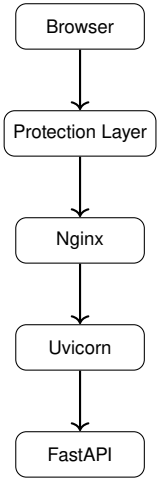
8 Deployment

CS 425
Web Applications Development

Alex Petrovici

Content

Deployment / Introduction



Sends requests to the server.

Cloudflare/Akamai/Amazon CloudFront
Provides CDN, DDOS protection, and SSL termination.

Handles static files (css, js, images) and acts as a reverse proxy.

Manages FastAPI workers (handles requests and distributes them to the FastAPI application).
It is not exposed directly to the internet.

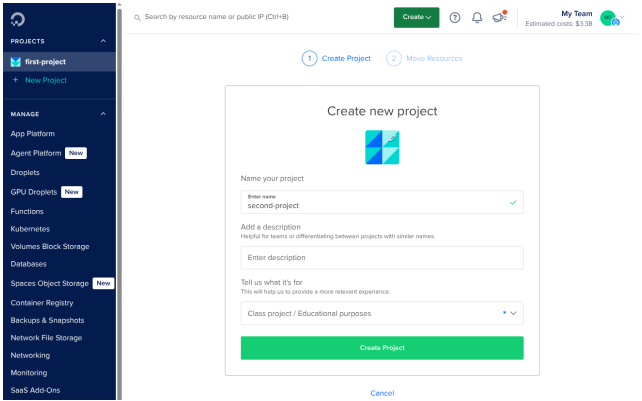
Handles application logic.

Server configuration / Create a new server

1. Create a new server
2. Setup users
3. Setup the server
4. Setup uvicorn
5. Setup nginx

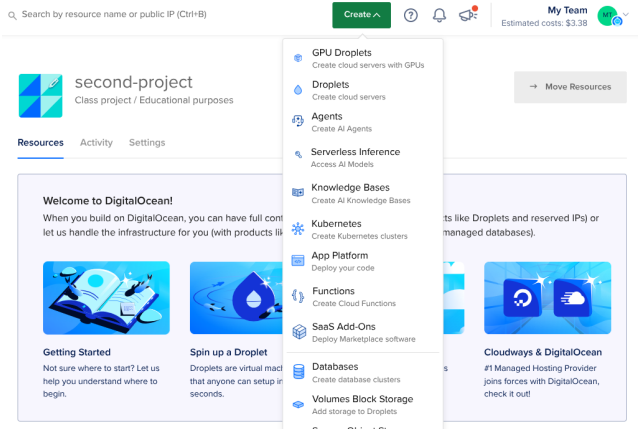
Server configuration / Create a new server

1. Make an account in DigitalOcean.
2. Make a new project.



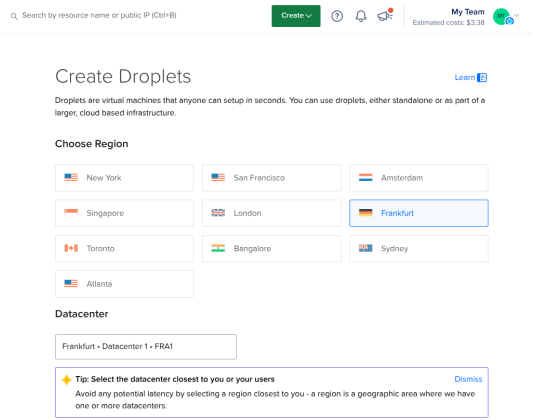
Server configuration / Create a new server

3. Make the server. (Go to Create / Droplets)



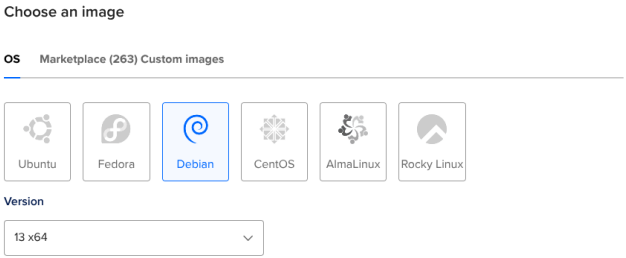
Server configuration / Create a new server

