10

CHAPTER

EC2 インスタンスとデプロイ

本書最大の山場 — Django を世界に公開する

本書のサンプル章

AGENDA

この章で学ぶこと

01 **EC2 インスタンスの起動**
AWS CLI で SG ・キーペア・ AMI ・起動

02 **SSH 接続**
ターミナルからサーバーにログイン

03 **サーバーの初期設定**
Ubuntu + uv + Python 3.13 + Nginx

04 **アプリケーションのデプロイ**
rsync + .env.prod + migrate + collectstatic

05 **Gunicorn + Systemd**
WSGI サーバーの自動起動

06 **Nginx リバースプロキシ**
静的ファイル配信 + Gunicorn 連携

ARCHITECTURE

本番環境のアーキテクチャ

PART 1

EC2 セットアップ

AWS CLI でインスタンスを起動 → SSH で初期設定

EC2 LAUNCH

AWS CLI で起動 — 5 ステップ

01 セキュリティグループ

ファイアウォール作成
+ ポート 22/80/443 開放

02 キーペア

SSH 秘密鍵を生成
.pem ファイルを DL

03 AMI ID 取得

Ubuntu 22.04 LTS
最新パッチ版

04 インスタンス起動

t2.micro (無料枠)
run-instances

05 情報確認

パブリック IP 取得
running 状態を待つ

→ 約 30 秒〜 1 分で起動完了。 aws ec2 wait instance-running で待機。

セキュリティグループ — 仮想ファイアウォール

ポート開放コマンド（3 つ）

```
#  現在の IP を取得
MY_IP=$(curl -s https://checkip.amazonaws.com)

# SSH (port 22) —  自分の IP のみ
aws ec2 authorize-security-group-ingress \
    --group-id $SG_ID --protocol tcp \
    --port 22 --cidr $MY_IP/32

# HTTP (port 80) —  全公開
aws ec2 authorize-security-group-ingress \
    --group-id $SG_ID --protocol tcp \
    --port 80 --cidr 0.0.0.0/0

# HTTPS (port 443) —  全公開 ( 第11 章で使用 )
aws ec2 authorize-security-group-ingress \
    --port 443 --cidr 0.0.0.0/0 ...
```

開放するポート

22 — **SSH**
自分の IP だけ

80 — **HTTP**
全世界に公開

443 — **HTTPS**
次章 SSL 化で使用

キーペア → AMI → インスタンス起動

KEY PAIR

SSH 秘密鍵

```
aws ec2 create-key-pair \
  --key-name \
    adventure-game-key \
  --query 'KeyMaterial' \
  --output text \
  > ~/.ssh/\
    adventure-game-key.pem

# パーミッション必須
chmod 400 \
  ~/.ssh/\
    adventure-game-key.pem
```

AMI ID

Ubuntu 22.04 LTS

```
# 最新AMI ID 取得
AMI_ID=$(\
  aws ec2 describe-images \
  --owners 099720109477 \
  --filters \
    "Name=name,Values=\
      ubuntu/images/.../\
      ubuntu-jammy-22.04-\
      amd64-server-*" \
  --query 'Images | \
    sort_by(@,&CreationDate)\
    | [-1].ImageId')
```

RUN

インスタンス起動

```
aws ec2 run-instances \
  --image-id $AMI_ID \
  --instance-type \
    t2.micro \
  --key-name \
    adventure-game-key \
  --security-group-ids \
    $SG_ID \
  --subnet-id $SUBNET_ID

# 起動完了を待つ
aws ec2 wait \
  instance-running
```

SSH 接続 — 公開鍵で安全にログイン

COMMAND

Mac / Linux

```
$ ssh -i ~/.ssh/adventure-game-key.pem ubuntu@$PUBLIC_IP
```

公開鍵 = 南京錠 / 秘密鍵 = 開ける鍵

公開鍵

南京錠

サーバーに設置
誰でも見て OK

秘密鍵（.pem）

開ける鍵

あなたの PC に保存
絶対に他人に渡さない

接続成功

```
Welcome to Ubuntu 22.04.3 LTS
(GNU/Linux 6.2.0-aws x86_64)

ubuntu@ip-172-31-xx-xx:~$ _
```

SETUP

サーバーの初期設定

01 システム更新

```
$ sudo apt update
$ sudo apt upgrade -y
```

5 〜 10 分かかる

02 uv インストール

```
$ curl -LsSf https://astral.sh\
    /uv/install.sh | sh
```

PATH を通す: source ~/.local/bin/env

03 Python 3.13

```
$ uv python install 3.13
```

uv で管理 → 仮想環境で自動使用

04 Nginx

```
$ sudo apt install nginx -y
$ sudo systemctl status nginx
```

active (running) なら OK

PART 2

アプリのデプロイ

rsync でファイルを送り、本番設定を整える

rsync でファイル送信 + uv sync

ローカル → サーバー (rsync)

```
# プロジェクトディレクトリで実行
cd ~/projects/adventure-game

# 除外ファイルを指定して送信
rsync -avz \
  --exclude '.venv/' \
  --exclude '__pycache__/' \
  --exclude '*.pyc' \
  --exclude '.env' \
  --exclude 'db.sqlite3' \
  --exclude 'staticfiles/' \
  -e "ssh -i ~/.ssh/\
      adventure-game-key.pem" \
  ./ ubuntu@$PUBLIC_IP:\
    /home/ubuntu/adventure-game/
```

rsync オプション

`-a`	属性保持
`-v`	詳細表示
`-z`	圧縮転送
`--exclude`	除外指定

サーバー側で依存解決

```
$ uv sync
```

→ *pyproject.toml から一括*

→ *仮想環境を再現*

.env.prod + migrate + collectstatic

01

.env.prod 作成

ローカルで DEBUG=False
ALLOWED_HOSTS 設定
rsync で送信

```
DEBUG=False
SECRET_KEY=…
ALLOWED_HOSTS=3.x.x.x
```

