LINUX VIRTUAL ENVIRONMENT SOLUTION FOR HOSTING A WEB APPLICAION

THREADING LABS

MARCO ZETINA VARGAS
SYSTEM ADMINISTRATOR

Table of Contents

Executive Summary

Threading Labs is prioritizing to have a rapid deployment of its Django Web Application, as well as emphasizing reliability and security as their critical requirements. Thus, the objective of this project solution is to setup an efficient, secure and stable production ready Linux based virtual environment to host the Threading clouds web application and its PostgreSQL database. The implemented environment is meant to ensure reliability, security, and scalability for future growth.

The proposed solution involves the use of Ubuntu LTS as the Linux distribution due to how its characteristics aligns with the companies goals for this project. This approach offers minimal downtime, secure configurations, and easiness of use for the Threading Labs' team members who are currently focused on the ongoing development of the web application. Apart from that, since the company is considering moving to a cloud provider, the cost effectiveness that Ubuntu LTS offers, supports and encourages the idea of moving to a cloud provider in the future by offering a budget friendly alternative.

Key components of the solution include:

- Logical Volume Management, for scalability.
- Automated backups and monitoring, to prevent data loss and any downtime.
- Configured firewall and network security, to maintain the application safe from treats.

By implementing this solution, Threading Labs can confidently launch its platform to serve their clients and staff as quick as possible, ensuring is both, stability and security. Apart from that, it leaves the door open for potential growth for the future in terms of user requests or data increase as well as possible cloud adoption.

Linux Distribution Comparison

There is a significant amount of Linux distribution in the current market that are ideal for different use cases, such as cybersecurity, software development and production environments. For this project, different Linux distributions where evaluated making emphasis in reliability, security, and ease of management as well as their use for a production environment.

Some Linux distributions, although highly valued in the industry since the excel in their own fields, where excluded from the selection, because they are not the most ideal options to host a production ready environment.

Excluded Distributions

Distributions like Kali Linux and Parrot Os that are specially designed for ethical hacking and penetration testing, make unnecessary overhead to host the web application, and since their main focus is cybersecurity operations, they lack the stability and long term support needed for this project. Every lightweight or minimalistic distribution type was also excluded due to their lack of robustness and scalability for this project, along with community oriented, rolling release and experimental distributions that inherently do not align with the project objectives.

Selected Distributions for Further Evaluation

1. **Ubuntu LTS**
   This Linux distribution prioritizes long term stability and security with extended support cycles of five years. It is widely used in production environments, it contains robust default repositories, an active community and a bast documentation.

   It has an excellent compatibility with software like Nginx and PostreSQL. And it offers a large ecosystem for cloud integration and containerization, making it ideal when dealing with DevOps tools. For some particular projects this distribution may lack the in depth customization that other distributions can offer.

2. **Debian**

   It provides a solid stability and reliability, it serves as the foundation for Ubuntu. Moreover, it focuses on long term support, and it is also suitable for production environments. It provides an extensive community support and it gives more control over the system with just a minimal amount of default configurations.
   Debian might be a good option, however it has less focus on user experience, so this would mean that a more manual setup would be required, and for teams with a bit less expertise on Linux might be a drawback.

3. **Red Hat Enterprise Linux**

   RHEL is a commercial distribution specifically designed for enterprise environments. It offers long term stability, it makes focus on advanced security features and provides professional support. It is designed for production and enterprise scale project deployments, which makes it an excellent choice for organizations that require enterprise grade dedicated support for their projects.

   RHEL's commercial license might not fit all budgets, since it is heavily oriented for an enterprise use, and its limited community support may the troubleshooting done by the team members harder.

4. **CentOs Stream**

   It is a strong candidate for environments that require stability, but also access to some of the RHEL features. It provides a balance between stability and innovation. It has regular updates to provide newer software without necessarily sacrificing stability. It bridges the gap between having the latest software and being production ready at the same time.

   A drawback that was taken into consideration is that CentOs serves as a testing ground for the RHEL distribution, making it prone to be more instable with the addition of the latest features. This regular updates might introduce conflicts that have not been fully tested yet for a production environment as the project demands. It also has a smaller community compared to other distributions like Ubuntu.

5. **Fedora**

   This Linux distribution its known for its rapid innovation, so it is ideal for environments that precise the latest software available. It is always up to date, so it is mostly suitable for projects that prioritize up to date technologies, and are also comfortable with frequent updates. It provides a strong focus on developers, making it a great platform for modern applications.

