

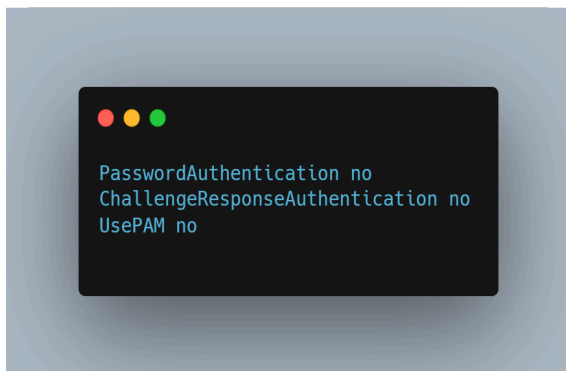
Cloud-Based Server Setup Documentation

Project Overview

This project involves setting up a secure cloud-based server for a startup, ensuring the server has strong authentication measures, firewall configuration, Fail2Ban protection, and the Apache2 web server installed. The server is configured to allow users to log in using SSH keys, with password-based authentication disabled. The server is also protected by a firewall, allowing only necessary ports.

Setting Up SSH Key Authentication

1. Disable Password Authentication:
`/etc/ssh/sshd_config`



2. Restart the SSH service to apply changes

Create a New User and Add SSH Key

1. Create a User:



2. SSH Key



Private key (`id_rsa`): This is stored on your machine and should be kept secret.

Public key (`id_rsa.pub`): This is the key you can share with the remote server you want to access.

To access a server using your SSH key, you need to copy the public key (`id_rsa.pub`) to the server's `~/.ssh/authorized_keys` file.

Configuring the Firewall

1. Install UFW
2. **Allow Only Necessary Ports:** Allow only ports 21, 22, and 80:



Configure AWS Firewall Rules

Set Up Security Group: In the AWS management console, create a security group that only allows:

- Port 21 (FTP)
- Port 22 (SSH)
- Port 80 (HTTP)

Inbound rules Info					
Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	
sgr-0c2385f21a0ff186a	Custom TCP ▼	TCP	21	Custom ▼	Q 0.0.0.0/0 ✕
sgr-0dc2892b07e5928c2	HTTP ▼	TCP	80	Custom ▼	Q 0.0.0.0/0 ✕
sgr-0e3e6c54168852357	SSH ▼	TCP	22	Custom ▼	Q 0.0.0.0/0 ✕
Add rule					

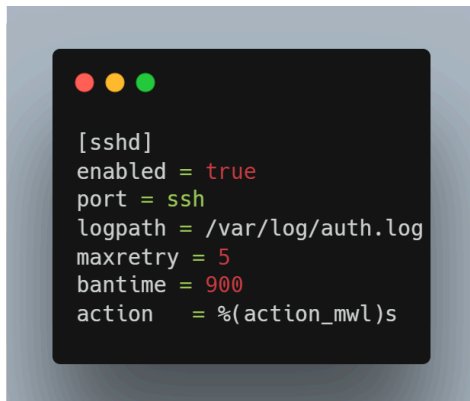
Configuring Fail2Ban

Fail2Ban is a tool that prevents brute-force login attacks by blocking IPs after a set number of failed login attempts.

- **Install Fail2Ban**

```
sudo apt install fail2ban
```
- **Configure SSH Protection: Open the Fail2Ban configuration file**

```
sudo nano /etc/fail2ban/jail.local
```
- **Set the Configuration: Add the following configuration:**