02

シンボリックリンク

サーバー側で
.env.prod を
.env として参照

```
$ ln -s .env.prod .env
```

03

マイグレーション

DB スキーマを適用
スーパーユーザー作成

```
$ uv run python\
  manage.py migrate
$ ... createsuperuser
```

04

静的ファイル収集

CSS/JS/ 画像を
staticfiles/ に集約
Nginx が配信

```
$ uv run python\
  manage.py\
  collectstatic\
  --noinput
```

WSGI サーバーをテスト起動

インストール → 起動 → 確認

```
# ローカルで gunicorn を追加
$ uv add gunicorn

# pyproject + lock を rsync で送信
$ rsync ... pyproject.toml uv.lock ...

# サーバー側で同期
$ uv sync

# テスト起動（8000 ポート）
$ uv run gunicorn \
    server.wsgi:application \
    --bind 0.0.0.0:8000

# ブラウザ確認用 — SSH ポートフォワード
$ ssh -i ... -L 8000:localhost:8000\
    ubuntu@$PUBLIC_IP
```

runserver vs Gunicorn

項目	runserver	Gunicorn
性能	低 / 単一プロセス	高 / マルチワーカー
セキュリティ	開発用機能あり	本番最適化
静的配信	Django (非効率)	Nginx (高速)
自動起動	手動	Systemd

PART 3

本番運用設定

Systemd で自動化 + Nginx で公開

サービスファイルで Gunicorn を自動化

/etc/systemd/system/adventure-game.service

```
[Unit]
Description=Adventure Game Gunicorn
After=network.target

[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/adventure-game
EnvironmentFile=/home/.../.env

ExecStart=/.../.venv/bin/gunicorn \
    --workers 3 \
    --bind unix:/.../adventure_game.sock \
    server.wsgi:application

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

サービス操作

daemon-reload 設定反映

$ sudo systemctl …

start サービス起動

$ sudo systemctl …

enable 自動起動 ON

$ sudo systemctl …

status 状態確認

$ sudo systemctl …

journalctl -u … ログ確認

$ sudo systemctl …

Nginx — 静的配信 + リバースプロキシ

/etc/nginx/sites-available/adventure-game

```
server {
    listen 80;
    server_name 3.112.xxx.xxx;

    # 静的ファイル
    location /static/ {
        alias /home/ubuntu/\
            adventure-game/staticfiles/;
    }

    # Gunicorn へのプロキシ
    location / {
        include proxy_params;
        proxy_pass http://unix:\
            /home/ubuntu/.../\
            adventure_game.sock;
    }
}
```

有効化と再起動

01 シンボリックリンク

```
$ sudo ln -s …\
  sites-available/…\
  sites-enabled/
```

02 デフォルト削除

```
$ sudo rm \
  sites-enabled/default
```

03 構文チェック

```
$ sudo nginx -t
```

04 再起動

```
$ sudo systemctl \
  restart nginx
```

VERIFY

動作確認 + よくある躓き

ACCESS

ブラウザで確認

```
http://3.112.xxx.xxx
```

成功のチェックリスト

- ✓ トップページが表示される
- ✓ CSS / 画像が読み込まれている
- ✓ 新しいゲームを開始できる
- ✓ /admin/ もレイアウト崩れなし

※ HTTP なので警告が出る場合あり（次章で HTTPS 化）

よくある躓きポイント

ALLOWED_HOSTS 未設定

DEBUG=False で 400 Bad Request

`.env.prod に IP を追加`

502 Bad Gateway

Gunicorn が起動していない

`journalctl -u adventure-game`

403 Forbidden

soft/static ファイルの権限

`www-data → ubuntu グループに`

CSS が当たらない

collectstatic 未実行

`uv run … collectstatic --noinput`

DEPLOYED

世界に公開された。次は HTTPS 化。

この章で学んだこと

- ✓ EC2 インスタンスを CLI で起動
- ✓ セキュリティグループ + キーペア
- ✓ SSH 接続 + Ubuntu 初期設定
- ✓ rsync で本番にデプロイ
- ✓ Gunicorn (WSGI) のテスト起動
- ✓ Systemd で Gunicorn を自動化
- ✓ Nginx でリバースプロキシ + 静的配信

NEXT

第 11 章

HTTPS 化

- ▸ Let's Encrypt で証明書発行
- ▸ Nginx で TLS 終端
- ▸ https:// での公開

→ ブラウザの「保護されていない」を消す

EC2 を使わない時は stop-instances で課金を抑える。