However, Fedora's short lifecycle, might force the team members to upgrade the system frequently, which can potentially disrupt the environment, and for a project that requires stability and long term quality as this one, the constant upgrades might increase the probability of introducing bugs or challenges.

6. **Amazon Linux**

Amazon Linux is a distribution optimized for use with AWS environments, it is meant to provide the most seamless integration with AWS services, it has preconfigured AWS tools and it is perfect for a project hosted on AWS, offering great cost efficiency, enhancements and reduced time set up with Amazon Web Services.

Although this project may consider to implement cloud services for the future, for now it would not be the most practical choice. Amazon Linux is mostly limited for cloud environments that look to be integrated with AWS. For the current project that means that Amazon Linux would make things more complicated when trying to access tools or packages outside AWS repositories, and the level of compatibility with the components of the architecture later proposed might be affected.

Criteria	Ubuntu LTS	CentOS Stream	Fedora	Debian	Amazon Linux	Red Hat (RHEL)
Stability	High	Medium	Medium	High	Medium	High
Security	High	High	Medium	High	High	High
Ease of Use	High	Medium	Medium	Low	Medium	Medium
Community Support	High	Medium	High	High	Medium	Medium
Update Frequency	Medium	Medium	High	Low	Medium	Medium
Enterprise Support	Medium	High	Low	Medium	Medium	High
Cloud Readiness	High	Medium	Medium	Low	High	High
Scalability	High	High	Medium	Medium	High	High
Cost	Free (Community)	Free (Community)	Free (Community)	Free (Community)	Free (AWS-exclusive)	Paid Subscription

Reason of Choosing Ubuntu LTS

After an evaluation of the different Linux distributions available in the market, Ubuntu LTS was selected as the most suitable option for the solution of this project. Below there are the key reasons of why such choice.

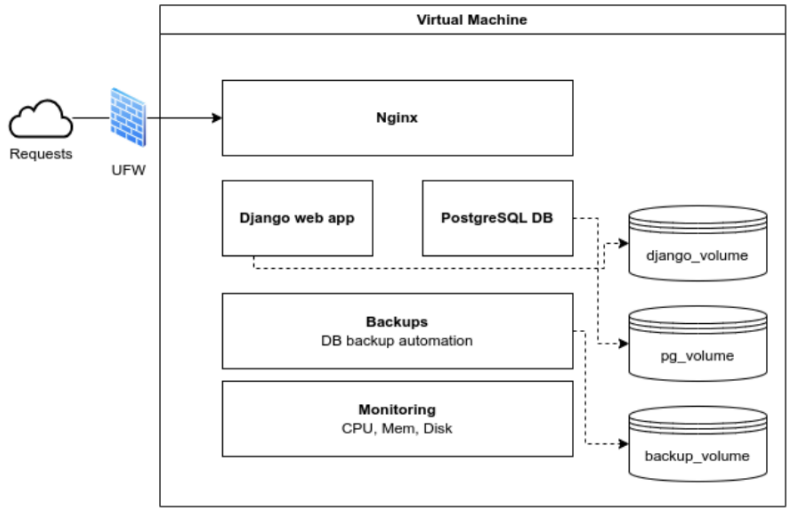
Ubuntu LTS was chosen as the hosting environment for the web application due to its closest alignment with the goals for this project, since it offers the perfect balance between stability, security, and also usability. Apart from that, its long term support ensures a reliable system, which is critical for a production environment. It also has an extensive package repository, and user friendliness in the setup simplifying installation of tools like Nginx, PostgreSQL and UFW, and, although the setup could be with no problem also accomplished in other Linux

distributions, by the nature of this project Ubuntu offers the most practical and efficient choice for this particular project due not only its user friendly characteristics, combined with an extensive documentation and community support, and also alignment with modern DevOps practices, making it the option that stands out due to their mixed benefit points for this specific project.

Architecture

The architecture incorporates several components for application hosting, database management, data backups and monitoring. Each component is carefully configured in order to provide a reliable production environment.

Architecture Diagram



Components

1. Virtual Machine

- **Purpose:**

The virtual machine is the base of the infrastructure. It provides an isolated and secure environment to host the web application, database and the other supporting tools. It ensures efficiency in the resource management, scalability, and reliability for the production environment.

- **Role in the architecture:**

It is in charge of providing the Linux environment, Ubuntu LTS, to run all the components safely and efficiently. It ensures that the system is independent from other environments, so external interference is minimized. It allocates the CPU, memory and disk resources to provide consistent performance for the rest of the components, and also gives easiness to a future migration to a cloud platform without significant amounts of reconfiguration.

