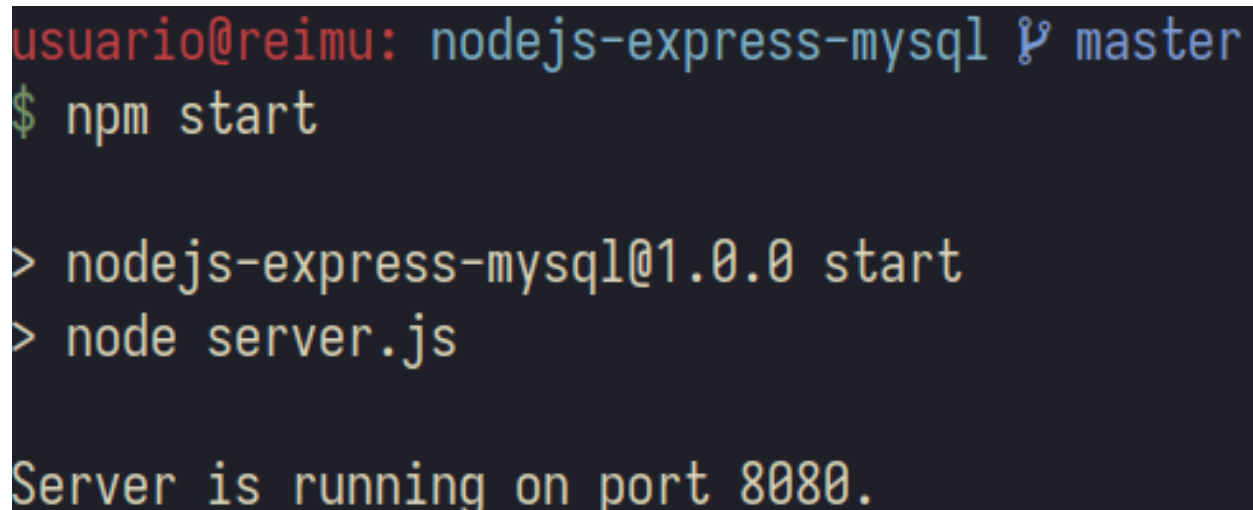
Y se actualiza igualmente el módulo que se conecta a la base de datos en `./app/models/db.js` para usar `mysql2`:

```
const mysql = require("mysql2");
const dbConfig = require("../config/db.config.js");

var connection = mysql.createPool({
  host: dbConfig.HOST,
  user: dbConfig.USER,
  password: dbConfig.PASSWORD,
  database: dbConfig.DB
});

module.exports = connection;
```

Una vez hechos estos cambios, se crea un script **run** para poner en marcha la aplicación, lo que nos permitirá ejecutarla tal que:

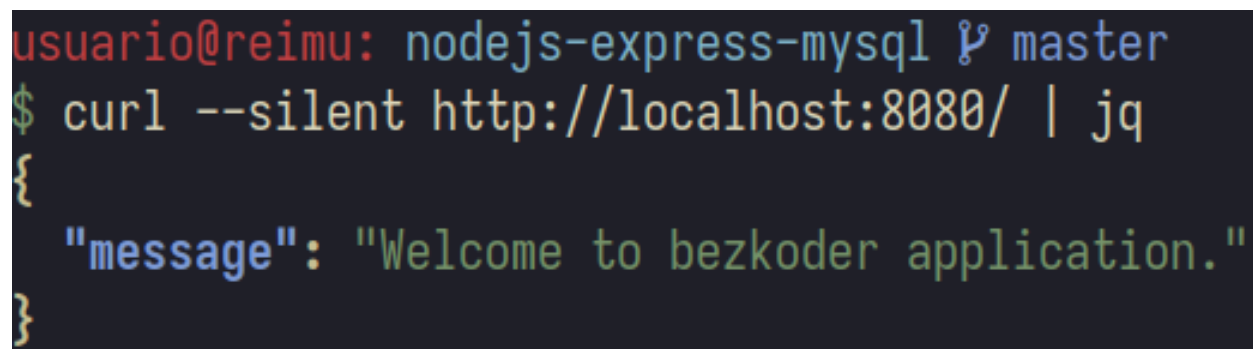


```
usuario@reimu: nodejs-express-mysql & master
$ npm start

> nodejs-express-mysql@1.0.0 start
> node server.js

Server is running on port 8080.
```