4. Choose the region closest to your users.



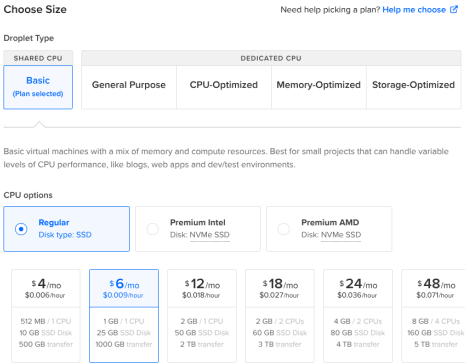
Server configuration / Create a new server

5. Choose the OS you prefer (recommended Ubuntu or Debian).



Server configuration / Create a new server

6. Choose the size of the server. (We can start with the smallest one. For reference, Debian can run with at least 512MB of RAM, but is recommended to use at least 1GB of RAM.)



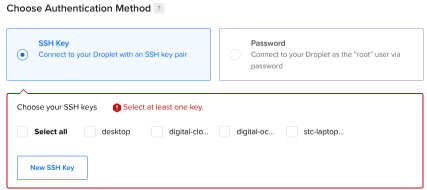
Server configuration / Create a new server

7. Now make a ssh key to access the server.

```
❱ cd ~/.ssh
❱ ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/alex/.ssh/id_ed25519): class_project
Enter passphrase for "class_project" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in class_project
Your public key has been saved in class_project.pub
The key fingerprint is:
SHA256:kaRDwO5lstihC204zbIvUSRfuKx5kLr1E7K42OU1WHY alex@zenbook-debian
The key[1]s randomart image is:
+--[ED25519 256]--+
|   o... .        |
|. o o. o .       |
| * +  o o        |
|*.ooo. .         |
|o+o ..           |
+----[SHA256]-----+
❱ cat class_project.pub | wl-copy
```

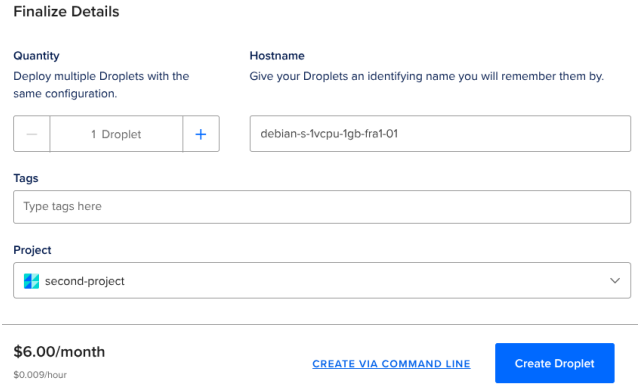
Server configuration / Create a new server

8. Copy the public key to the server.



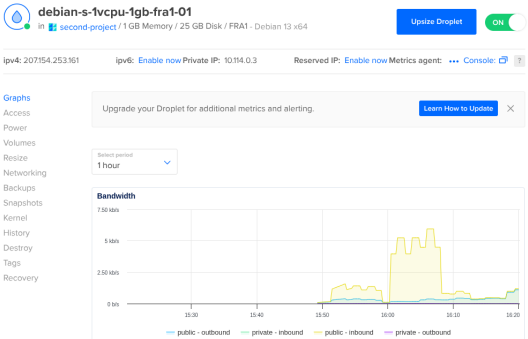
Server configuration / Create a new server

9. Finnish creating the server.



Server configuration / Create a new server

10. Wait until the server is ready.



Server configuration / Create a new server

Setup users

1. Enter in the server as root.

The end goal is to connect via ssh as a user (**NOT** as a root).
To do so, first we need to create this new user. Initially we will enter as root, make the new user, and from there only allow to enter as the user we just created.
Use the previous ip address and the **private** key we created earlier.

```
❱ ssh root@207.154.253.161 -i ~/.ssh/class_project
Linux debian-s-1vcpu-1gb-fra1-01 6.12.43+deb13-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.12.43-1
↪  (2025-08-27) 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.
```

Server configuration / Create a new server

2. Add the new user.

```
adduser <username>
```

In practice, add the user as follows:

```
root@debian-s-1vcpu-1gb-fra1-01:~# adduser project
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for project
Enter the new value, or press ENTER for the default
        Full Name []: project
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] y
root@debian-s-1vcpu-1gb-fra1-01:~#
```

Server configuration / Create a new server

3. Add the user to the sudoers file so we can use sudo and install packages.

```
usermod -aG sudo <username>
```

- `-a`: **append** the user to the supplementary group(s) specified by the `-G` flag. Without this, the user would be removed from all other supplementary groups not listed.
- `-G sudo`: Add the user to the `sudo` group.
- `<username>`: The user to be modified.