2. Nginx

- Purpose:

Light weight, high performance web server and reverse proxy responsible for handling incoming http/https requests and routing them to the web application.

- **Role in the architecture:**

Act as a reverse proxy to forward incoming client requests to the Django application, and also serves the static files for it. It also secures connections by enabling TLS and encrypting the traffic so that the user requests are protected against interception or modification.

3. Django Web Application

- **Purpose:**

The Django web application is the core of the project, it is responsible for providing CRUD management functionalities for the videogames catalog.

- **Role in the architecture:**

Handles user interactions as well as business logic, and communicates with the PostgreSQL database to retrieve the stored data, it runs on a dedicated django_volume for isolation and better storage management.

4. PostgreSQL Database

- **Purpose:**

Store all the videogame data managed by the web application.

- **Role in the architecture:**

Stores the videogame data: Titles, genres, release dates and descriptions. It uses a pg_volume to separate data from other system files to improve security and performance.

5. **Backups**

- **Purpose:**

Automated backups are meant to ensure data integrity and recovery in case of disasters.

- **Role in the architecture:**

Store the backups of the PostgreSQL database and application files in a dedicated backup_volume. It minimizes downtime in case of failures.

6. **Monitoring**

- **Purpose:**

Tracking system performance and usage of resources, in this case memory, CPU performance and disk space.

- **Role in the architecture:**

In case of potential problems, like high resource usage it detects the potential issues in the environment and alerts about them so that the problem can be fixed in an early stage.

7. **UFW**

- **Purpose:**

The Uncomplicated Firewall Network is used to secure the virtual machine by controlling the incoming and outgoing network traffic.

- **Role in the architecture:**

It ensures only the necessary ports for connection and restricts the network access which is unauthorized protecting the application form external threats.

8. **Logical Volume Management**

- **Purpose:**

It is used to provide flexibility to the disk management in order to accommodate data growth in an ordered manner.

- **Role in the architecture:**

It allocates the logical volumes for a better organization of the storage management, and allows resizing of storage without disrupting the ongoing services.

How the Components Work Together

Incoming Requests enter the system by the UFW, which is mean to ensure that only legitimate traffic is allowed, by restricting access to only specific ports for HTTP and HTTPS. Once validated, Nginx is in charge of processing this requests, and forwarding to the web application.

The Django web application handles the user interaction and requests so that when data is required, it interacts with the PostgreSQL database. The database is isolated in its own pg_volume, ensuring efficiency in the performance and better security. Automated backups are scheduled at a certain time of the day to provide recoveries. Backups include the application files from the django_volume, as well as the database files form the pg_volume. These backups are stored in another dedicated backup_volume, for an easy recovery in case of system failures.

Regarding the monitoring, the system monitoring tools have alerts configured to detect potential issues to encourage their fast resolution.

In terms of scalability, the LVM makes the architecture designed for future growth. Either if user demands or data volume increases, the storage volumes can be resized dynamically without disrupting the system.

Reason of The Chosen Architecture

This architecture establishes security by controlling invalid access, while Nginx acts as a reverse proxy to filter the malicious requests. It has great level of reliability, since the components are isolated and the possibility of having issues does not mean having them everywhere but only in their own isolated environment. The automated backups protect the system from disasters. Moreover, the architecture provides scalability and performance metrics, and management easiness where manual effort for backups is reduced and logical volumes also simplify the disk management.

Implementation

○ Set up the VM and Ubuntu LTS

For the virtual machine, use VirtualBox. install it from the official website[1] and after setting up the installation, allocate at least 2 CPU cores, 4GB of ram, and at least 20 GB of storage in order for the VM to work correctly. After that, download Ubuntu LTS[2] and in the virtual machine add a new configuration with the path of the previously downloaded Ubuntu image and provide a username and password for further installation steps.

○ Set up the Firewall with UFW

Install UFW with the following command:

sudo apt update

sudo apt install ufw -y

First of all is necessary to install UFW if necessary (although Ubuntu LTS already has to have it pre-installed but the above command will confirm it).

After that, it will be necessary to allow the SSH traffic before enabling the firewall. For that use:

sudo ufw allow OpenSSH

After that,

sudo ufw allow 'Nginx Full' to save the rule for later steps when configuring Nginx

in order to allow http and https for web traffic. This command will allow traffic on ports 80 for HTTP and 443 for HTTPS.