- **Restart Fail2Ban: Restart the Fail2Ban service**

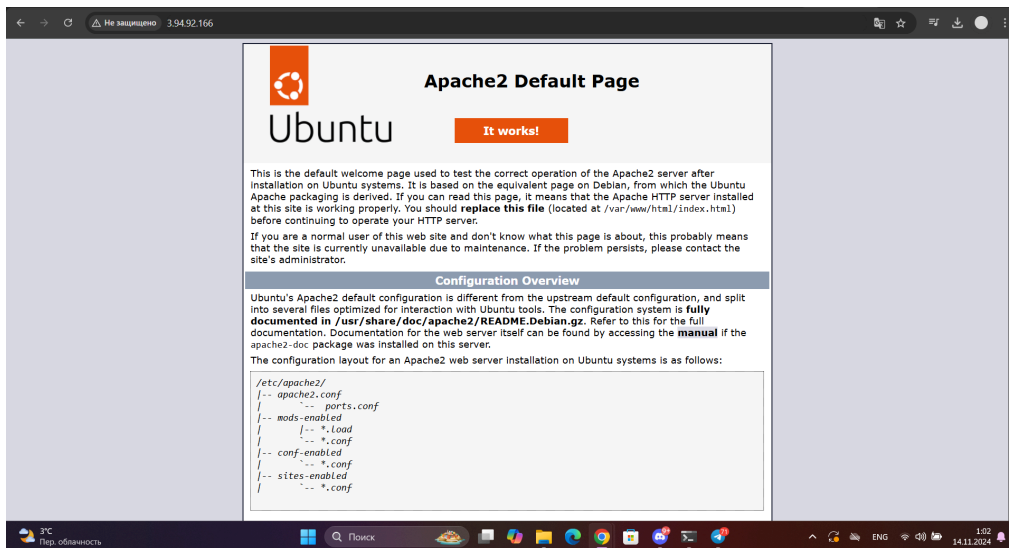
Installing Apache2 Web Server

- **Install Apache2**

```
sudo apt update
```

```
sudo apt install apache2
```

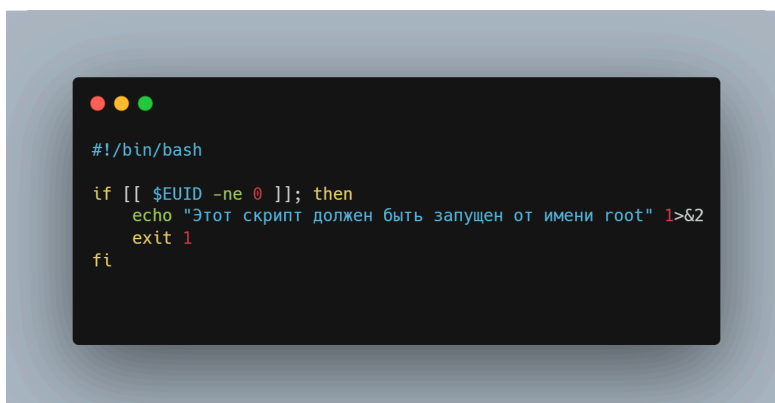
- **Verify Apache2 is Running:** Open a browser and navigate to the server's IP address to confirm Apache2 is serving the default page.



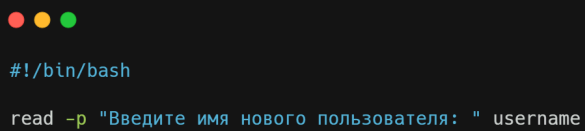
Automated User Setup Script

Bash script that automatically adds a new user, generates an SSH key pair for them, and configures SSH access.

1. We check if the user has root privileges or if it uses sudo.

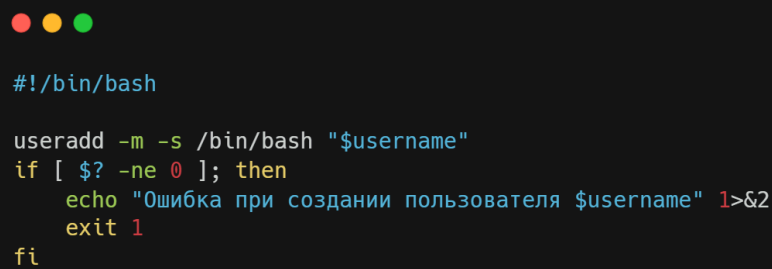


2. The settings of the new user.



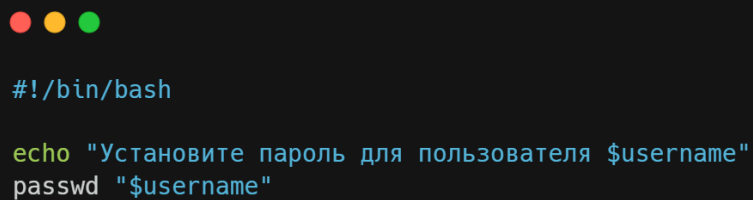
```
#!/bin/bash  
read -p "Введите имя нового пользователя: " username
```

3. Creation of the new user.



```
#!/bin/bash  
  
useradd -m -s /bin/bash "$username"  
if [ $? -ne 0 ]; then  
    echo "Ошибка при создании пользователя $username" 1>&2  
    exit 1  
fi
```

4. Setting the password.



```
#!/bin/bash  
  
echo "Установите пароль для пользователя $username"  
passwd "$username"
```

5. Generating ssh key.

```
#!/bin/bash

if [ ! -f "/home/$username/.ssh/id_rsa" ]; then
    echo "Генерация SSH-ключей для пользователя $username"
    sudo -u "$username" ssh-keygen -t rsa -b 4096 -f "/home/$username/.ssh/id_rsa" -N ""
else
    echo "SSH-ключи уже существуют для пользователя $username"
fi
```

6. We check if the .ssh folder exists or not.

```
#!/bin/bash

mkdir -p "/home/$username/.ssh"
chown "$username":"$username" "/home/$username/.ssh"
chmod 700 "/home/$username/.ssh"
```

7. We add the public key into authorized_keys.

```
#!/bin/bash

cat "/home/$username/.ssh/id_rsa.pub" >> "/home/$username/.ssh/authorized_keys"
chown "$username":"$username" "/home/$username/.ssh/authorized_keys"
chmod 600 "/home/$username/.ssh/authorized_keys"

echo "Настройка пользователя $username завершена."
echo "Пользователь $username теперь может использовать SSH-ключи для аутентификации."
```

Security Notification Script

Bash script that monitors system logs for suspicious activity and sends an email alert in case of potential breaches or security incidents.

