Automated Flask Environment

Automated Setup of a Flask Web App with Vagrant, Gunicorn, Nginx & MySQL

V2Fzc2ltRg

2025

Introduction

This simple project aims to automate the provisioning of a complete development environment using **Vagrant** and a custom **Bash script**. It sets up a basic **Flask web application** running on **Ubuntu**, configured with **MySQL**, **Gunicorn**, and **Nginx**, all inside a virtual machine.

The main goal is to provide a **plug-and-play lab environment** for anyone who want to quickly deploy and test a realistic web application stack without using Docker or any cloud infrastructure.

By running a single command (`vagrant up`), you will have:

- A working **Flask app** with database connectivity
- A preconfigured **MySQL database** with an example table and sample data
- Gunicorn as the **WSGI** server and Nginx as the **reverse proxy**

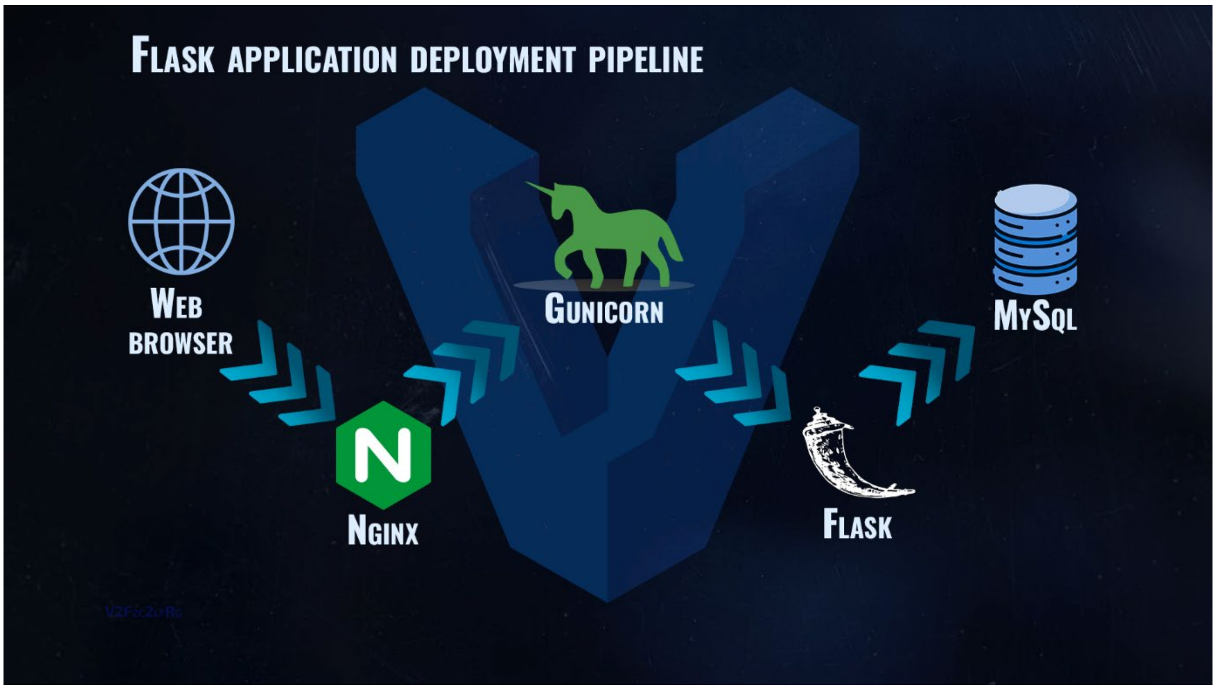
WSGI (Web Server Gateway Interface) is a standard interface between web servers and Python web applications or frameworks.

- Basic hardening with **UFW** and **Fail2Ban**
- A simple **CRUD interface** over `/users`, and a `/health` route for monitoring

This VM is ideal for:

- Rapid prototyping
- Learning provisioning and deployment processes

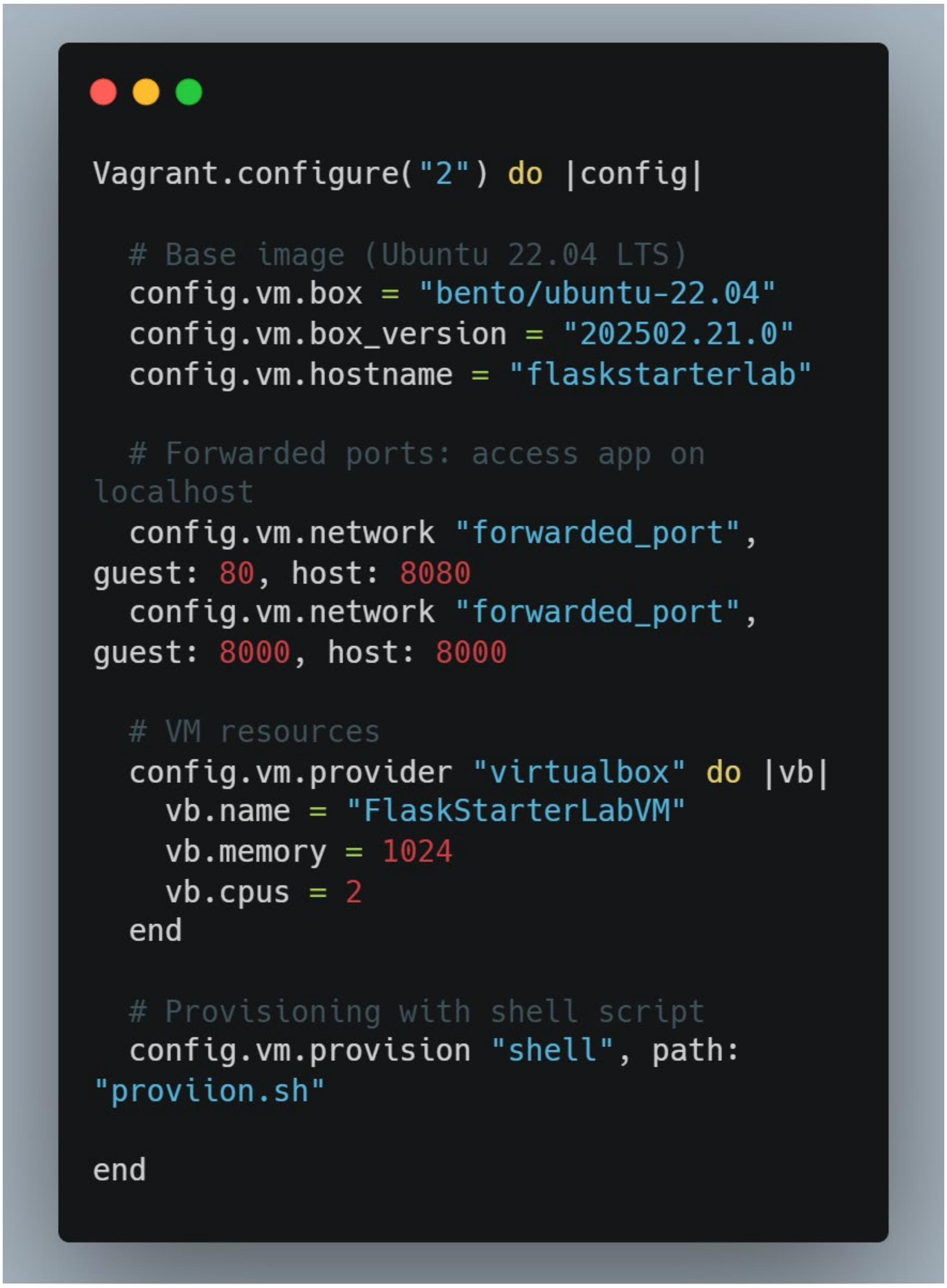
Project architecture



The architecture includes:

- **Vagrant**: Used to create and manage a disposable and reproducible development environment.
- **Bash Script**: Automates the entire provisioning process: installation, configuration, app setup, and deployment.
- **Flask**: A lightweight and flexible Python web framework used to build the web app.
- **MySQL**: A relational database management system used to store and retrieve user data.
- **Gunicorn**: A WSGI-compliant application server that runs the Flask app in a production setting.
- **Nginx**: A high-performance web server that acts as a reverse proxy, forwarding client requests to Gunicorn.
- **UFW (Uncomplicated Firewall):** Configured to allow only essential traffic (SSH and HTTP).
- **Fail2Ban**: Monitors log files for suspicious activity and blocks potential brute-force attacks.