---

[1] https://www.virtualbox.org/
[2] https://ubuntu.com/download

The next step is to deny all the other possible traffic by default, this must be done by following the commands:

sudo ufw default deny incoming

sudo ufw default allow outgoing

Lastly, after having defined the rules, enable the firewall:

sudo ufw enable

and then, check the list of active rules to confirm that the configuration is correct, this command will show a list of the enabled features:
sudo ufw status verbose

so the enabled features should appear as '**ALLOW**'.

See the official Ubuntu documentation about UFW for even more details about other commands[3] and firewalls[4].

○ Create and Configure User Accounts with limited privileges

To ensure secure management of system resources and provide isolation to each component the following users should be created:

[3] https://help.ubuntu.com/community/UFW
[4] https://ubuntu.com/server/docs/firewalls

User	Component	Purpose
django_user	Django Application	Runs the Django web application.
postgres	PostgreSQL Database	Administers the database service.
db_user	Django's DB User	Handles queries from the Django application.
backup_user	Backup Service	Runs scripts to back up application and data.
monitoring_user	Monitoring Tools	Monitors system performance and health.

In order to create those users, write in the terminal the following commands[5]:

sudo adduser django_user

sudo adduser db_user

sudo adduser backup_user

sudo adduser monitoring_user

After that it will be necessary to change the ownership using the chmod command to assign all files and directories under the specified path, to the specified users and groups. Also it is necessary to create a backup directory and assign the ownership to the backup user.

sudo chown -R django_user:django_user /var/www/django_app

[5] https://ubuntu.com/server/docs/user-management

sudo chown -R postgres:postgres /var/lib/postgresql

sudo chown -R monitoring_user:monitoring_user /var/monitoring_tools

sudo mkdir /var/backups && sudo chown -R backup_user:backup_user /var/backups

Now, for setting permissions the command ***sudo chmod -R 700,*** where the 700 permission means that only the owner of the directory has permission to read write and execute and other users will have no access to the specified directory. and -R means it is a recursive command, so instead of applying permissions to each file in the directory, -R automates the process, although we must be careful when dealing with critical directories[6].

sudo chmod -R 700 /var/monitoring_tools

sudo chmod -R 700 /var/www/django_app

sudo chmod -R 700 /var/backups

This command should not be applied to postgreSQL's directory since in the installation process it will already have automatically enabled restrictive access, so there is no need to do it again.

Lastly, it will be necessary to disable the log in for each of the users, so that the vulnerability of having a brute force attack is reduced, as well as unauthorized attempts of login in, so the environment remains secure.

- Install and Configure the PostgreSQL database and connect Django

First, install PosgreSQL[7]:
sudo apt install postgresql

[6] https://manpages.ubuntu.com/manpages/trusty/man1/chmod.1.html

[7] https://ubuntu.com/server/docs/install-and-configure-postgresql

By default only the connections from the local system are allowed[8] which is a good thing, because it increases the security of our environment.

After that, enable PosgreSQL to run on boot, for that use:
sudo systemctl start postgresql

sudo systemctl enable postgresql

After that, switch to the posgre user and access the PostgreSQL interactive terminal:
sudo -i -u postgres

And then,

Psql

After that, create a new database and also a new user, name it postgres to have the same credentials as in the previous project implementation and grant privileges to that user.

Once PostgreSQL is ready, we must get the Django web application by simply cloning the repository from GitHub to have it on the VM.

So run the command:

git clone https://github.com/Marrrco7/Django-web-Application-with-CRUD-operations.git /var/www/django_app

Important: since we transitioned from the previous local environment, it will be necessary to recreate the credentials.env file containting the credentials that the django application used to connect to the database and that it is used in order to not hardcode the credentials directly in the settings.py of the django application. This file is not contained in the cloned repository because it was excluded in gitignore. Therefore it must be created again.

Go to the directory of the django application

cd /var/www/django_app

[8] https://ubuntu.com/server/docs/install-and-configure-postgresql#configure-postgresql

Create the credentials.env file again:
nano credentials.env

And add again the credentials of the database as it was previously done.

