VPS Spring Boot Deployment

How to host a spring boot project in a vps, the full process of setup the vps and hot it.

Hosting a Spring Boot project on a VPS involves several steps, including setting up the VPS, configuring security, deploying the application, and ensuring it's running as a service. Below is a detailed guide:

Step 1: Choose a VPS Provider

You can choose any VPS provider such as:

- **DigitalOcean**
- **Linode**
- **Vultr**
- **AWS EC2**
- **Google Cloud Compute Engine**
- **Azure Virtual Machines**

Step 2: Set Up the VPS

Once you have purchased a VPS, follow these steps:

1. Connect to the VPS

Use SSH to connect to your VPS:

```sh
ssh root@your-vps-ip
```

If you're using a different user:

```sh
ssh youruser@your-vps-ip
```

If your provider uses a key file:

```sh
ssh -i /path/to/private-key.pem user@your-vps-ip
```

---

2. Update System Packages

Update your system to ensure everything is up to date.

For **Ubuntu/Debian**:

```sh
sudo apt update && sudo apt upgrade -y
```

For **CentOS/RHEL**:

```sh
sudo yum update -y
```

---

3. Install Java

Spring Boot requires Java to run. Install Java if not already installed.

For **Ubuntu/Debian**:

```sh
sudo apt install openjdk-17-jdk -y
```

For **CentOS/RHEL**:

```sh
```

```
sudo yum install java-17-openjdk -y
```

Verify the installation:

```sh
java -version
```

Step 3: Deploy the Spring Boot Application

1. Transfer the JAR File

If your Spring Boot application is packaged as a JAR file, upload it to the VPS.

Using **scp** (replace `your.jar` with your actual file name):

```sh
scp your.jar user@your-vps-ip:/home/user/
```

Alternatively, you can use **FTP/SFTP** with tools like **FileZilla**.

---

2. Run the Application

Navigate to the directory where you uploaded the JAR file and start the Spring Boot application:

```sh
java -jar your.jar --server.port=8080
```

This will run the application but will terminate if you close the SSH session.

Step 4: Run Spring Boot as a Service

To ensure the application runs continuously, set it up as a systemd service.

1. Create a Service File

Create a service file:

```sh
sudo nano /etc/systemd/system/springboot.service
```

Add the following content:

```ini
[Unit]
Description=Spring Boot Application
After=network.target

[Service]
User=user
WorkingDirectory=/home/user
ExecStart=/usr/bin/java -jar /home/user/your.jar
SuccessExitStatus=143
Restart=always
RestartSec=5

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

Save the file (`CTRL+X` , then `Y` , then `ENTER`).

---

2. Enable and Start the Service

Reload systemd and enable the service:

```sh
sudo systemctl daemon-reload
sudo systemctl enable springboot
sudo systemctl start springboot
```

To check the status:

```sh
sudo systemctl status springboot
```

To restart the service:

```sh
sudo systemctl restart springboot
```

Step 5: Configure Firewall

If you are using **UFW** (Ubuntu/Debian):

```sh
sudo ufw allow 8080/tcp
sudo ufw enable
```

For **firewalld** (CentOS/RHEL):

```sh
sudo firewall-cmd --permanent --add-port=8080/tcp
sudo firewall-cmd --reload
```

Step 6: Set Up Reverse Proxy (Optional)

If you want to serve the application using a domain, you can use **NGINX** as a reverse proxy.

1. Install NGINX

For Ubuntu/Debian:

```sh
sudo apt install nginx -y
```

For CentOS:

```sh
sudo yum install nginx -y
```

2. Configure NGINX

Edit the configuration file:

```sh
sudo nano /etc/nginx/sites-available/springboot
```

Add the following content:

```nginx
server {
    listen 80;
    server_name yourdomain.com;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

Enable the site:

```sh
sudo ln -s /etc/nginx/sites-available/springboot /etc/nginx/sites-enabled/
```

Restart NGINX:

```sh
sudo systemctl restart nginx
```

If using a domain, ensure **DNS records** are pointing to your VPS IP.

Step 7: Secure with SSL (HTTPS)

Use **Certbot** to get a free SSL certificate:

```sh
sudo apt install certbot python3-certbot-nginx -y
```

Run the following command:

```sh
sudo certbot --nginx -d yourdomain.com
```

Follow the instructions to complete SSL installation.

To automatically renew:

```sh
sudo certbot renew --dry-run
```

Step 8: Monitoring and Logging

Check logs using:

```sh
journalctl -u springboot -f
```

For NGINX logs:

```sh
sudo tail -f /var/log/nginx/access.log
```

Step 9: Set Up a Database (If Needed)

If your application requires a database like MySQL or PostgreSQL, install it:

For MySQL:

```sh
sudo apt install mysql-server -y
sudo mysql_secure_installation
```

For PostgreSQL:

```sh
sudo apt install postgresql postgresql-contrib -y
```

Create a database:

```sql
CREATE DATABASE yourdb;
CREATE USER youruser WITH ENCRYPTED PASSWORD 'yourpassword';
GRANT ALL PRIVILEGES ON DATABASE yourdb TO youruser;
```

Configure **application.properties** in your Spring Boot project to connect to the database.

Step 10: Automate Deployment (Optional)

To automate deployments, consider using **Git and a deployment script**:

1. Install Git:

```sh
sudo apt install git -y
```

2. Clone your repository:

```sh
git clone https://github.com/your-repo.git
```

3. Pull latest changes and restart service:

```sh
cd /home/user/your-repo
git pull origin main
sudo systemctl restart springboot
```

You can automate this using **GitHub Actions** or a **CI/CD pipeline**.

Conclusion

Now your Spring Boot application is successfully hosted on a VPS, running as a service, secured with a firewall and SSL, and optionally using a reverse proxy. 🚀

Let me know if you need further help!