In summary, this command adds the user `project` to the `sudo` group, allowing them to execute commands as superuser.
In practice, add the user as follows:

```
root@debian-s-1vcpu-1gb-fra1-01:~# usermod -aG sudo project
```

Server configuration / Create a new server

4. Change the ssh configuration to allow only the new user to login via ssh key.

Since we made the server from digitalocean, we already have the public key loaded for the root user. You can check it by running:

```
cat /root/.ssh/authorized_keys
```

Now we will allow the new user <username> to login by using this ssh key.

```
mkdir -p /home/<username>/.ssh
cp ~/.ssh/authorized_keys /home/<username>/.ssh/
chown -R <username>:<username> /home/<username>/.ssh
chmod 700 /home/<username>/.ssh
chmod 600 /home/<username>/.ssh/authorized_keys
```

- `mkdir -p /home/<username>/.ssh`: Creates the user's `.ssh` directory if it doesn't exist.
- `cp ~/.ssh/authorized_keys /home/<username>/.ssh/`: Copies your current user's public key(s) to the new user's authorized keys file, allowing SSH login with your key.
- `chown -R <username>:<username> /home/<username>/.ssh`: Changes ownership of the `.ssh` directory and its contents to the new user, ensuring they have the correct permissions.
- `chmod 700 /home/<username>/.ssh`: Sets the directory permissions so only the user can read, write, execute.
- `chmod 600 /home/<username>/.ssh/authorized_keys`: Sets the authorized keys file to be readable and writable only by the user for security.

Server configuration / Create a new server

So, in practice, we will run the following commands:

```
root@debian-s-1vcpu-1gb-fra1-01:~# mkdir -p /home/project/.ssh
root@debian-s-1vcpu-1gb-fra1-01:~# cp ~/.ssh/authorized_keys /home/project/.ssh/
root@debian-s-1vcpu-1gb-fra1-01:~# chown -R project:project /home/project/.ssh
root@debian-s-1vcpu-1gb-fra1-01:~# chmod 700 /home/project/.ssh
root@debian-s-1vcpu-1gb-fra1-01:~# chmod 600 /home/project/.ssh/authorized_keys
```

Server configuration / Create a new server

5. Check that you can login with the new user.

```
systemctl restart ssh
```

Now, in the client, in a separate terminal, try to log in with the new user.

```
❱ ssh project@207.154.253.161 -i ~/.ssh/class_project
Linux debian-s-1vcpu-1gb-fra1-01 6.12.43+deb13-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.12.43-1
↪  (2025-08-27) 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.
project@debian-s-1vcpu-1gb-fra1-01:~$
```

Since we connected successfully as <username>, then exit the root session.

Server configuration / Setup the server

1. Update the system.

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

2. Give some swap space to the server. Read more about the process at https://www.digitalocean.com/community/tutorials/how-to-add-swap-space-on-debian-11

```
sudo fallocate -l 2G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
sudo swapon --show
```

Then make permanent by adding the file to the fstab file.

```
sudo cp /etc/fstab /etc/fstab.bak
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

Server configuration / Setup the server

- `fallocate -l 2G /swapfile`: Allocates a file of exactly 2 GiB by reserving blocks in the filesystem metadata. Fast. Does not write zeros; it just marks space as used.
- `chmod 600 /swapfile`: Make the file only accessible to root only.
- `mkswap /swapfile`: Writes swap metadata (header, UUID, page map). Without this, the kernel does not recognize the file as swap.
- `swapon /swapfile`: Registers the file with the kernel and starts using it as swap.
- `swapon --show`: Verification. Shows active swap devices/files.

In practice, we will run the following commands:

```
root@debian-s-1vcpu-1gb-fra1-01:~# sudo fallocate -l 2G /swapfile
root@debian-s-1vcpu-1gb-fra1-01:~# sudo chmod 600 /swapfile
root@debian-s-1vcpu-1gb-fra1-01:~# sudo mkswap /swapfile
Setting up swapspace version 1, size = 2 GiB (2147479552 bytes)
no label, UUID=090997d6-3fd9-4e7c-876d-8abba6289ad0
root@debian-s-1vcpu-1gb-fra1-01:~# sudo swapon /swapfile
root@debian-s-1vcpu-1gb-fra1-01:~# sudo swapon --show
NAME      TYPE SIZE USED PRIO
/swapfile file   2G   0B   -2
```

Server configuration / Setup the server

4. Install required system packages:

```
sudo apt install python3 python3-pip python3-venv nginx git pipx ncurses-term nano -y
pipx ensurepath
pipx install poetry
pipx ensurepath
```

   Then close the terminal and connect again.

5. Inside the server, link the server to your github account using ssh (using `ssh-keygen`).
6. Then enable the ssh client to use the private key with:

```
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/<private_key>
```

Server configuration / Setup the server

7. And clone your project repository.

```
git clone <repository-url>
git clone git@github.com:agpetrovici/fastapi_project.git
cd fastapi_project
```

8. Install the dependencies.

```
python3 -m venv ./.venv
source ./.venv/bin/activate
poetry install --only main --no-interaction --no-ansi
```

Server configuration / Setup the server

Sanity check: test whether the server can run the application.

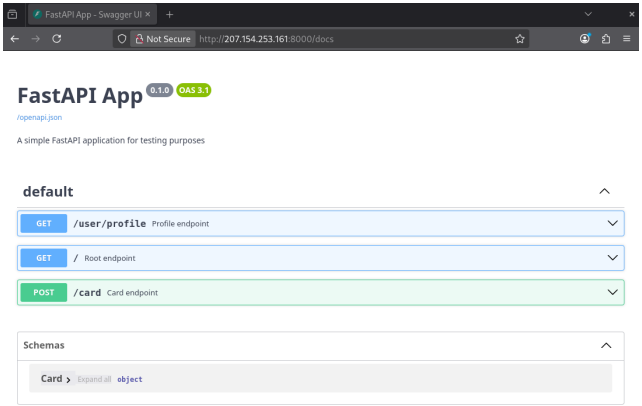
```
source ./.venv/bin/activate
python3 app/backend/main.py
```

Expected output:

```
(.venv) project@debian-s-1vcpu-1gb-fra1-01:~/fastapi_project$ python3 app/backend/main.py
INFO:     Started server process [11842]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     86.127.228.177:55316 - "GET / HTTP/1.1" 307 Temporary Redirect
INFO:     86.127.228.177:55316 - "GET /docs HTTP/1.1" 200 OK
INFO:     86.127.228.177:55316 - "GET /openapi.json HTTP/1.1" 200 OK
```

Server configuration / Setup the server

In a browser, go to `http://<server-ip>:8000` and you should see it working:



This would work as long as the terminal is working. Next step is to configure a systemd

Server configuration / Setup uvicorn and nginx

Setup uvicorn

We will setup nginx and uvicorn to run the application automatically.
First, check that uvicorn works.

```
uvicorn app.backend.application:create_app --factory --reload --host 0.0.0.0 --port 8000
```

Expected output:

```
(.venv) project@debian-s-1vcpu-1gb-fra1-01:~/fastapi_project$ pwd
/home/project/fastapi_project
(.venv) project@debian-s-1vcpu-1gb-fra1-01:~/fastapi_project$ uvicorn
↪  app.backend.application:create_app --factory --reload --host 0.0.0.0 --port 8000
INFO:     Will watch for changes in these directories: ['/home/project/fastapi_project']
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [11861] using WatchFiles
INFO:     Started server process [11863]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Now that it works, we will make the systemd service.

Server configuration / Setup uvicorn and nginx

First create the systemd service file with `sudo nano /etc/systemd/system/project.service`. Add the following service description:

```
[Unit]
Description=Uvicorn instance to serve the project app
After=network.target

[Service]
User=project
Group=www-data
WorkingDirectory=/home/project/fastapi_project
Environment="PATH=/home/project/fastapi_project/.venv/bin"
ExecStart=/home/project/fastapi_project/.venv/bin/uvicorn app.backend.application:create_app
↪  --factory --reload --host 0.0.0.0 --port 8000

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

Save and exit. Then start the service with `sudo systemctl start project`

Server configuration / Setup uvicorn and nginx

In production we'll use a Unix domain socket instead of specifying the host and port, so let's first try that it works.

First, check that uvicorn.sock works.