1. Mail configurations
2. We check if there are no blocked IP files, create one
3. We use the logs of fail2ban for the monitoring
4. We look for lines where the word "Ban" is present
5. Separate the blocked IP's
6. We check if it was banned previously
7. We check if the IP is not in the file, we send it to the specified mail
8. We send it by ssmtp
9. We add a new file for blocked IPs in order to not have them in the list of new messages

```
#!/bin/bash

# maili confignery
TO="s001kh001@gmail.com"
FROM="hommeehome04@gmail.com"
SUBJECT="Fail2Ban Blocked IP Notification"
LOG_FILE="/var/log/fail2ban.log"
LAST_BLOCKED_IPS="/tmp/last_blocked_ips"

#stugum enq ete blokavorac ipineri file chka, bacuma
if [ ! -f "$LAST_BLOCKED_IPS" ]; then
    touch "$LAST_BLOCKED_IPS"
fi

#ogtagorcum enq tali fail2ban i logery monitoring anelu hamar
tail -Fn0 $LOG_FILE | \
while read line ; do
    # pntrum enq toxer vortex ka ays bary "Ban" (т.е. блокировку IP)
    if [[ "$line" =~ "Ban" ]]; then
        # arandznacnum enq blokavorvac IP nery
        IP=$(echo $line | awk '{print $NF}')

        # stugum enq naxkinum ban exela ip in
        if ! grep -q "$IP" "$LAST_BLOCKED_IPS"; then
            # ete ipin chka fileum, uxxarkuma mailin
            EMAIL_BODY="The following IP address has been blocked by Fail2Ban:\n\n$IP"

            # uxxarkum enq ssmtp ov
            echo -e "Subject:$SUBJECT\nFrom:$FROM\nTo:$TO\n\n$EMAIL_BODY" | ssmtp $TO

            #avelacnum enq blokavorvac ipineri file, vorpeszi nor namakneri mej chlini
            echo "$IP" >> "$LAST_BLOCKED_IPS"
        fi
    fi
done
```

Install SSMTP service

- `sudo apt install ssmtp`
- open this file for configurations for using Gmail
`sudo nano /etc/ssmtp/ssmtp.conf`

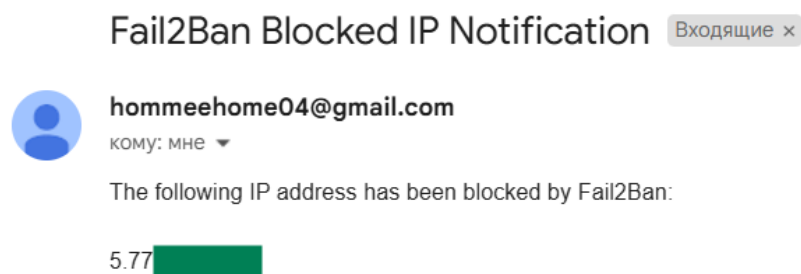
```
GNU nano 6.2
#
# Config file for sSMTP sendmail
#
# The person who gets all mail for userids < 1000
# Make this empty to disable rewriting.
#root=postmaster

# The place where the mail goes. The actual machine name
# MX records are consulted. Commonly mailhosts are named
#mailhub=mail

# Where will the mail seem to come from?
#rewriteDomain=

# The full hostname
# root=your-email@gmail.com
root=hommeehome04@gmail.com
mailhub=smtp.gmail.com:587
FromLineOverride=YES
UseSTARTTLS=YES
AuthUser=hommeehome04@gmail.com
AuthPass=
hostname=ip-172-31-88-25.ec2.internal
UseTLS=YES
rewriteDomain=gmail.com
# Are users allowed to set their own From: address?
# YES - Allow the user to specify their own From: address
# NO - Use the system generated From: address
#FromLineOverride=YES
```

Lastly, we run the script, and after 5 wrong inputs we get an email



Docker + Apache2 Reverse Proxy

Research and implement a solution using Apache2 as a reverse proxy with Docker containers

Install Docker

- `sudo apt update`
- `sudo apt install docker-ce docker-ce-cli containerd.io`

Install Apache2

- `sudo apt install apache2`
- `sudo systemctl status apache2`

Install Modules for Reverse Proxy

- `sudo a2enmod proxy`
- `sudo a2enmod proxy_http`
- `sudo a2enmod rewrite`
- `sudo a2enmod headers`

Open the Docker

- `sudo docker build -t my-html-project`
- `sudo docker run -d -p 8080:80 my-html-project`
- Set the Apache for proxifying the request on your container. Open the configuration file of the virtual host Apache
`sudo nano /etc/apache2/sites-available/000-default.conf`

```
<VirtualHost *:80>
    DocumentRoot /var/www/html
    ServerAdmin webmaster@localhost

    # Реверс-прокси
    ProxyPreserveHost On
    ProxyPass / http://localhost:8080/
    ProxyPassReverse / http://localhost:8080/

    # Логирование
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

- `sudo systemctl restart apache2`

Open browser