Then, install python and the venv module[9]:

sudo apt install python3 python3-pip

sudo apt install python3-venv

Go to the django service directory:
cd /var/www/django_app

Create and activate the virtual environment:

python3 -m venv venv

source venv/bin/activate

After that install the list of dependencies specified in the requirements.txt file of the django project and also verified that Django is installed:

pip install -r requirements.txt

After that, it is of course necessary to apply the migrations to see the changes in the database.

python manage.py migrate

And also, in order to test that the app is running install the waitress server, and after that run the server to test the app[10]:

pip install waitress

---

[9] https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/#create-and-use-virtual-environments

[10] https://pypi.org/project/waitress/

python run_waitress_server.py

- Configure the volumes for the Web Application and PostgreSQL

To do that, create the following volume directories using the following commands:

For Django:

sudo mkdir -p /var/lib/postgresql/pg_volume

sudo chown postgres:postgres /var/lib/postgresql/pg_volume

sudo chmod 700 /var/lib/postgresql/pg_volume

For PostgreSQL:

sudo mkdir -p /var/www/django_app/django_volume

sudo chown django_user:django_user /var/www/django_app/django_volume

sudo chmod 700 /var/www/django_app/django_volume

After that, add the entries for mounting the volumes.

Then, set the PostgreSQL configuration file to point to the pg_volume, so:

data_directory = '/var/lib/postgresql/pg_volume'

Similarly with Django, it is necessary to update the settings.py to use the django volume

- Set the Automated Back Up

Set the automated back up

For that, we need to create a backup directory and use the previously configured backup volume for storing the backups:

sudo mkdir -p /var/backups/postgres

sudo chown backup_user:backup_user /var/backups/postgres

sudo chmod 700 /var/backups/postgres

Then, it is necessary to set a backup script to back up the PostgreSQL database

sudo nano /var/backups/postgres/backup_postgres.sh

and then add the following content into that file:

```
#!/bin/bash

TIMESTAMP=$(date +"%F_%H-%M-%S")

BACKUP_DIR="/var/backups/postgres"

DATABASE_NAME="your_database_name"

USER="backup_user"


export PGPASSWORD=your_backup_user_password


pg_dump -U $USER $DATABASE_NAME >
"$BACKUP_DIR/$DATABASE_NAME_$TIMESTAMP.sql"
```

Save an exit the script with crt + O and ctr + X.

After that, the script must be set as executable:
sudo chmod +x /var/backups/postgres/backup_postgres.sh

And lastly, we need to schedule the backup using cron[11], so for that:

sudo crontab -e

[11] https://help.ubuntu.com/community/CronHowto

0 2 * * * /var/backups/postgres/backup_postgres.sh

Where the 0 2 means 2 am, so that means that the backup would be automatically scheduled at that time, in our example.

Do similarly for the django files, but with the difference of providing this script for the backup:

#!/bin/bash

TIMESTAMP=$(date +"%F_%H-%M-%S")

BACKUP_DIR="/var/backups/django"

SOURCE_DIR="/var/www/django_app/django_volume/media"

tar -czf "$BACKUP_DIR/media_backup_$TIMESTAMP.tar.gz" $SOURCE_DIR

Save an exit the script with crt + O and ctr + X.

- Setting and Configuring Nginx

Similarly as with previous steps, install Nginx[12]:

sudo apt update

sudo apt install nginx

Then go to the already available sites-available directory:

cd /etc/nginx/sites-available

---

[12] https://ubuntu.com/tutorials/install-and-configure-nginx#1-overview

Create a configuration file:

sudo nano django_app

And write the following script:

```
server {
   listen 80;
   server_name 127.0.0.1;

   location /static/ {
      alias /var/www/django_app/static/;
   }

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;
      proxy_set_header X-Forwarded-Proto $scheme;
   }
}
```

Save an exit the script with crt + O and ctr + X.

Then link this configuration file to the sites-enabled directory to activate it:

sudo ln -s /etc/nginx/sites-available/django_app /etc/nginx/sites-enabled/

Lastlly, restart Nginx:
sudo systemctl restart nginx

And confirm the setup by accessing in the browser:

http://127.0.0.1/videogame

○ Setup and Configure Netdata

Install netdata[13]:
curl https://get.netdata.cloud/kickstart.sh > /tmp/netdata-kickstart.sh && sh/tmp/netdata-kickstart.sh

This script will automatically start the Netdata service and also configure it to start at boot.