Vagrant file



```
Vagrant.configure("2") do |config|

  # Base image (Ubuntu 22.04 LTS)
  config.vm.box = "bento/ubuntu-22.04"
  config.vm.box_version = "202502.21.0"
  config.vm.hostname = "flaskstarterlab"

  # Forwarded ports: access app on
localhost
  config.vm.network "forwarded_port",
guest: 80, host: 8080
  config.vm.network "forwarded_port",
guest: 8000, host: 8000

  # VM resources
  config.vm.provider "virtualbox" do |vb|
    vb.name = "FlaskStarterLabVM"
    vb.memory = 1024
    vb.cpus = 2
  end

  # Provisioning with shell script
  config.vm.provision "shell", path:
"proviion.sh"

end
```

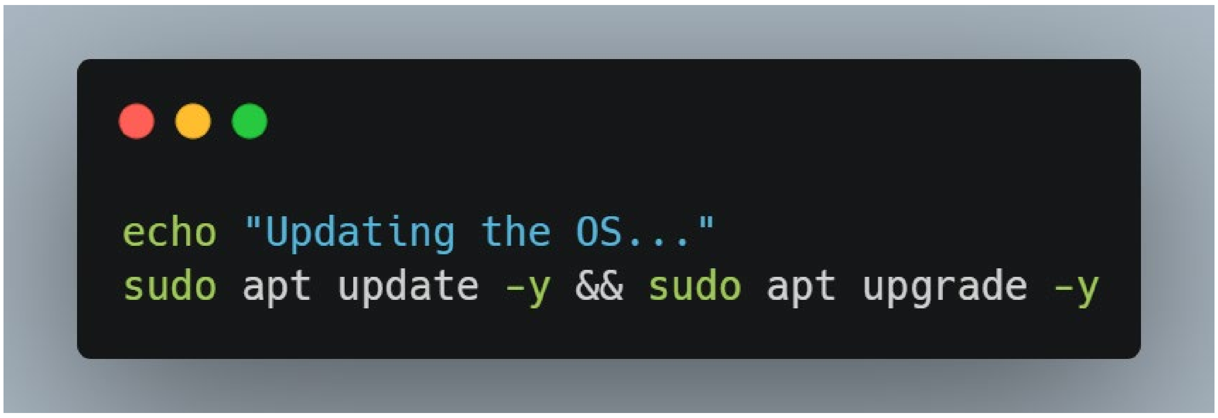
V2Fzc2ItRg

Automated Flask Environment

This file defines the VM configuration for the development environment.

- `config.vm.box`: Specifies the base image used to create the VM. We use the Bento **Ubuntu** 22.04 LTS box, which provides a clean Linux environment.
- `config.vm.hostname`: Assigns a hostname to the VM, useful for identification within a networked environment.
- `config.vm.network:`
  - Maps port 80 (**used by Nginx**) inside the VM to port 8080 on the host, allowing you to access the app via **http://localhost:8080**.
  - Maps port 8000 (**used by Flask development server**) to port 8000 on the host, for local debugging or testing.
- `config.vm.provider`:
  - Allocates resources to the VM: 1024MB of RAM and 2 CPU cores.
  - Sets a custom name for the VM inside VirtualBox, making it easier to identify.
- `config.vm.provision:` Points to the **provision.sh** script, which will automatically install and configure the entire stack when the VM is started.

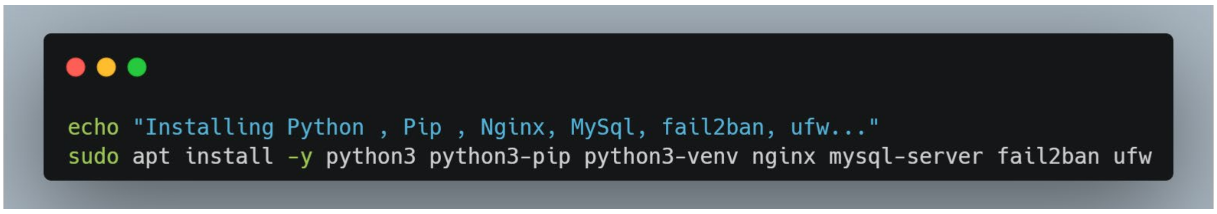
Provision.sh



```
echo "Updating the OS..."
sudo apt update -y && sudo apt upgrade -y
```

This command ensures the system is up to date before installing any new packages, it retrieves the latest package information from all configured **repositories** and installs the **newest versions** of all currently installed packages.

The **-y** flag automatically **confirms** all prompts, allowing the process to run without manual intervention.



```
echo "Installing Python , Pip , Nginx, MySql, fail2ban, ufw..."
sudo apt install -y python3 python3-pip python3-venv nginx mysql-server fail2ban ufw
```

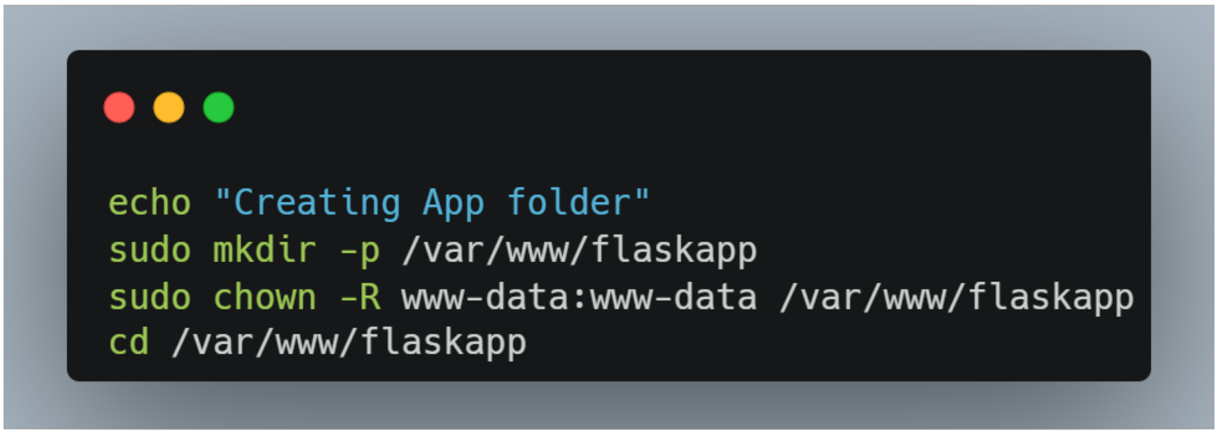
This command installs the essential components needed to build and secure the application environment:

- **Python3 & pip**: These are required to run the Flask application and manage Python dependencies.
- **python3-venv**: Used to create a virtual environment, isolating the app's Python packages from the system.
- **Nginx**: Web server that will act as a reverse proxy to forward requests to the Flask app running with Gunicorn.
  *A **reverse proxy** is a server that receives HTTP requests from users and forwards them to another server*

- **MySQL Server**: Relational database used to store user data for the application.
- **fail2ban**: Monitors log files and blocks IP addresses that show signs of malicious activity like multiple failed login attempts, it’s a **Brute-force protection**.

Automated Flask Environment

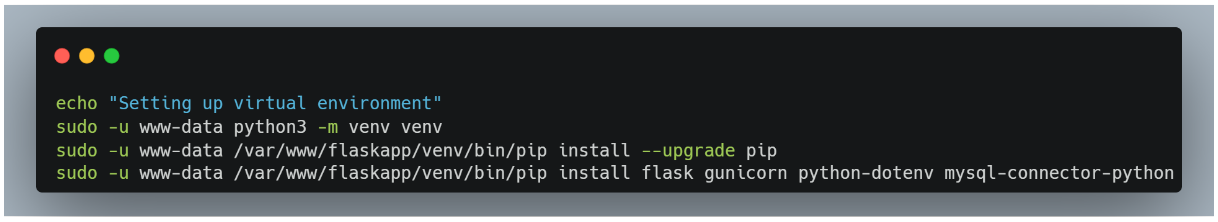
- **ufw (Uncomplicated Firewall)**: A firewall tool used to manage network access rules and increase server security.



```
echo "Creating App folder"
sudo mkdir -p /var/www/flaskapp
sudo chown -R www-data:www-data /var/www/flaskapp
cd /var/www/flaskapp
```

This set of commands creates the directory where the Flask app will reside, adjusts its permissions, and navigates to it.