Lo siguiente es verificar que la API está corriendo correctamente:



```
usuario@reimu: nodejs-express-mysql & master
$ curl --silent http://localhost:8080/ | jq
{
  "message": "Welcome to bezkoder application."
}
```

Igualmente, se verifica que la API web sea capaz de acceder al servidor de la BD y realizarle peticiones para extraer datos:

```
usuario@reimu: nodejs-express-mysql ʘ master
$ curl -s http://localhost:8080/api/tutorials | jq
[
  {
    "id": 1,
    "title": "Node.js Basics",
    "description": "Tut#1 Description",
    "published": 0
  },
  {
    "id": 2,
    "title": "Rest APIs",
    "description": "Tut#2 Description",
    "published": 0
  },
  {
    "id": 3,
    "title": "Node Rest APIs",
    "description": "Tut#3 Description",
    "published": 0
  },
  {
    "id": 4,
    "title": "MySQL database",
    "description": "Tut#4 Description",
    "published": 0
  },
  {
    "id": 5,
    "title": "Node Rest Apis with MySQL",
    "description": "Tut#5 Description",
    "published": 0
  }
]
```


3.2 Ansible

Para usar Ansible dentro de este proyecto se debió armar primeramente un Ansible Playbook, que consiste en la sucesión de instrucciones que Ansible ejecutará en las máquinas objetivo, en este caso particular tratándose únicamente de la máquina virtual que ejecutará la base de datos. Para esto se emplean varios roles, que tienen como ventaja crear un entorno modular, reaprovechable y fácilmente expandible.

3.2.1 Estructura general

Explicado lo anterior, el Playbook se encuentra en la carpeta **ansible** y tiene una estructura de directorios general tal que:

```
## ../iac/ansible/
## +-- ansible.cfg
## +-- hosts
## +-- roles
## |   +-- docker
## |   +-- firewall
## |   +-- mysql
## |   \-- system
## +-- run.yml
## \-- vars
##     \-- main.yml
```

3.2.2 Funciones específicas

Como se puede apreciar, el Playbook posee varios subdirectorios, los cuales contienen dentro otros multiples archivos, se hará una descripción general de cada módulo.

Archivos base Se tienen tres archivos principales que describen el funcionamiento general del Ansible Playbook:

- **ansible.cfg**: posee configuraciones de Ansible referentes a las conexiones hacia la máquina objetivo.
- **hosts**: tiene la información de la máquina objetivo necesaria para ejecutar Ansible remotamente en ella.
- **run.yml**: explica a rasgos generales qué instrucciones ejecutará Ansible en la máquina objetivo.

Roles Los principales roles que se tienen y sus respectivas tareas son:

- **system**: realiza la configuración base del sistema operativo.
 - Instala y actualiza paquetes.
 - Inicia ciertos servicios esenciales.
 - Configura las opciones de seguridad de SSH.
- **firewall**: configura el firewall de la máquina a través de **ufw**.
 - Permite acceso por SSH únicamente a la dirección IP desde la que se ejecutó Terraform.

- Permite acceso por MySQL únicamente a las direcciones IP que usa el Azure App Service para conectarse.
- Bloquea por defecto el resto de conexiones hacia la VM.

Aclaración: las reglas que genera Docker al usar el redireccionamiento de puertos están por encima del firewall, por lo que se podrá conectar a la base de datos desde cualquier dirección IP. Una de las soluciones para solventar esto es usar directamente el modo de red `host`.

- **docker:** realiza la configuración de Docker.
 - Instala e inicia el servicio de Docker.
 - Añade el usuario por defecto al grupo `docker`.
- **mysql:** realiza la configuración de la base de datos.
 - Copia archivos esenciales.
 - Construye la imagen de la base de datos.
 - Corre el contenedor con la imagen de la base de datos.

Variables Si bien cada rol maneja internamente sus propias variables, también se busca que haya ciertas variables globales para que todos los roles tengan acceso a ellas; estas son definidas en `vars/main.yml`.

3.3 Terraform

Terraform funciona con archivos de configuración que permiten desplegar una infraestructura en la nube automáticamente a través de un archivo de configuración inicial. Para esto se usan los siguientes archivos dentro de la carpeta `terraform`. En este caso, el directorio solo contiene unos cuantos archivos así que se explicará la utilidad de cada uno.

