**Модуль 14: Тестування та розгортання веб-додатків**

**Заняття 2: Deployment**

**Вступ до деплою**

Рано чи пізно будь-який заст. осунок потрібно показувати користувачам. Отже, його необхідно розмістити на будь-якому доступному сервері в мережі.

Доставку нової версії застосунку на сервер заведено називати деплоєм (від англ. "deploy"). Ця процедура включає декілька кроків і є досить складною. Все залежить від технологій, які використовувалися під час розробки.

Майже завжди будь-який деплой складається з наступних кроків:

1. Доставка коду проекту на сервер.
2. Підготовка оточення. Застосунок може спиратися на різні змінні оточення, тому необхідно їх встановити на стороні сервера.
3. Встановлення залежностей.
4. Міграції. Якщо застосунок працює з базою даних, то її необхідно розгорнути або оновити структуру.
5. Запуск застосунку.

**Сервер**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/intro#%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80)Існує кілька способів розгортання застосунку **FastAPI** або**Django**, залежно від вашої інфраструктури та вимог. Ось кілька популярних варіантів:

1. **Використання виділеного сервера або VPS:** ви можете налаштувати виділений сервер або віртуальний приватний сервер ( VPS) і запустити свій застосунок за допомогою веб-сервера, такого як NGINX або Apache. Вам потрібно буде налаштувати веб-сервер для проксі-запитів до вашого застосунку і налаштувати диспетчер процесів, такий як systemd або Supervisor, щоб застосунок працював.
2. **Використання постачальника платформи як послуги (PaaS)**. Існує кілька постачальників PaaS, які дають змогу розгортати застосунок, не турбуючись про серверну інфраструктуру. Деякі популярні варіанти включають Heroku, Google App Engine і AWS Elastic Beanstalk. Ці постачальники зазвичай мають власний процес розгортання, тому вам необхідно слідувати їх документації.
3. **Використання платформи оркестрації контейнерів.** Ви можете контейнеризувати свій застосунок за допомогою Docker і розгорнути його за допомогою платформи оркестровки контейнерів, такої як Kubernetes або Docker Swarm. Цей варіант дає вам більший контроль над інфраструктурою і дає змогу легко масштабувати ваш застосунок.
4. **Використання постачальника безсерверних функцій.** Якщо у вашого застосунку невелика кількість запитів, ви також можете використовувати постачальника безсерверних функцій, наприклад AWS Lambda, Azure Functions або Google Cloud Functions. Вам потрібно буде налаштувати функцію для запуску вашого застосунку та обробки вхідних запитів.

Кожен із цих варіантів має свій власний набір переваг та компромісів, тому вам необхідно враховувати свої конкретні вимоги, перш ніж вибрати найкращий варіант для вашого застосунку. Майже всі варіанти не є безкоштовними і це відразу обмежує нам вибір у процесі навчання.