- **`mkdir -p /var/www/flaskapp`** :
  - Creates the directory **`/var/www/flaskapp`**, which will contain the Flask application. The **`-p`** flag ensures that the entire directory path is created if it doesn't already exist.
- **`chown -R www-data:www-data /var/www/flaskapp`**:
  - This changes the ownership of the **`/var/www/flaskapp`** directory to the **`www-data user`** which is the default user for Nginx and Gunicorn, allowing them to read and write in this directory without permission issues. The **`-R`** flag ensures all files inside this directory will also have the right ownership.
- **`cd /var/www/flaskapp`**:
  - This navigates into the newly created application folder so that further commands are executed in the right location.

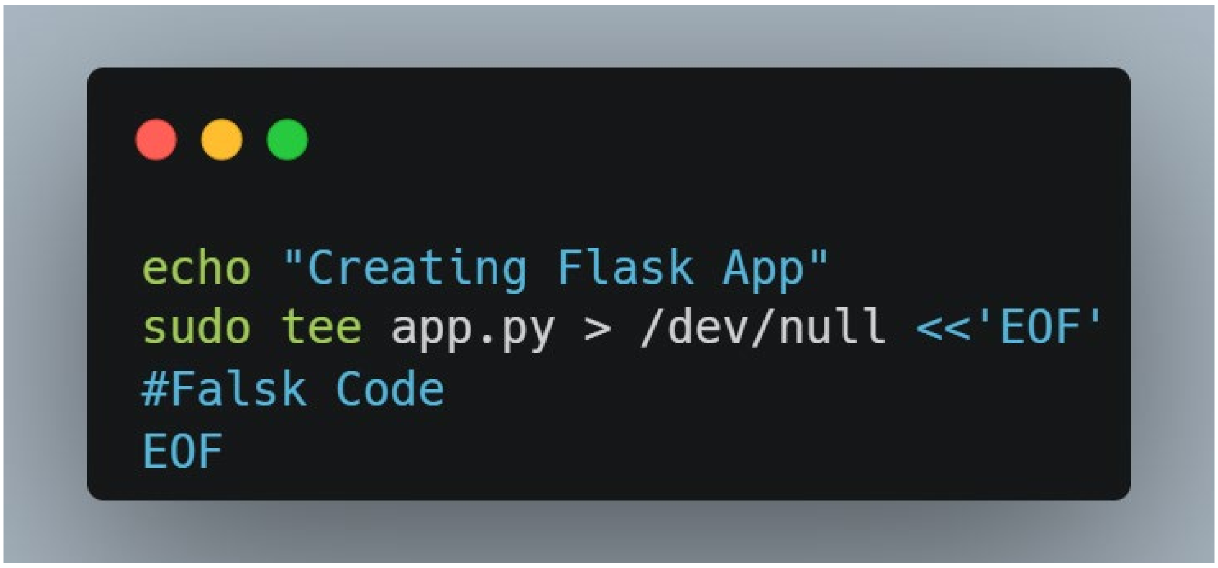


```
echo "Setting up virtual environment"
sudo -u www-data python3 -m venv venv
sudo -u www-data /var/www/flaskapp/venv/bin/pip install --upgrade pip
sudo -u www-data /var/www/flaskapp/venv/bin/pip install flask gunicorn python-dotenv mysql-connector-python
```

These commands create and activate a **virtual environment** for the Flask app. **`Virtual environments`** `help manage dependencies in isolation, ensuring that each project has its own dependencies, separate from the system's global Python packages.`

Automated Flask Environment

- **`sudo -u www-data`:**
  - runs the command as the **`www-data`** user, ensuring that this user has ownership of the virtual environment files.
- **`python3 -m venv venv`:**
  - Creates a virtual environment named **`venv`**. The **`-m venv`** flag tells Python to run the built-in **`venv`** module, which is responsible for creating the environment. The **`venv`** folder is where the isolated Python packages will reside.
- **`sudo -u www-data /var/www/flaskapp/venv/bin/pip install --upgrade pip`:**
  - This line upgrades pip, the Python package installer, **within the virtual environment**.
- **`sudo -u www-data /var/www/flaskapp/venv/bin/pip install flask gunicorn python-dotenv mysql-connector-python`:**
  - **`pip install python-dotenv`:**
    - **`python-dotenv`** is used to load environment variables from a **`.env`** file. This package is important for keeping **sensitive information** like database credentials out of the codebase.
  - **`Pip install mysql-connector-python`:**
    - Enables Python to interact with MySQL database.

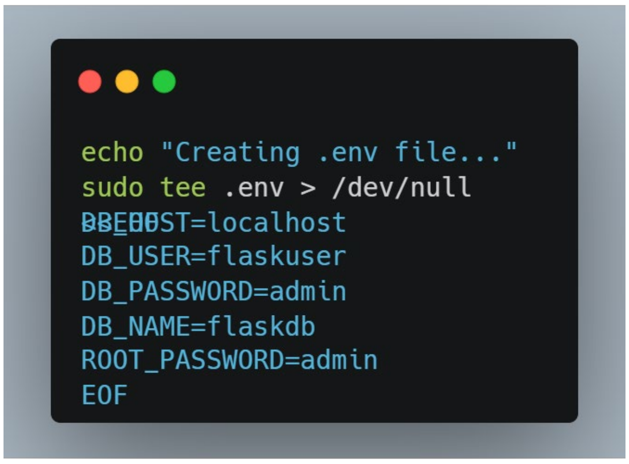


```
echo "Creating Flask App"
sudo tee app.py > /dev/null <<'EOF'
#Falsk Code
EOF
```

- **`sudo tee app.py`**: This creates a file named **`app.py`** with elevated privileges using **`sudo`**.
  `The tee command writes standard input to the file app.py.`
- **`>/dev/null`: `tee`** writes the content both to the file and to the terminal, But we don't want to display the entire script content on the screen, so by redirecting the output to **`/dev/null`**, we silence it.
- **`<<'EOF' ... EOF`:** This is a **here-document**, used to pass a block of text as input to the **`tee`** command. Everything between the opening

<<**'EOF'** and the closing **EOF** is treated as input to be written into **app.py**, here we are passing the Flask code.

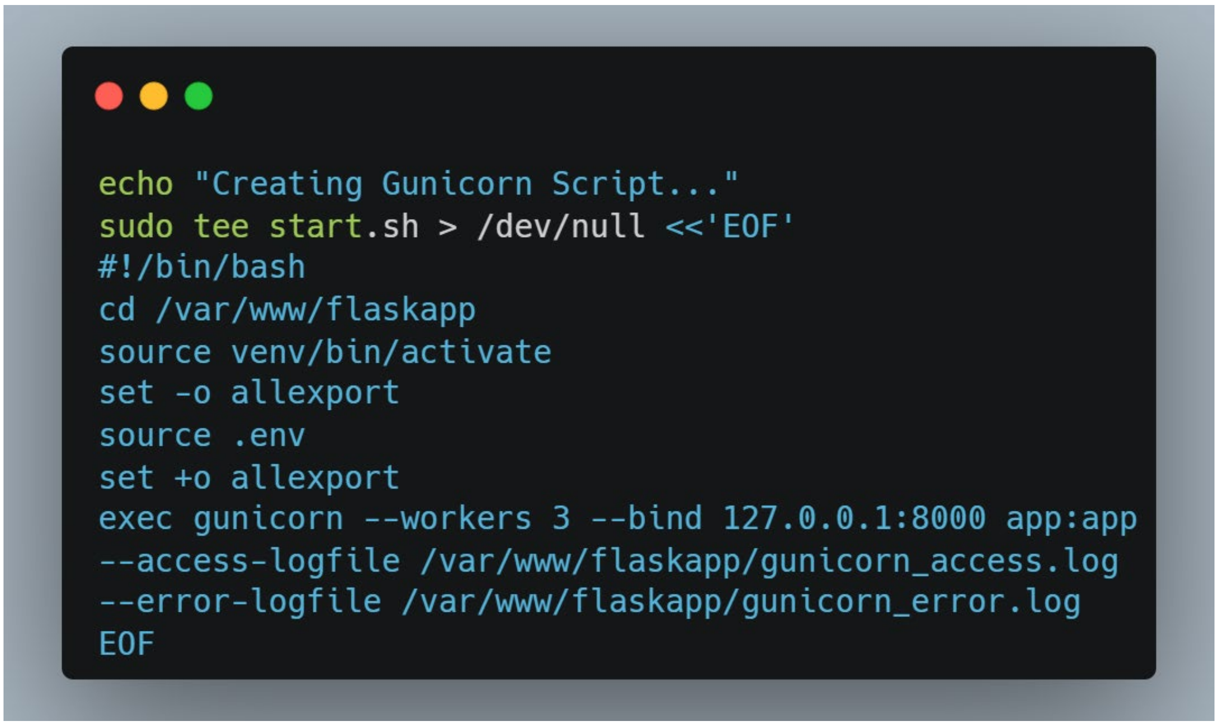
- Putting **single quotes** around **'EOF'** is useful when you're writing code that includes **$variables** or special characters, and you don't want the terminal to change them.



```
echo "Creating .env file..."
sudo tee .env > /dev/null
DB_HOST=localhost
DB_USER=flaskuser
DB_PASSWORD=admin
DB_NAME=flaskdb
ROOT_PASSWORD=admin
EOF
```

This file contains **environment variables** used by the **Flask app** and **database configuration**:

- **DB_HOST**: The database server hostname.
- **DB_USER**: The username used by the Flask app to connect to the database.
- **DB_PASSWORD**: The password for the user.
- **DB_NAME**: The name of the database used by the app.
- **ROOT_PASSWORD**: The MySQL **root password**, used during database setup.