3.3.1 Funciones específicas

- **variables.tf:** maneja distintas variables.
 - Tenant ID.
 - Subscription ID.
 - Nombre de usuario de la VM.
 - Nombre del host de la VM.
- **output.tf:** imprime algunos valores luego de ejecutar Terraform.
 - IP de la VM con la DB. De modo que se pueda acceder fácilmente a ella por SSH en caso de que sea necesario.
- **main.tf:** contiene toda la configuración para el despliegue de los recursos de la infraestructura.
 - Crea un grupo de recursos donde estará la VM de la DB y el App Service de la API.
 - Configura el networking de dicho grupo, creando una subred, asignando una IP pública y permitiendo únicamente los puertos 22 (SSH) y 3306 (MySQL) en la VM.

- Crea un par de llaves para el acceso por SSH.
- Crea la VM usando Debian 12 como sistema operativo.
- Configura y crea el App Service para la API Web.
- **post-apply-script.sh**: script de **bash** que se usa para ejecutar el Ansible Playbook y otros comandos más luego de ejecutar el código de Terraform y crear la infraestructura; es lo que configura e interconecta todos los componentes de la infraestructura entre sí. Esto se logra tal que:
 - Se reemplazan dentro de los archivos de Ansible y de la Web App las distintas direcciones IP públicas de las máquinas involucradas en la infraestructura:
 - * VM con la DB, para poder tener la dirección IP a la que realizar la conexión de Ansible por SSH y desplegar la configuración remotamente.
 - * Máquina en la que se están ejecutando los comandos, para permitir acceso de SSH a la VM solo a través de la IP de este equipo.
 - * App Service en el que está corriendo la API, se toman sus distintas direcciones IP de salida con el fin de permitirle el acceso al servidor MySQL únicamente desde esta dirección.
 - Se genera un archivo comprimido **.zip** de la carpeta **nodejs-express-mysql**, el cual se usa para desplegar la aplicación en el Azure App Service con la dirección IP actualizada de la base de datos.

Como se puede apreciar, todas estas labores permiten que la totalidad del proceso sea automática con mínima intervención por parte del administrador.

3.4 Azure CLI

4 Despliegue

Una vez que están todos los archivos listos se procede a desplegar la infraestructura; para esto, nos dirigimos al directorio de `terraform`, ya que a través de esta herramienta es que se pone en marcha todo el despliegue automático de las máquinas y sus respectivas configuraciones.

4.1 `terraform init`

Lo primero consiste en iniciar el entorno de trabajo de Terraform:

```
usuario@reimu: terraform % master
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/azurerm versions matching "~> 3.0"...
- Finding latest version of hashicorp/local...
- Finding latest version of hashicorp/tls...
- Installing hashicorp/azurerm v3.75.0...
- Installed hashicorp/azurerm v3.75.0 (signed by HashiCorp)
- Installing hashicorp/local v2.4.0...
- Installed hashicorp/local v2.4.0 (signed by HashiCorp)
- Installing hashicorp/tls v4.0.4...
- Installed hashicorp/tls v4.0.4 (signed by HashiCorp)

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.
```

4.2 `terraform validate`

Una vez descargados los proveedores a usar, continuamos validando que nuestra configuración sea válida:

```
usuario@reimu: terraform ♯ master
$ terraform validate
Success! The configuration is valid.
```

4.3 terraform apply

Finalmente, damos marcha al despliegue como tal de la infraestructura y su configuración. En primer lugar, nos hará una vista general de todo lo que se creará con Terraform:

```
usuario@reimu: terraform ♯ master
$ 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:

# azurerm_linux_virtual_machine.db will be created
+ resource "azurerm_linux_virtual_machine" "db" {
  + admin_username           = "azureuser"
  + allow_extension_operations = true
  + bypass_platform_security_checks_on_user_schedule_enabled = false
}
```

Ahora bien, confirmamos estos cambios y empezará a crear los recursos de la infraestructura:

```
Plan: 12 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ app_url = "parcial2ipti.azurewebsites.net"
+ db_ip   = (known after apply)

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

tls_private_key.db: Creating...
tls_private_key.db: Creation complete after 1s [id=ddc5ec5e883a8488f5a7aad7ef4b551445040cd7]
local_sensitive_file.db: Creating...
local_sensitive_file.db: Creation complete after 0s [id=e01c066461e560f3542e00d0e432125f9996f6772]
```