To check a successful installation:

sudo systemctl status netdata

Also, it will be necessary to allow traffic on port 19999, since there is there Netdata runs, so run:

sudo ufw allow 19999

sudo ufw reload

Since by default the netdata's dashboard is accessible for anyone that knows the server IP and port, for security reasons is necessary to restrict its access.

create a new Nginx configuration and add the script:

[13] https://learn.netdata.cloud/docs/netdata-agent/installation/linux/

sudo nano /etc/nginx/sites-available/netdata

```
server {
   listen 80;
   server_name 127.0.0.1;

   location /netdata/ {
      proxy_pass http://127.0.0.1:19999/;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header X-Forwarded-Proto $scheme;
      proxy_http_version 1.1;
      proxy_request_buffering off;
      proxy_buffering off;
   }
}
```

Activate the new configuration file and restart Ngnix to apply the changes:

sudo ln -s /etc/nginx/sites-available/netdata /etc/nginx/sites-enabled/

sudo systemctl restart nginx

Finally access the netdata's dashboard:

http://127.0.0.1:19999

And once there it will be possible to see the metrics of memory, CPU and disks each in their own section.

Troubleshooting, Maintenance and Monitoring Guide

Common Issues and Solutions

- **Nginx is not starting:**

Run the command in order to test the configuration for any errors, if any error is found, it will be possible to get information from the messages:

sudo nginx -t

- **Error in the connection with the database:**

Be sure to have recreated the credentials.env file correctly and with the same credentials as in the previous implementation in the remote repository, the settings.py already have loaded the credentials file. Also check that the firewall settings don't disallow the connection.

It is also possible to check the status of the posgreSQL database by running:

sudo systemctl status postgresql

And test the connection to the database by:

psql -U postgres -d videogames_db

- **Static files are not loading:**

Ensure that the directory has the proper permissions, and that also points to the correct path, try to run

sudo chown -R www-data:www-data /var/www/django_app/static/

sudo chmod -R 755 /var/www/django_app/static/

and in case that the static files need to be restored in the remote repository, run ***python manage.py collectstatic***

to create them again, and they will appear in a new folder staticfiles/

- **Not possible to access Netdata**

In case that there is an issue when accessing ***http://127.0.0.1:19999*** check if the status of the UFW to check what ports are allowed, in case that the port 1999 is disallowed, it will be necessary to enable it manually:

sudo ufw allow 19999

And try to access it again, also ensure the correct installation of Netdata.

It is possible to check if Netdata is running by writing the command:
sudo systemctl status netdata

Maintenance and Monitoring

- Check periodically for updates in the system, to check that it stays secure and up to date.

- Regularly check the backups to be sure that they are running as intended, and also test restoration from a backup from time to time, at least once a month.

- It is also possible to review applications logs in case of unusual activity, for that run the following commands:
  sudo tail -f /var/log/nginx/access.log
  sudo tail -f /var/log/nginx/error.log

- Regularly track the Netdata's dashboard ***http://127.0.0.1:19999*** to get information about the resource usage and to check that they are not being overused.

References

Ubuntu. (2019). Get Ubuntu | Download | Ubuntu. [online] Available at: https://ubuntu.com/download

Oracle (2023). Oracle VM VirtualBox. [online] VirtualBox. Available at: https://www.virtualbox.org/

help.ubuntu.com. (n.d.). *Official Ubuntu Documentation*. [online] Available at: https://help.ubuntu.com/?_gl=1

www.postgresql.org. (n.d.). *PostgreSQL: Linux downloads (Ubuntu)*. [online] Available at: https://www.postgresql.org/download/linux/ubuntu/

Ubuntu. (2024). Install and configure PostgreSQL | Ubuntu. [online] Available at: https://ubuntu.com/server/docs/install-and-configure-postgresql#configure-postgresql

Ubuntu. (n.d.). User management. [online] Available at: https://ubuntu.com/server/docs/user-management

Ubuntu.com. (2024). Available at: https://ubuntu.com/server/docs/firewalls.

Ubuntu. (n.d.). *Install and configure Nginx*. [online] Available at: https://ubuntu.com/tutorials/install-and-configure-nginx#1-overview

help.ubuntu.com. (n.d.). *UFW - Community Help Wiki*. [online] Available at: https://help.ubuntu.com/community/UFW

PyPI. (2024). *waitress*. [online] Available at: https://pypi.org/project/waitress/

help.ubuntu.com. (n.d.). *CronHowto - Community Help Wiki*. [online] Available at: https://help.ubuntu.com/community/CronHowto

packaging.python.org. (n.d.). *Install packages in a virtual environment using pip and venv - Python Packaging User Guide*. [online] Available at: https://packaging.python.org/en/latest/guides/installing-using-pip-and-virtual-environments/#create-and-use-virtual-environments

learn.netdata.cloud. (2024). *Install Netdata with kickstart.sh | Learn Netdata*. [online] Available at: https://learn.netdata.cloud/docs/netdata-agent/installation/linux/