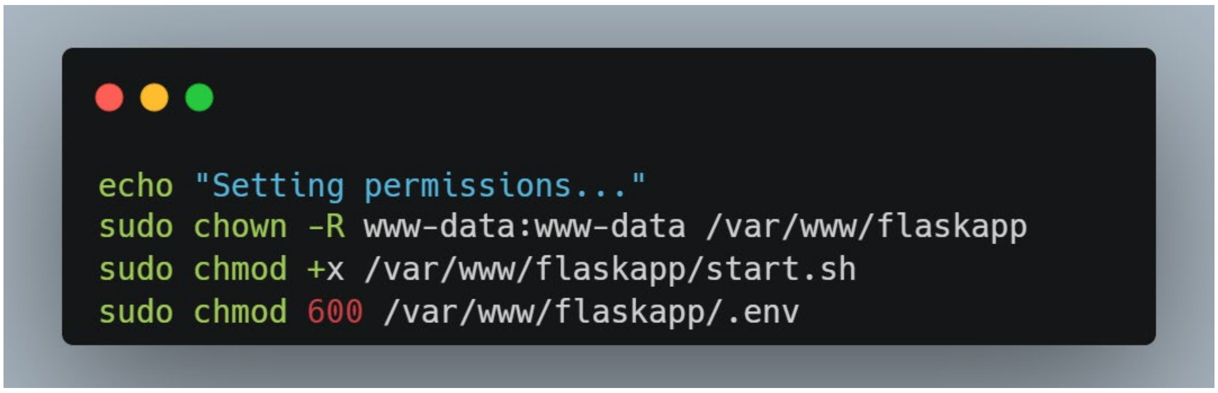


```
echo "Creating Gunicorn Script..."
sudo tee start.sh > /dev/null <<'EOF'
#!/bin/bash
cd /var/www/flaskapp
source venv/bin/activate
set -o allexport
source .env
set +o allexport
exec gunicorn --workers 3 --bind 127.0.0.1:8000 app:app
--access-logfile /var/www/flaskapp/gunicorn_access.log
--error-logfile /var/www/flaskapp/gunicorn_error.log
EOF
```

Automated Flask Environment

- **`#!/bin/bash`** :
  - Specifies that this script should be run using the Bash shell.
- **`cd /var/www/flaskapp:`**
  - Changes the current directory to the Flask app folder.
- **`source venv/bin/activate:`**
  - Activates the Python virtual environment so that the script uses the local Python packages.
- **`export DB_* variables`**:
  - Defines environment variables needed by the Flask app to connect to the database. These are passed into the app's runtime environment.
- **`set -o allexport :`**
  - Enables automatic exporting of environment variables, making them accessible to any subprocess.
- **`source .env:`**
  - Loads the environment variables defined in the **`.env`** file
- **`set +o allexport:`**
  - Disables the automatic export mode.
- **`exec gunicorn ...:`**
  Starts the Flask app using Gunicorn:
  - **`--workers 3`**:
    - Runs the app with 3 worker **processes** to handle multiple requests in parallel.
  - **`--bind 127.0.0.1:8000`**:
    - Binds the app to the localhost interface on port 8000 so Nginx can reverse proxy it.
  -
  - **`app:app`**:
    - Tells Gunicorn to look for the Flask instance **`app`** inside the **`app.py`** file

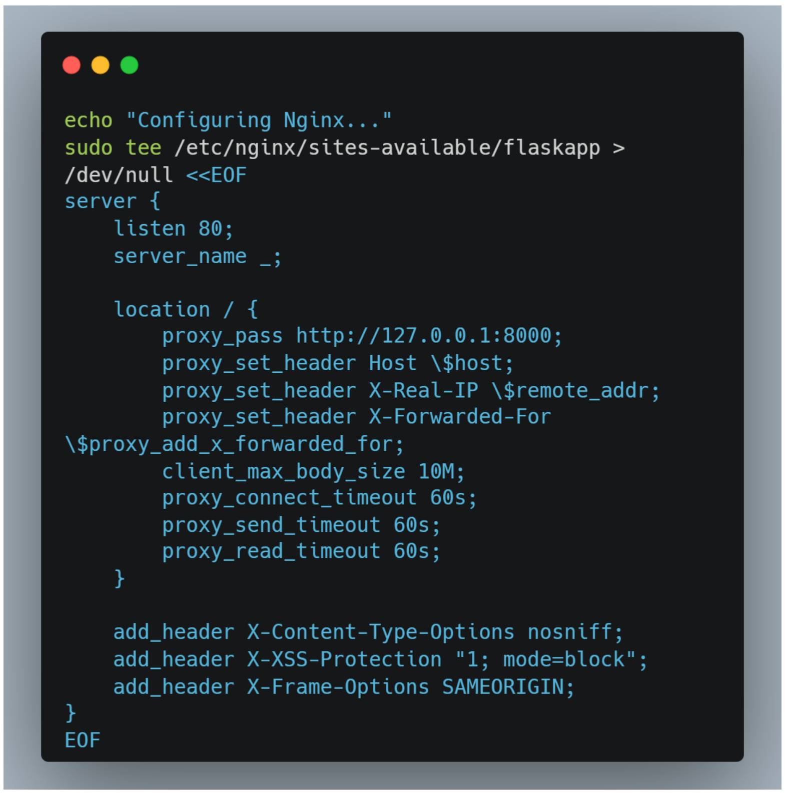
      `Using the syntax` **`"MODULE_NAME : VARIABLE_NAME".`**
  - **`--access-logfile / --error-logfile`**:
    - Logs HTTP access and errors to specific files.



```
echo "Setting permissions..."
sudo chown -R www-data:www-data /var/www/flaskapp
sudo chmod +x /var/www/flaskapp/start.sh
sudo chmod 600 /var/www/flaskapp/.env
```

- `sudo chown -R www-data:www-data /var/www/flaskapp`:
    - Even though this command is already executed earlier in the script, it is repeated here to ensure that files created later (**`.env`**, **`app.py`**, **`and start.sh`**) are also owned by **`www-data`**, not **`root`**.

- **`+x`**:
    - Makes the script **`start.sh`** executable.
- `600` means:
    - **Read and write** permissions for the file owner which is now **`www-data`**.
    - **No access** for group or others.