Así sería el proceso de ejecución, cuando termina de crear los recursos ejecuta el script:

```

azurermlinux_virtual_machine.db: Still creating... [10s elapsed]
azurermlinux_web_app.parcial2ipti: Still creating... [30s elapsed]
azurermlinux_virtual_machine.db: Still creating... [20s elapsed]
azurermlinux_web_app.parcial2ipti: Still creating... [40s elapsed]
azurermlinux_virtual_machine.db: Still creating... [30s elapsed]
azurermlinux_web_app.parcial2ipti: Creation complete after 50s [id=/subscriptions/4932121c-77b7-40c6-9415-f82ad0199aa9/resourceGroups/parcial2
parcial2ipti]
azurermlinux_virtual_machine.db: Still creating... [40s elapsed]
azurermlinux_virtual_machine.db: Still creating... [50s elapsed]
azurermlinux_virtual_machine.db: Provisioning with 'local-exec'...
azurermlinux_virtual_machine.db (local-exec): Executing: ["/bin/sh" "-c" "./post-apply-script.sh azureuser db 20.172.199.171 ../secrets/db"]
azurermlinux_virtual_machine.db: Still creating... [1m0s elapsed]

```

Siguiendo este proceso, podemos ver el resultado de desplegar el .zip en el que se encuentra la App Web:

```

azurermlinux_virtual_machine.db (local-exec): {
azurermlinux_virtual_machine.db (local-exec):   "active": true,
azurermlinux_virtual_machine.db (local-exec):   "author": "N/A",
azurermlinux_virtual_machine.db (local-exec):   "author_email": "N/A",
azurermlinux_virtual_machine.db (local-exec):   "build_summary": {
azurermlinux_virtual_machine.db (local-exec):     "errors": [],
azurermlinux_virtual_machine.db (local-exec):     "warnings": []
azurermlinux_virtual_machine.db (local-exec):   },
azurermlinux_virtual_machine.db (local-exec):   "complete": true,
azurermlinux_virtual_machine.db (local-exec):   "deployer": "OneDeploy",
azurermlinux_virtual_machine.db (local-exec):   "end_time": "2023-10-12T00:07:42.3710515Z",
azurermlinux_virtual_machine.db (local-exec):   "id": "0151d423-9750-4673-8613-80067cfdb844",
azurermlinux_virtual_machine.db (local-exec):   "is_readonly": true,
azurermlinux_virtual_machine.db (local-exec):   "is_temp": false,
azurermlinux_virtual_machine.db (local-exec):   "last_success_end_time": "2023-10-12T00:07:42.3710515Z",
azurermlinux_virtual_machine.db (local-exec):   "log_url": "https://parcial2ipti.scm.azurewebsites.net/api/deployments/0151d423-9750-4673-8613-80067cfdb844/log",
azurermlinux_virtual_machine.db (local-exec):   "message": "OneDeploy",
azurermlinux_virtual_machine.db (local-exec):   "progress": "",
azurermlinux_virtual_machine.db (local-exec):   "received_time": "2023-10-12T00:06:50.5954465Z",
azurermlinux_virtual_machine.db (local-exec):   "site_name": "parcial2ipti",
azurermlinux_virtual_machine.db (local-exec):   "start_time": "2023-10-12T00:06:51.9966606Z",
azurermlinux_virtual_machine.db (local-exec):   "status": 4,
azurermlinux_virtual_machine.db (local-exec):   "status_text": "",
azurermlinux_virtual_machine.db (local-exec):   "url": "https://parcial2ipti.scm.azurewebsites.net/api/deployments/0151d423-9750-4673-8613-80067cfdb844"
azurermlinux_virtual_machine.db (local-exec): }

```

Y vemos cómo se inicia el Ansible Playbook:

```

azurermlinux_virtual_machine.db: Still creating... [2m50s elapsed]

azurermlinux_virtual_machine.db (local-exec): PLAY [Base installation] *****

azurermlinux_virtual_machine.db (local-exec): TASK [Gathering Facts] *****
jazermlinux_virtual_machine.db (local-exec): ok: [db]

azurermlinux_virtual_machine.db (local-exec): TASK [system : include_tasks] *****
azurermlinux_virtual_machine.db (local-exec): included: /home/usuario/documentos/universidad/sistemas/infraestructura_para_ti/p
sks/essential.yml for db

azurermlinux_virtual_machine.db (local-exec): TASK [system : Install aptitude] *****
azurermlinux_virtual_machine.db: Still creating... [3m0s elapsed]