```
uvicorn app.backend.application:create_app --factory --uds
↪  /home/project/fastapi_project/uvicorn.sock
```

Expected output:

```
(.venv) project@debian-s-1vcpu-1gb-fra1-01:~/fastapi_project$ pwd
/home/project/fastapi_project
(.venv) project@debian-s-1vcpu-1gb-fra1-01:~/fastapi_project$ uvicorn
↪  app.backend.application:create_app --factory --uds
↪  /home/project/fastapi_project/uvicorn.sock
INFO:     Started server process [12305]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on unix socket /home/project/fastapi_project/uvicorn.sock (Press
↪  CTRL+C to quit)
```

Now that it works, we will make the systemd service.

Server configuration / Setup uvicorn and nginx

```
sudo nano /etc/systemd/system/project.service
```

Add the following service description:

```
[Unit]
Description=Uvicorn instance to serve the project app
After=network.target

[Service]
User=project
Group=www-data
WorkingDirectory=/home/project/fastapi_project
Environment="PATH=/home/project/fastapi_project/.venv/bin"
ExecStart=/home/project/fastapi_project/.venv/bin/uvicorn app.backend.application:create_app
↪  --factory --reload --uds /home/project/fastapi_project/uvicorn.sock

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

Save and exit.

Server configuration / Setup uvicorn and nginx

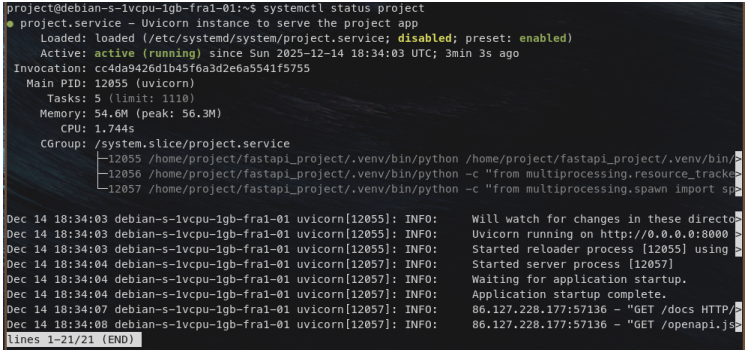
Then start the service.

```
sudo systemctl start project
```

At this point, you can't access the app yet since we need nginx to redirect trafic through the socket. But if you use the host and port option, then you should be able to access the application at `http://<server-ip>:8000`.
Then enable the service to start on boot.

```
sudo systemctl enable project
```

Server configuration / Setup uvicorn and nginx



Now we can control the behaviour using a service, which is great because we can enter and exit the server without depending on the client terminal.

Server configuration / Setup uvicorn and nginx

Fixing issues

If we make mistakes in the configuration file, then the service will not start (and we won't see a traceback since we're not runningthe script from the terminal).
If the service is active, (running) but it has an internal error, it won't appear in the status, but we can check the detailed logs with

```
sudo journalctl -u project
```

Then we would need to update project.service file, and after changing it, we need to reload systemctl and then restart the service.

```
sudo systemctl daemon-reload
sudo systemctl restart project
```

Server configuration / Setup uvicorn and nginx

Setup nginx

We will setup nginx to connect external traffic to the uvicorn service.

```
sudo nano /etc/nginx/sites-available/project.conf
```

Add the following service description:

```
server {
    listen 80;
    server_name 207.154.253.161;
    location / {
        include proxy_params;
        proxy_pass http://unix:/home/project/fastapi_project/uvicorn.sock;
    }
}
```

Save and exit.

Server configuration / Setup uvicorn and nginx

We need fix the permissions of the socket file so nginx can access it.

```
sudo chmod 755 /home/project
sudo chmod 755 /home/project/fastapi_project
```

Then restart the uvicorn service to recreate the socket with correct permissions.

```
sudo systemctl restart project
```

Now enable the configuration (makes a symlink to the file in the sites-enabled directory).

```
sudo ln -s /etc/nginx/sites-available/project.conf /etc/nginx/sites-enabled/
```

Server configuration / Setup uvicorn and nginx

Test the configuration.

```
sudo nginx -t
```

Expected output:

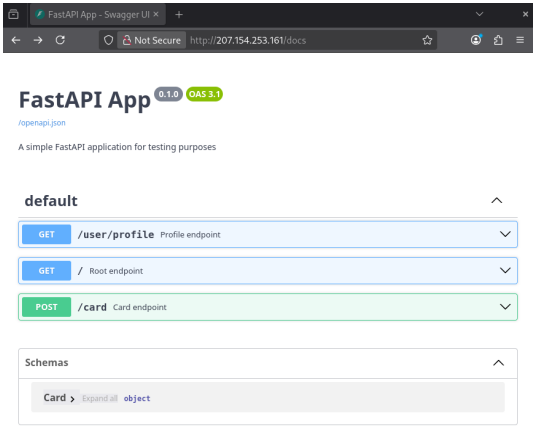
```
project@debian-s-1vcpu-1gb-fra1-01:~$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

Now restart nginx to load the new configuration.

```
sudo systemctl restart nginx
```

Server configuration / Setup uvicorn and nginx

Now you should be able to access the application at `http://<server-ip>`.



Security / SSH security

Disable remote login and login via password.

First edit the ssh configuration file.

```
sudo nano /etc/ssh/sshd_config
```

Add the following lines:

```
PermitRootLogin no
PasswordAuthentication no
```

Save and exit. Then restart the ssh service.

```
sudo systemctl restart ssh
```

Security / Firewall configuration

Block unused ports.

Since this is a production server that serves a website (80, 443) and ssh (22), then we will only allow these ports to be accessible from the internet.
We will use **ufw** (uncomplicated firewall) to manage the firewall.

```
sudo apt-get install ufw -y
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable  # This command activates the firewall with the rules you just configured.
```

Security / Control access

Ban attackers that try to log in over ssh.

```
sudo apt update
sudo apt install fail2ban -y
```

Make a jail to control ssh access.

```
sudo nano /etc/fail2ban/jail.d/sshd.local
```

Security / Control access

Add the following lines:

```
[DEFAULT]
bantime   = -1
findtime  = 300
maxretry  = 1
backend   = systemd
loglevel  = INFO
logtarget = /var/log/fail2ban.log

[sshd]
enabled   = true
port      = 22
filter    = sshd
logpath   = /var/log/auth.log
```

Save and exit. Then reload to put it into production `sudo fail2ban-client reload` or `sudo systemctl restart fail2ban`

Security / Control access

Now you can check the logs to see if any attackers are being banned.

```
sudo fail2ban-client status sshd
```

Expected output:

```
project@debian-s-1vcpu-1gb-fra1-01:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed: 8
|  `- Journal matches:  _SYSTEMD_UNIT=ssh.service + _COMM=sshd
`- Actions
   |- Currently banned: 1
   |- Total banned: 1
   `- Banned IP list:   64.23.135.179
```

This shows fail2ban is already blocking unauthorized SSH login attempts.
To prevent an ever-growing permanent ban list, set bantime to a value greater than 0.

Security / Control access

If you want to block all access to the IPs detected by the sshd jail (not just SSH access), you can configure `fail2ban` to use firewall rules that apply system-wide.
In the `sshd.local` file, set the action to use the `iptables` banning for all ports (instead of just restricting SSH):

```
[sshd]
enabled   = true
port      = 22
filter    = sshd
logpath   = /var/log/auth.log
action    = iptables-allports[name=SSHD, protocol=all]
```

This configuration uses the `iptables-allports` action, which blocks traffic from banned IPs across all ports, not just SSH.

Security / Control access

After making this change, reload the fail2ban configuration:

```
sudo fail2ban-client reload
```

Now, any IP that gets banned by the sshd jail will be blocked from accessing *any* port on the server, not just SSH.

Security / Control access

If you are trying to test the firewall rules and you find yourself locked out of the server, you can unlock it by running the following command:

```
sudo fail2ban-client set <jail-name> unbanip <ip-address>
```

For example:

```
sudo fail2ban-client set sshd unbanip 170.39.218.202
```

NOTE: You can use `https://www.abuseipdb.com/` to check if an ip is associated with other risks.

Extras /

- Setup fail2ban to block access of ips that are trying to access sensitive files directly like /etc/passwd or /etc/shadow .
- Setup SSL so users can access the application using https (port 443).
- Setup a custom domain so users can access the application using a custom domain (e.g. `https://www.example.com`).
- Setup Cloudflare or another CDN & Security layer.