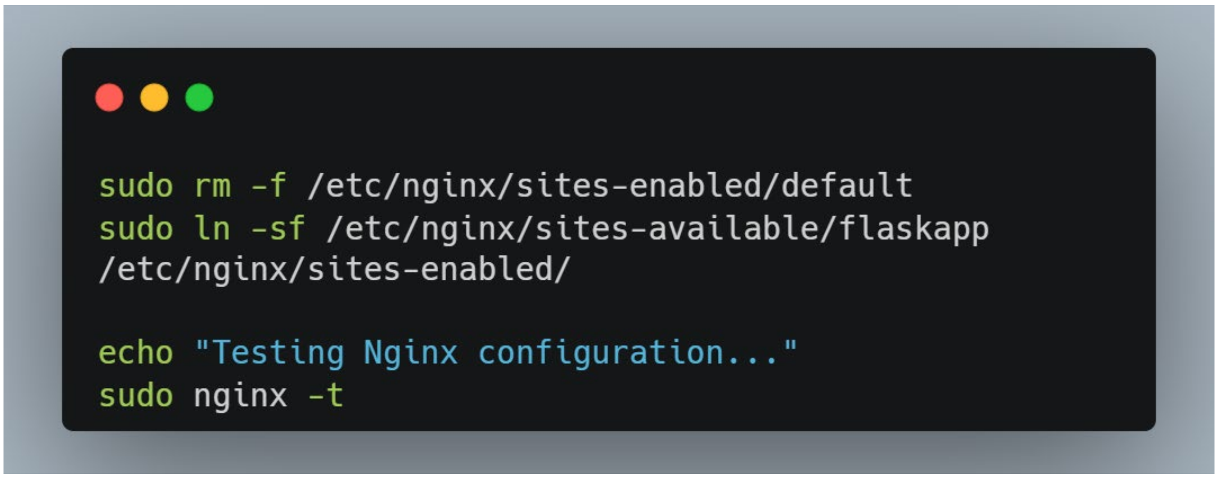
```
echo "Configuring Nginx..."
sudo tee /etc/nginx/sites-available/flaskapp >
/dev/null <<EOF
server {
    listen 80;
    server_name _;

    location / {
        proxy_pass http://127.0.0.1:8000;
        proxy_set_header Host \$host;
        proxy_set_header X-Real-IP \$remote_addr;
        proxy_set_header X-Forwarded-For
\$proxy_add_x_forwarded_for;
        client_max_body_size 10M;
        proxy_connect_timeout 60s;
        proxy_send_timeout 60s;
        proxy_read_timeout 60s;
    }

    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
    add_header X-Frame-Options SAMEORIGIN;
}
EOF
```

- **`sudo tee /etc/nginx/sites-available/flaskapp > /dev/null <<EOF`**:
    - This command creates a new Nginx configuration file called **`flaskapp`** inside the **`/etc/nginx/sites-available/`** directory.
- **`Server`**:
    - **`listen 80`**:
        - Tells Nginx to listen on **port 80** (standard **HTTP**).
    - **`server_name _`**:
        - Accepts **requests** from any **domain** or **IP**.
- **`location /`**:
    - **`proxy_pass http://127.0.0.1:8000`**:

    - Redirects **all incoming traffic** to **Gunicorn**, which listens on localhost:8000.
  - **`proxy_set_header Host \$host:`**
    - Preserves the original Host header from the client.
  - **`proxy_set_header X-Real-IP \$remote_addr:`**
    - Sends the real IP address of the client instead of Nginx's own IP.
  - **`proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for:`**
    - Keeps track of **all client IPs** in a **chain of proxies**.

  `A chain of proxies refers to a situation where a user's request passes through multiple layers of servers (proxies) before reaching your actual app.`

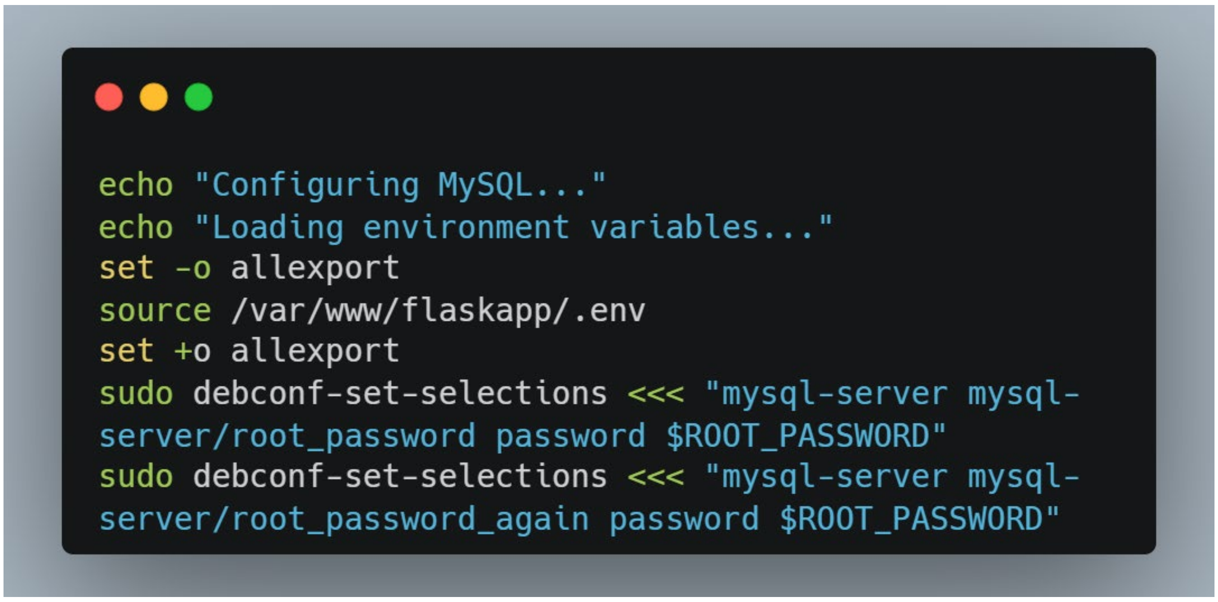
  - **`client_max_body_size 10M:`**
    - Limits **request body size** to 10 MB.
  - **`add_header X-Content-Type-Options nosniff:`**
    - Prevents browsers from sniffing the content type.
  - **`add_header X-XSS-Protection "1; mode=block":`**
    - Enables the browser's built-in XSS protection.
  - **`add_header X-Frame-Options SAMEORIGIN:`** Prevents the site from being embedded in an iframe on another domain.



```
sudo rm -f /etc/nginx/sites-enabled/default
sudo ln -sf /etc/nginx/sites-available/flaskapp
/etc/nginx/sites-enabled/

echo "Testing Nginx configuration..."
sudo nginx -t
```

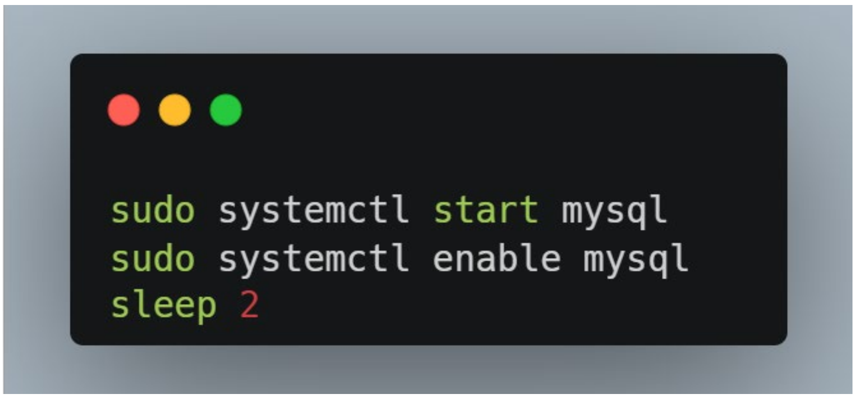
- **`sudo rm -f /etc/nginx/sites-enabled/default:`**
  - Removes the default Nginx site.
  - **`-f:`** forces the deletion without asking for confirmation.
- **`sudo ln -sf /etc/nginx/sites-available/flaskapp/etc/nginx/sites-enabled/:`**
  - Creates a **symbolic link** from the **Nginx config** file in **sites-available** to the **sites-enabled** directory, where Nginx looks for active site configurations.

  - **-s**: Creates a symbolic link.
  - **-f:** Replaces any existing link with the same name.
- **sudo nginx -t:**
  - Tests the nginx configuration file and checks if the Nginx configuration syntax is correct.