```

4.4 Resultados

Finalmente, al terminar de ejecutar el script, Terraform finaliza la infraestructura:

```

azurermlinux_virtual_machine.db (local-exec): PLAY RECAP *****
azurermlinux_virtual_machine.db (local-exec): db                               : ok=26   changed=19   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0

azurermlinux_virtual_machine.db: Creation complete after 7m18s [id=/subscriptions/4932121c-77b7-40c6-9415-f82ad0199aa9/resourceGroups/parcial2/providers/Microsoft.Compute/vi
rtualMachines/db]

Apply complete! Resources: 12 added, 0 changed, 0 destroyed.

Outputs:

app_url = "parcial2ipti.azurewebsites.net"
db_ip = "20.172.199.171"

```

5 Pruebas

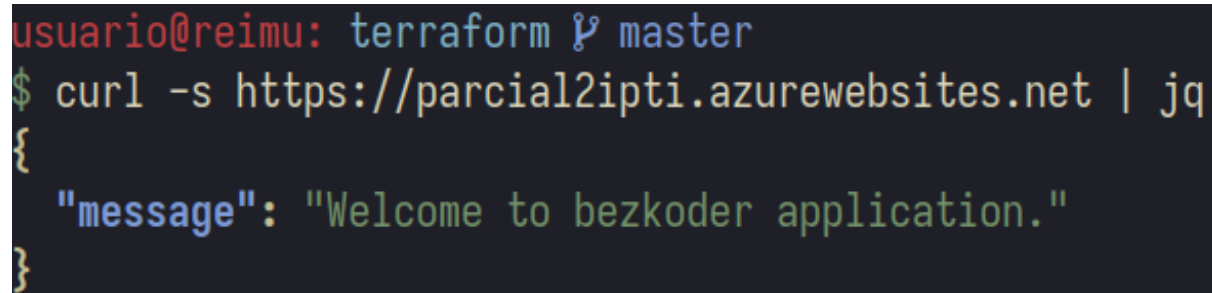
Una vez la infraestructura se encuentra en ejecución, realizaremos las pruebas que validen que se ha configurado correctamente.

5.1 API Web

Como primera prueba nos remitiremos al Azure App Service, ya que se trata del recurso principal de esta infraestructura.

5.1.1 Funcionamiento general

En primer lugar, veremos si se ha desplegado la API Web en el dominio que nos indicó Terraform, así como también validaremos que se está usando HTTPS:



```
usuario@reimu: terraform & master
$ curl -s https://parcial2ipti.azurewebsites.net | jq
{
  "message": "Welcome to bezkoder application."
}
```

5.1.2 Conexión con la base de datos

Lo siguiente es verificar que la API sea capaz de hacer peticiones a la base de datos y retornar sus valores:

```

usuario@reimu: terraform & master
$ echo "La dirección IP pública de este equipo es: $(curl -s ipconfig.org)" \
> && curl -s https://parcial2ipti.azurewebsites.net/api/tutorials | jq
La dirección IP pública de este equipo es: 186.116.70.138
[
  {
    "id": 1,
    "title": "Node.js Basics",
    "description": "Tut#1 Description",
    "published": 0
  },
  {
    "id": 2,
    "title": "Rest APIs",
    "description": "Tut#2 Description",
    "published": 0
  },
  {
    "id": 3,
    "title": "Node Rest APIs",
    "description": "Tut#3 Description",
    "published": 0
  },
  {
    "id": 4,
    "title": "MySQL database",
    "description": "Tut#4 Description",
    "published": 0
  },
  {
    "id": 5,
    "title": "Node Rest Apis with MySQL",
    "description": "Tut#5 Description",
    "published": 0
  }
]

```

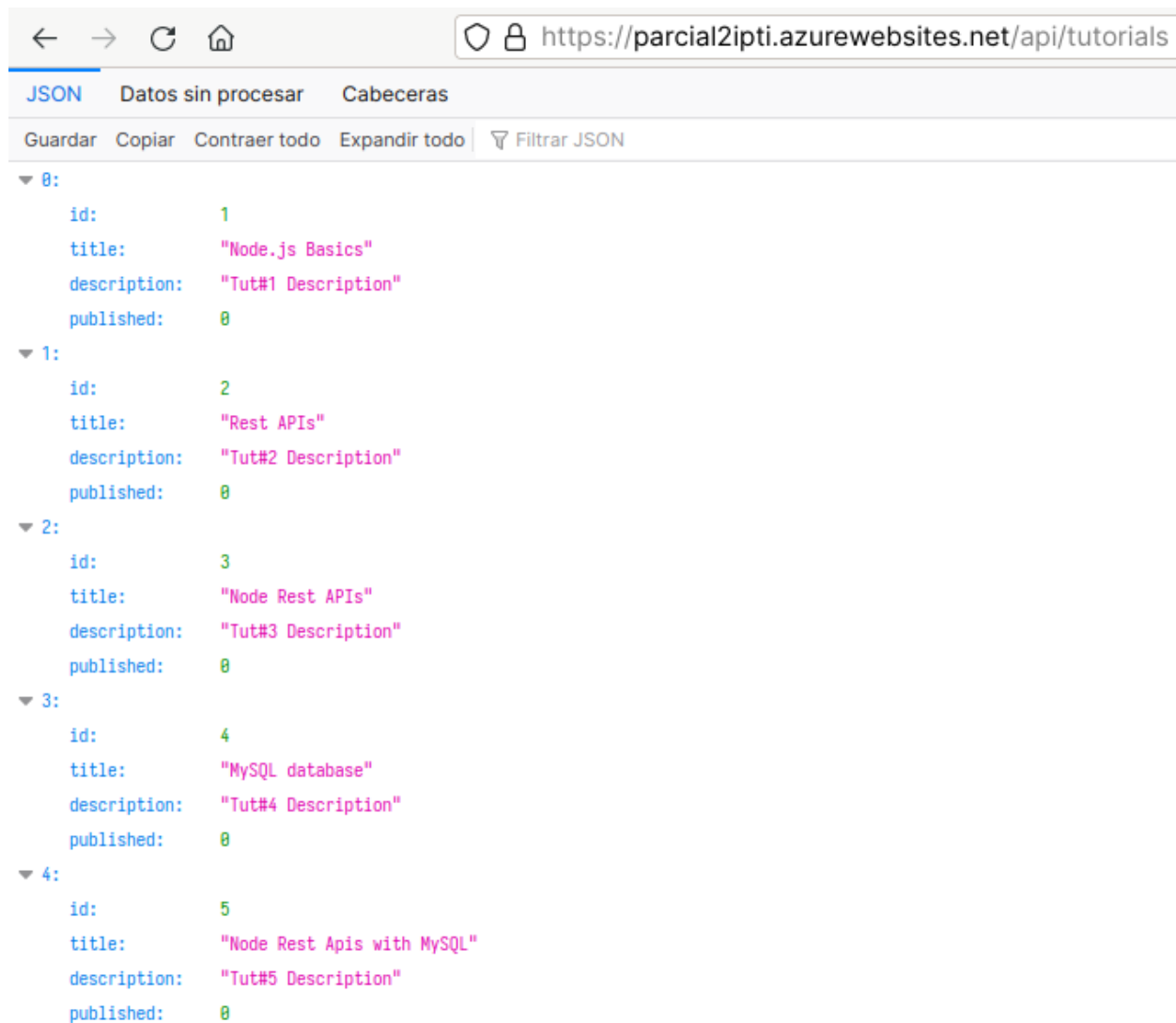
Desde otra máquina se obtiene el mismo resultado:


```

debian@tenshi:~$ echo "La dirección IP pública de este equipo es: $(curl -s ipconfig.org)" \
> && curl -s https://parcial2ipti.azurewebsites.net/api/tutorials | jq
La dirección IP pública de este equipo es: 51.222.143.27
[
  {
    "id": 1,
    "title": "Node.js Basics",
    "description": "Tut#1 Description",
    "published": 0
  },
  {
    "id": 2,
    "title": "Rest APIs",
    "description": "Tut#2 Description",
    "published": 0
  },
  {
    "id": 3,
    "title": "Node Rest APIs",
    "description": "Tut#3 Description",
    "published": 0
  },
  {
    "id": 4,
    "title": "MySQL database",
    "description": "Tut#4 Description",
    "published": 0
  },
  {
    "id": 5,
    "title": "Node Rest Apis with MySQL",
    "description": "Tut#5 Description",
    "published": 0
  }
]

```

Lo mismo pasa desde un navegador:



5.2 Máquina virtual con la base de datos

Ahora bien, se realizan las pruebas a la base de datos referentes a su configuración de seguridad con respecto al firewall y a Docker.