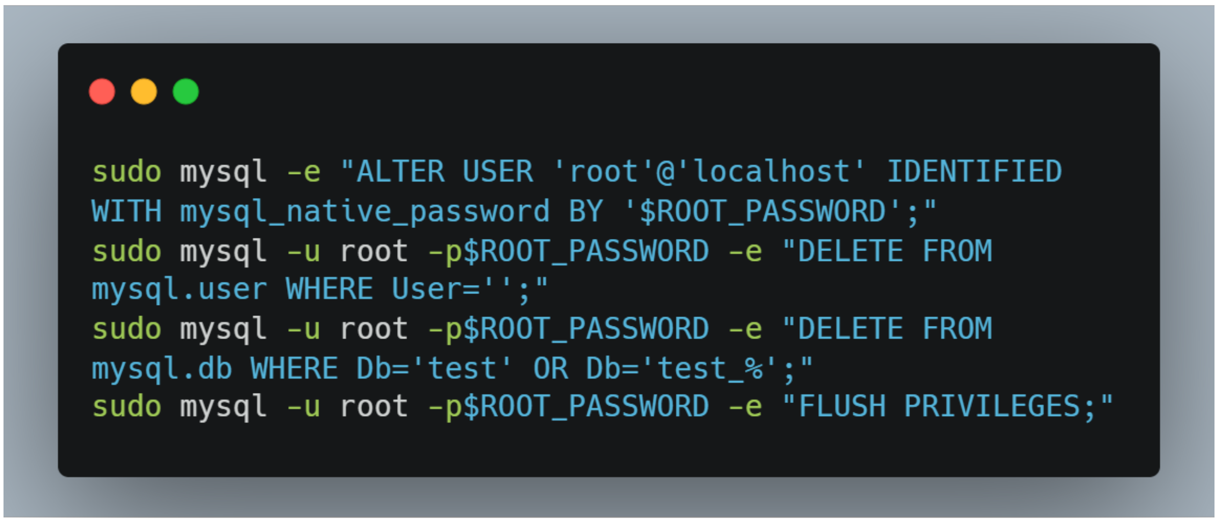
```
echo "Configuring MySQL..."
echo "Loading environment variables..."
set -o allexport
source /var/www/flaskapp/.env
set +o allexport
sudo debconf-set-selections <<< "mysql-server mysql-
server/root_password password $ROOT_PASSWORD"
sudo debconf-set-selections <<< "mysql-server mysql-
server/root_password_again password $ROOT_PASSWORD"
```

- **sudo debconf-set-selections <<< 'mysql-server mysql-server/root_password password $ROOT_PASSWORD'**
- **sudo debconf-set-selections <<< 'mysql-server mysql-server/root_password_again password $ROOT_PASSWORD''**
  - These commands pre-set the MySQL root password so that MySQL installation doesn't prompt for it interactively.



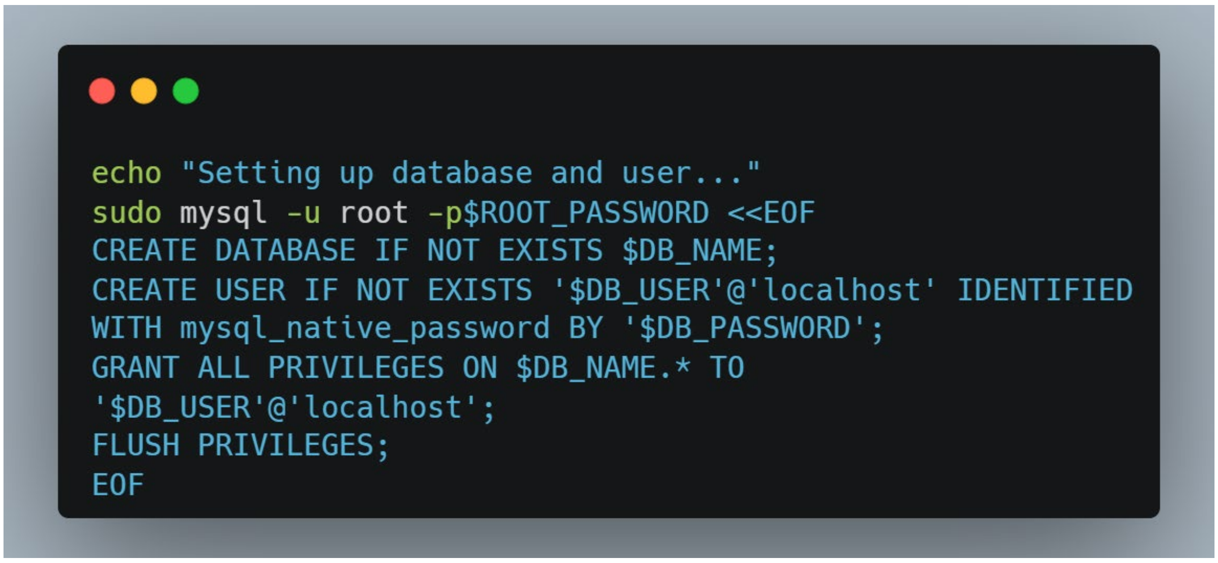
```
sudo systemctl start mysql
sudo systemctl enable mysql
sleep 2
```

- Starts the MySQL service and enables it to start on boot.
- **sleep 2**:
  - Waits a bit (2 seconds) to make sure the service is up before running SQL commands.



```
sudo mysql -e "ALTER USER 'root'@'localhost' IDENTIFIED
WITH mysql_native_password BY '$ROOT_PASSWORD';"
sudo mysql -u root -p$ROOT_PASSWORD -e "DELETE FROM
mysql.user WHERE User='';"
sudo mysql -u root -p$ROOT_PASSWORD -e "DELETE FROM
mysql.db WHERE Db='test' OR Db='test_%';"
sudo mysql -u root -p$ROOT_PASSWORD -e "FLUSH PRIVILEGES;"
```

- **`sudo mysql -e "ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '$ROOT_PASSWORD';"`:**
  - Updates the authentication method of the MySQL root user to use **`mysql_native_password`** instead of the default plugin.
  - It also sets the root password to the value of **`$ROOT_PASSWORD`**.
- **`sudo mysql -u root -p$ROOT_PASSWORD -e "DELETE FROM mysql.user WHERE User='';"`:**
  - Deletes all users with an empty name ').
- **`sudo mysql -u root -p$ROOT_PASSWORD -e "DELETE FROM mysql.db WHERE Db='test' OR Db='test_%';"`:**
  - Deletes the test databases **`(test, test_%)`** that MySQL installs by default.
- **`sudo mysql -u root -p$ROOT_PASSWORD -e "FLUSH PRIVILEGES;"`:**
  - Applies all recent changes by refreshing the in-memory permissions.

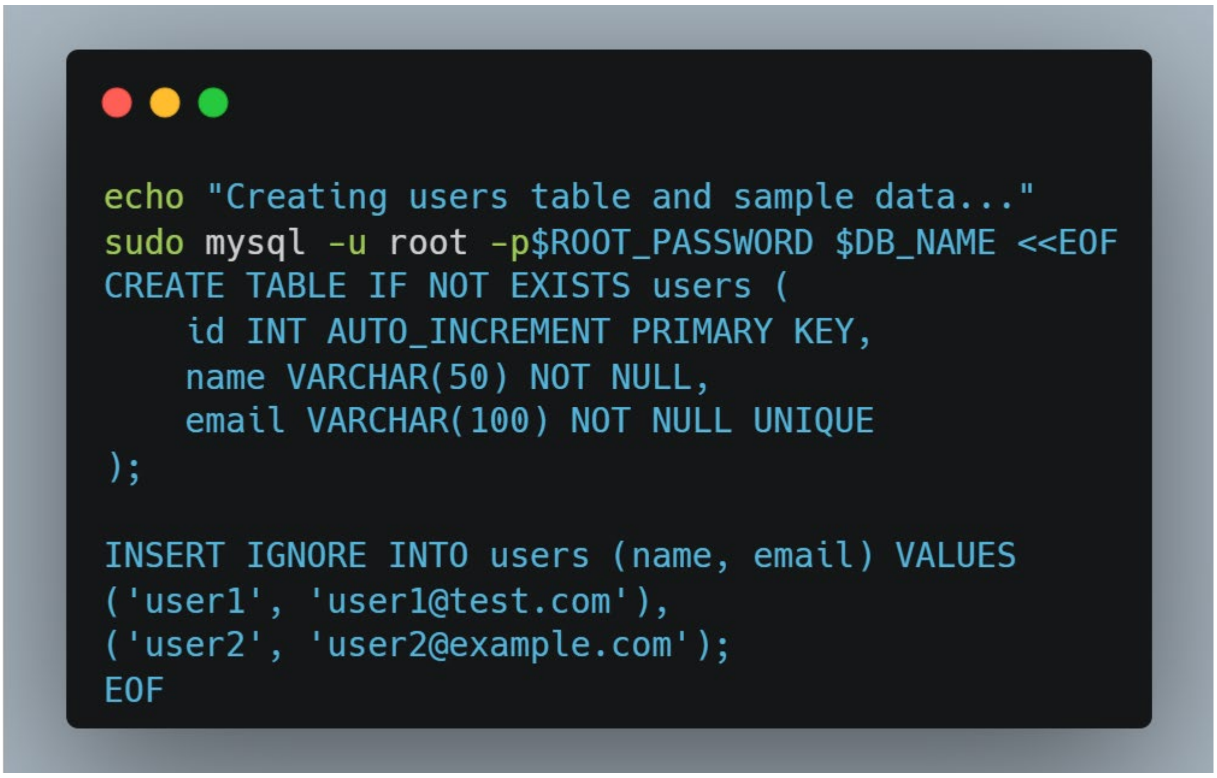


```
echo "Setting up database and user..."
sudo mysql -u root -p$ROOT_PASSWORD <<EOF
CREATE DATABASE IF NOT EXISTS $DB_NAME;
CREATE USER IF NOT EXISTS '$DB_USER'@'localhost' IDENTIFIED
WITH mysql_native_password BY '$DB_PASSWORD';
GRANT ALL PRIVILEGES ON $DB_NAME.* TO
'$DB_USER'@'localhost';
FLUSH PRIVILEGES;
EOF
```

Automated Flask Environment

This command sets up the MySQL database using environment variables from the .env file.

- **Creates the database** specified by ${DB_NAME}.
- **Creates a user** ${DB_USER} with the password ${DB_PASSWORD}.
- **Grants full privileges** on the database to that user.
- **Flushes privileges** so changes take effect immediately.

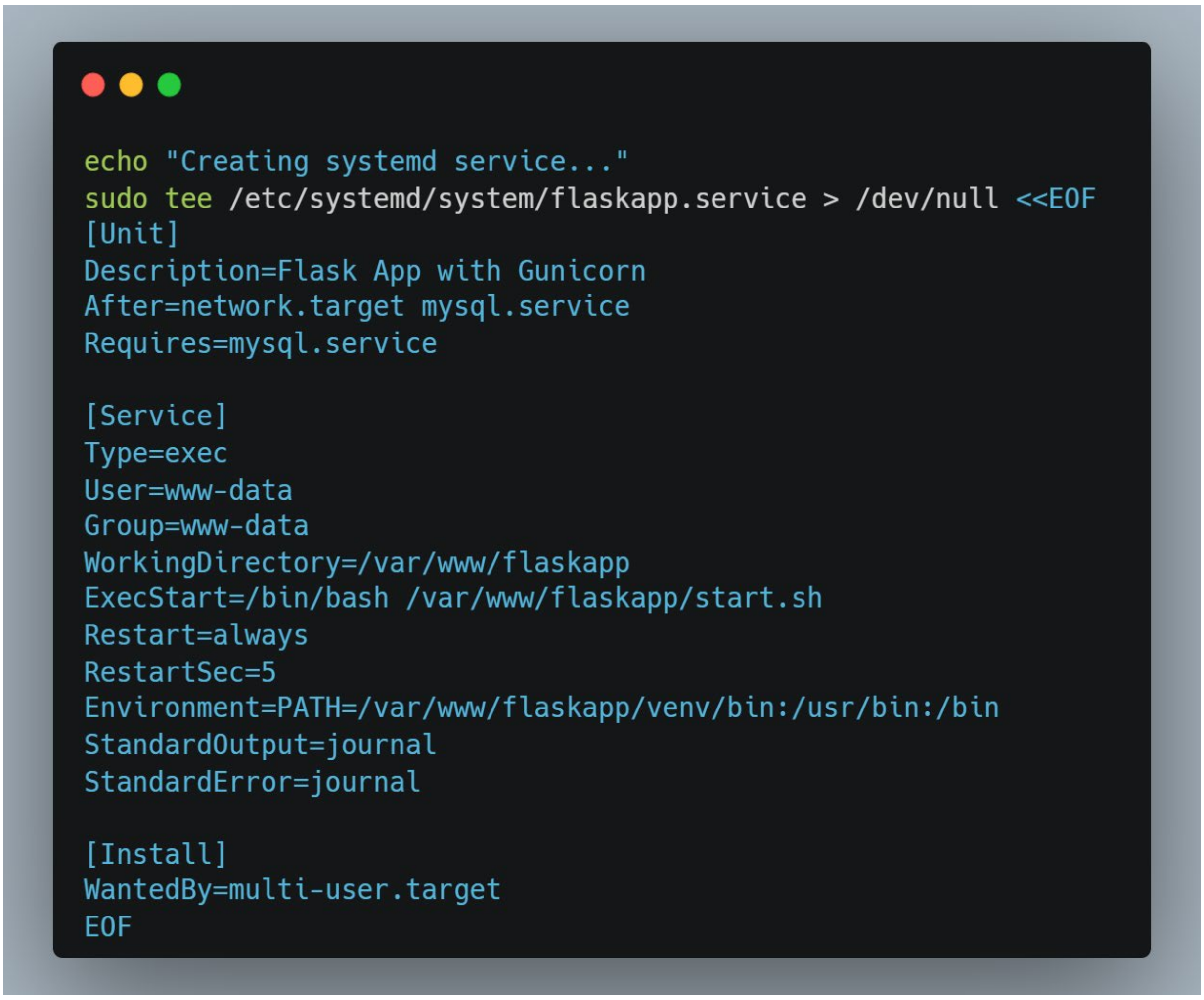


```
echo "Creating users table and sample data..."
sudo mysql -u root -p$ROOT_PASSWORD $DB_NAME <<EOF
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);

INSERT IGNORE INTO users (name, email) VALUES
('user1', 'user1@test.com'),
('user2', 'user2@example.com');
EOF
```

Connects to the flaskdb database as root using the password from the .env file.

Creates a table called users with three fields: id, name, and email and inserts two example users into the table.

Automated Flask Environment



```
echo "Creating systemd service..."
sudo tee /etc/systemd/system/flaskapp.service > /dev/null <<EOF
[Unit]
Description=Flask App with Gunicorn
After=network.target mysql.service
Requires=mysql.service

[Service]
Type=exec
User=www-data
Group=www-data
WorkingDirectory=/var/www/flaskapp
ExecStart=/bin/bash /var/www/flaskapp/start.sh
Restart=always
RestartSec=5
Environment=PATH=/var/www/flaskapp/venv/bin:/usr/bin:/bin
StandardOutput=journal
StandardError=journal

[Install]
WantedBy=multi-user.target
EOF
```

This step sets up a **Linux service** to manage the Flask application automatically using **systemd**.

A **Linux service** is a background program that starts automatically and keeps running, even when no user is logged in.

systemd is the system and service manager for most modern Linux distributions. It controls how services start, stop, restart, and interact with the system.

[Unit]: This section tells systemd when and how the service should start.

- **After=network.target mysql.service**
  - Ensures the service starts after the network and MySQL services are available.
- **Requires=mysql.service**
  - Declares a dependency on the MySQL service, if MySQL fails, this service is considered failed too.

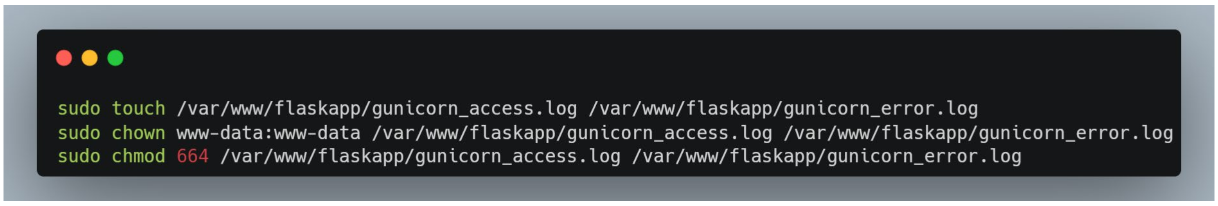
[Service]: This section contains the actual configuration for running the app.

- **Type=exec**: Runs the service as an executable.
- **User=www-data** : Runs the app under the www-data user.

- **`Group=www-data`**: Runs the app under the www-data group.
- **`ExecStart=/bin/bash /var/www/flaskapp/start.sh`**: This tells system to start the app using the "start.sh" script.
- **`Restart=always`**: Automatically restart the app if it crashes or fails.
- **`RestartSec=5`**: Wait 5 seconds before trying to restart it.
- **`WorkingDirectory=/var/www/flaskapp`**: Sets the working directory for the service.
- **`Environment=PATH=/var/www/flaskapp/venv/bin:/usr/bin:/bin`**: Sets the executable path for the service, it ensures systemd uses the virtual environment **`(venv/bin)`** and has access to essential system commands **`(/usr/bin, /bin)`**.
- **`StandardOutput / StandardError=Journal`**: Redirects logs to the system journal for centralized logging.