5.2.1 SSH

En primer lugar, se hará una conexión por SSH a la máquina virtual desde el equipo que corrió el comando de Terraform:

```

usuario@reimu: terraform & master
$ echo "La dirección IP pública de este equipo es: $(curl -s ipconfig.org)" \
> && ssh azureuser@20.172.199.171 -i ../secrets/db
La dirección IP pública de este equipo es: 186.116.70.138
Linux db 6.1.0-12-cloud-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.52-1 (2023-09-07) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Oct 12 00:23:44 2023 from 186.116.70.138
azureuser@db:~$

```

Como se puede ver, sí es capaz de conectarse desde la máquina de administración. No obstante, si se intenta la misma conexión desde un dispositivo distinto:

```

debian@tenshi:~$ echo "La dirección IP pública de este equipo es: $(curl -s ipconfig.org)" \
> && ssh azureuser@20.172.199.171
La dirección IP pública de este equipo es: 51.222.143.27
ssh: connect to host 20.172.199.171 port 22: Connection timed out

```

El servidor no acepta la conexión ya que proviene de una IP distinta a la de administración.

5.2.2 Docker

Una vez dentro de la máquina virtual, podemos verificar que está corriendo el contenedor indicado:

```

azureuser@db:~$ docker ps -a

```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
94eb9a3b9d54	mysql_database	"docker-entrypoint.s.."	21 minutes ago	Up 21 minutes		database

Aquí podemos verificar que efectivamente el contenedor está corriendo. Además, no indica puertos redireccionados ya que se encuentra en modo `host`.

5.2.3 Puertos

En ese orden de ideas, si vemos las conexiones realizadas a la máquina tenemos:

```

azureuser@db:~$ netstat -tn

```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	10.0.1.4:22	186.116.70.138:57086	ESTABLISHED
tcp6	0	0	10.0.1.4:3306	20.241.128.86:27584	ESTABLISHED

Se puede ver que solo hay dos puertos recibiendo tráfico: el de SSH que se está usando para esta prueba proveniente desde la IP de administración y el de MySQL proveniente de una de las IPs de salida de la API Web.

5.2.4 MySQL

Siguiendo con las pruebas de seguridad, ya se verificó que la API Web corriendo desde un Azure App Service tiene acceso a la base de datos: entrar a la dirección de la API retorna los valores esperados, así como desde el análisis de puerto se ve dicha conexión. Sin embargo, se debe verificar que solo la API tenga acceso a la base de datos, por lo que, si intentamos conectarnos desde cualquier otra máquina (incluso la de administración) tenemos:

```
usuario@reimu: terraform ↵ master
$ echo "La dirección IP pública de este equipo es: $(curl -s ipconfig.org)" \
> && mysql -u root -h 20.172.199.171 -p123456 -e 'select * from tables tutorials;' appdb
La dirección IP pública de este equipo es: 186.116.70.138
mysql: [Warning] Using a password on the command line interface can be insecure.
ERROR 2003 (HY000): Can't connect to MySQL server on '20.172.199.171:3306' (110)
```

Y desde otra máquina distinta a ésta el resultado es el mismo:

```
debian@tenshi:~$ echo "La dirección IP pública de este equipo es: $(curl -s ipconfig.org)" \
> && mysql -u root -h 20.172.199.171 -p123456 -e 'select * from tables tutorials;' appdb
La dirección IP pública de este equipo es: 51.222.143.27
ERROR 2002 (HY000): Can't connect to server on '20.172.199.171' (115)
```

5.2.5 Firewall

Si bien ya verificamos que la seguridad de la infraestructura es la esperada, no está de más verificar las reglas del firewall en pos de asegurarse que son las correctas:

```
azureuser@db:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: reject (incoming), allow (outgoing), deny (routed)
New profiles: skip
```

To	Action	From
--	-----	----
22/tcp	ALLOW IN	186.116.70.138
3306/tcp	ALLOW IN	20.124.51.21
3306/tcp	ALLOW IN	20.241.128.86
3306/tcp	ALLOW IN	20.124.50.142
3306/tcp	ALLOW IN	20.124.51.200
3306/tcp	ALLOW IN	20.241.128.180
3306/tcp	ALLOW IN	20.241.128.226
3306/tcp	ALLOW IN	20.119.8.41

Es así como se finalizan las pruebas evidenciando que el despliegue de la infraestructura fue completamente satisfactorio.