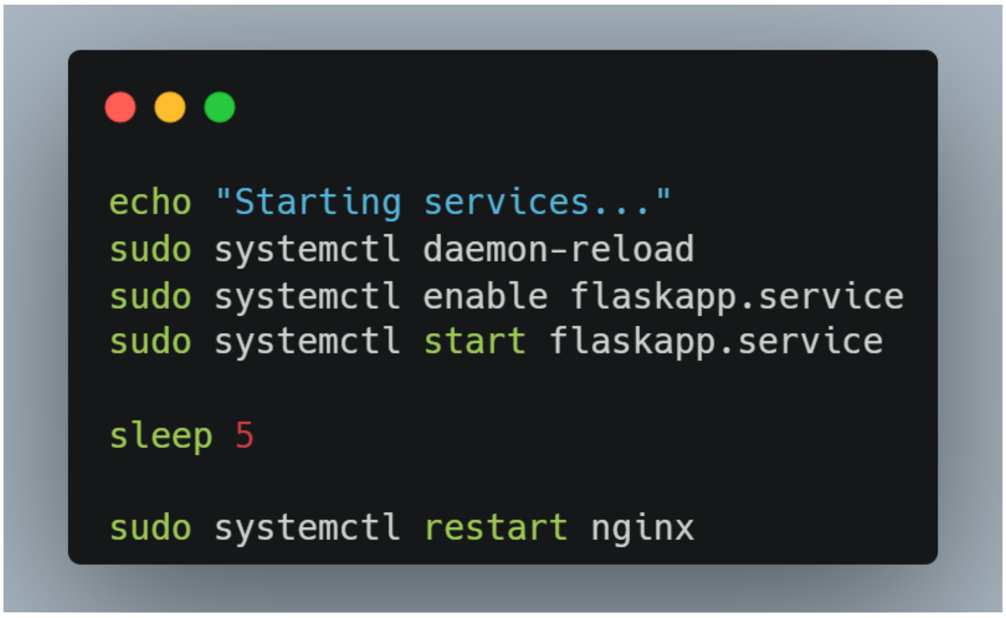
`[Install]`: This section defines when the service should be enabled.

- **`WantedBy=multi-user.target`**: This makes the app start automatically at boot time, when the server enters multi-user mode.



```
sudo touch /var/www/flaskapp/gunicorn_access.log /var/www/flaskapp/gunicorn_error.log
sudo chown www-data:www-data /var/www/flaskapp/gunicorn_access.log /var/www/flaskapp/gunicorn_error.log
sudo chmod 664 /var/www/flaskapp/gunicorn_access.log /var/www/flaskapp/gunicorn_error.log
```

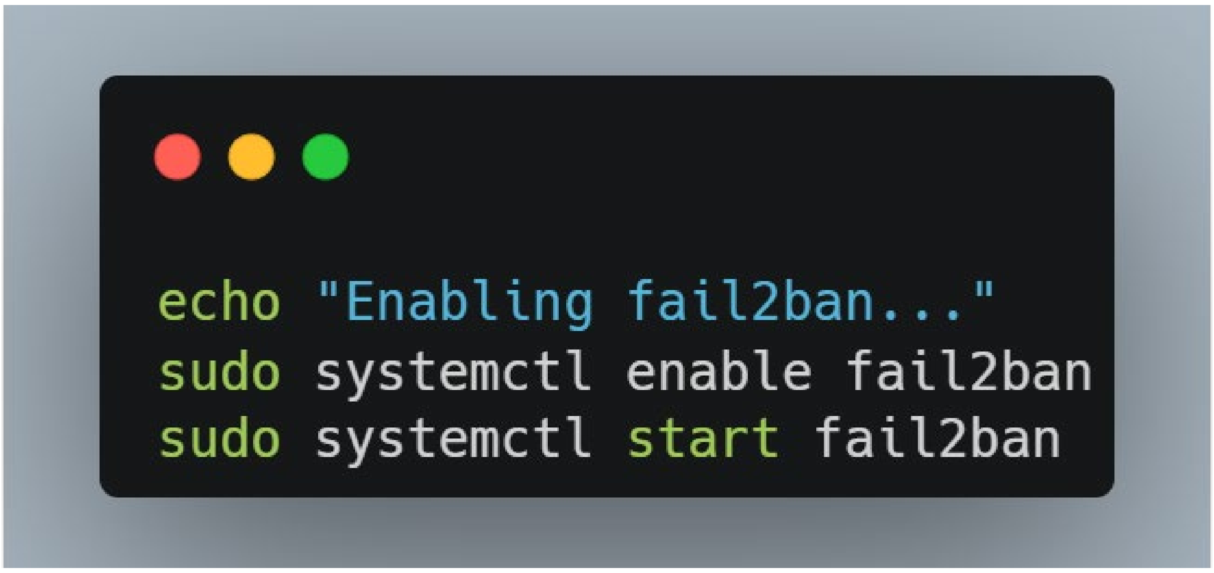
This set of commands creates the log files if they do not exist, and then sets the owner of the files to **`www-data`**, which is the user running Gunicorn, and gives write permission to the owner and read permission to others.



```
echo "Starting services..."
sudo systemctl daemon-reload
sudo systemctl enable flaskapp.service
sudo systemctl start flaskapp.service

sleep 5

sudo systemctl restart nginx
```

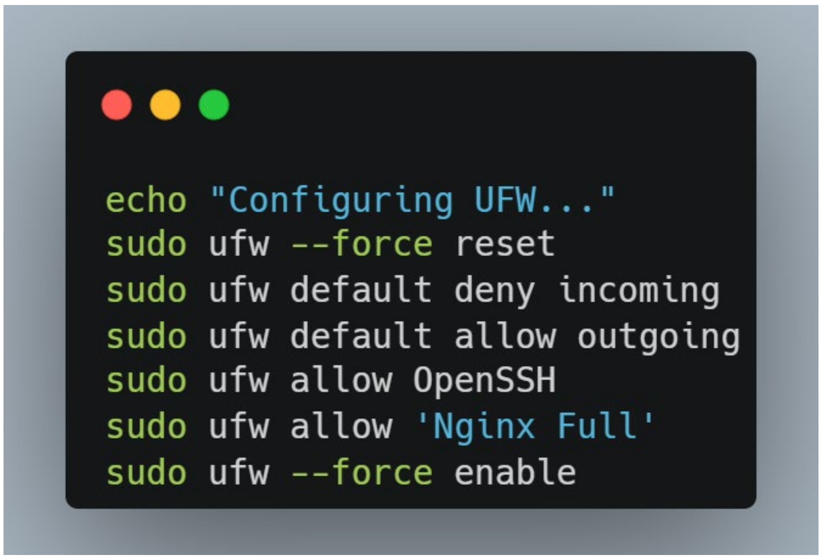
Automated Flask Environment

- **`sudo systemctl daemon-reload`** :
  - Reloads systemd to recognize the newly created **`flaskapp.service`** file.
- **`sudo systemctl enable flaskapp.service`**:
  - Enables the service to **automatically start at boot**.
- **`sudo systemctl start flaskapp.service:`**
  - **Starts** the Flask app via Gunicorn immediately.
- **`restart nginx:`**
  - Reloads Nginx to apply the reverse proxy configuration.



```
echo "Enabling fail2ban..."
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

This command ensures that **Fail2Ban** starts automatically on every boot and immediately starts it.



```
echo "Configuring UFW..."
sudo ufw --force reset
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow OpenSSH
sudo ufw allow 'Nginx Full'
sudo ufw --force enable
```

- **`sudo ufw --force reset`**
  - Resets UFW to its default state by removing all existing rules.

- **`sudo ufw default deny incoming:`**
  - Denies all incoming connections by default to protect the server from unauthorized access.
- **`sudo ufw default allow outgoing:`**
  - Allows all outgoing connections so that the server can access external resources.
- **`sudo ufw allow OpenSSH:`**
  - Explicitly allows SSH (port 22), which is essential for maintaining remote access to the server.
- **`sudo ufw allow 'Nginx Full':`**
  - Enables web traffic by allowing HTTP (port 80) and HTTPS (port 443) for the Nginx web server.
- **`sudo ufw --force enable`**
  - Activates the firewall immediately without prompting for confirmation.