INFO

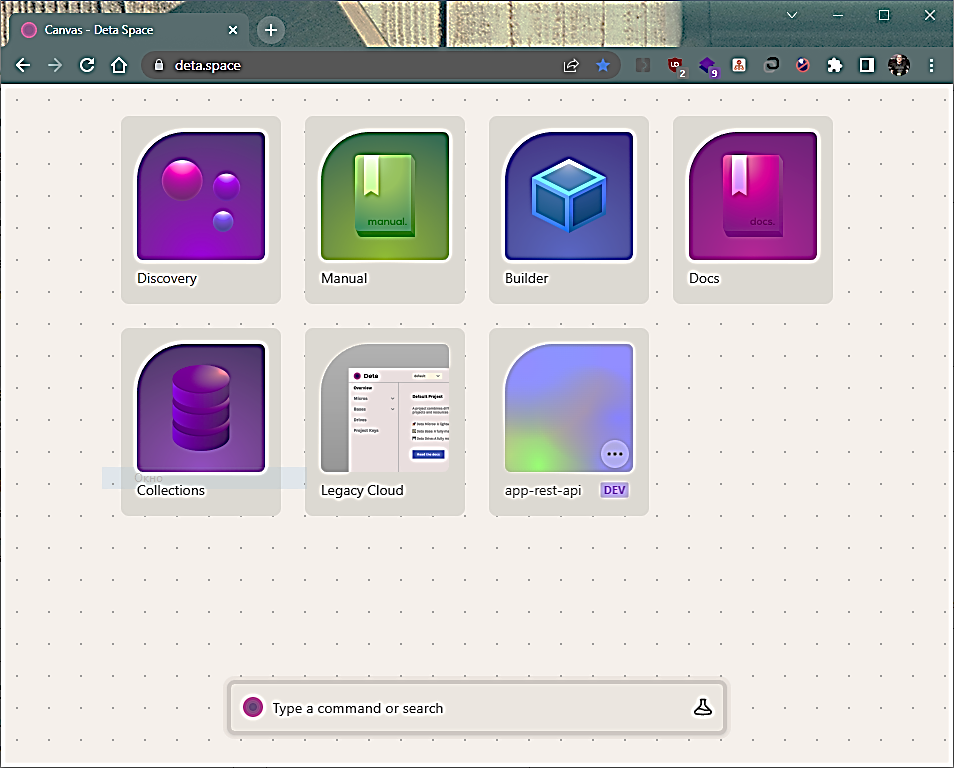
Також варто зазначити, що спеціальні інструменти встановлення Django, такі як django-heroku і django-whitenoise, також можуть допомогти вам у процесі розгортання ваших проектів.

**Сервіс Deta Space**

Перший безкоштовний хмарний сервіс для деплою нашого застосунку, який ми розглянемо, - це **Deta Space**. Зареєструйтеся на сервісі [**Deta Space**](https://deta.space/) (https://deta.space/)**.**

Виконувати деплой (розгортання) ми будемо для останнього варіанта нашого [**застосунку**](https://github.com/GoIT-Python-Web/rest-api-tests) (https://github.com/GoIT-Python-Web/rest-api-tests)

Після реєстрації ми потрапимо в дашборд сервісу:



**Deta Space dashboard**

Нам необхідно вибрати Builder. Тут будуть усі наші застосунки, для яких ми виконаємо процес деплою.



**Deta Space dashboard**

Натиснувши кнопку "Start Building", ми потрапимо на процес деплою нашого застосунку, але спочатку необхідно виконати підготовку нашого коду до процесу.

**Підготовка коду до деплою**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#%D0%BF%D1%96%D0%B4%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%BA%D0%B0-%D0%BA%D0%BE%D0%B4%D1%83-%D0%B4%D0%BE-%D0%B4%D0%B5%D0%BF%D0%BB%D0%BE%D1%8E)**Deta Space** не використовує poetry, тому для нього необхідний файл requirements.txt. Створіть його вручну, скопіювавши необхідні пакети з файлу pyproject.toml. Обов'язково замініть пакет psycopg2 на його готову бінарну версію psycopg2-binary. Оскільки **Deta Space**не може у своєму середовищі його нормально зібрати, нам доводиться використовувати вже зібрану заздалегідь версію пакета.

fastapi

psycopg2-binary

python-jose[cryptography]

passlib[bcrypt]

python-multipart

libgravatar

fastapi-mail

redis

fastapi-limiter

cloudinary

sqlalchemy

pydantic[dotenv]

Враховуйте, що на момент написання розділу, **Deta Space** підтримував Python версії 3.9. Тому у всіх місцях, де використовувався тип сполучення як Note | None, замініть на Union[Note, None] для підтримки версії Python 3.9. Використовувати оператор match теж поки не вийде. За необхідності проставте версії пакетів у файлі requirements.txt.

Також сервіс **Deta Space**запускає наш застосунок за допомогою сервера uvicorn самостійно і є вимога, щоб точка входу в наш застосунок називалася main.py, а застосунок app. У нашому випадку все так і є, але, якщо у вас інакше, змініть назву.

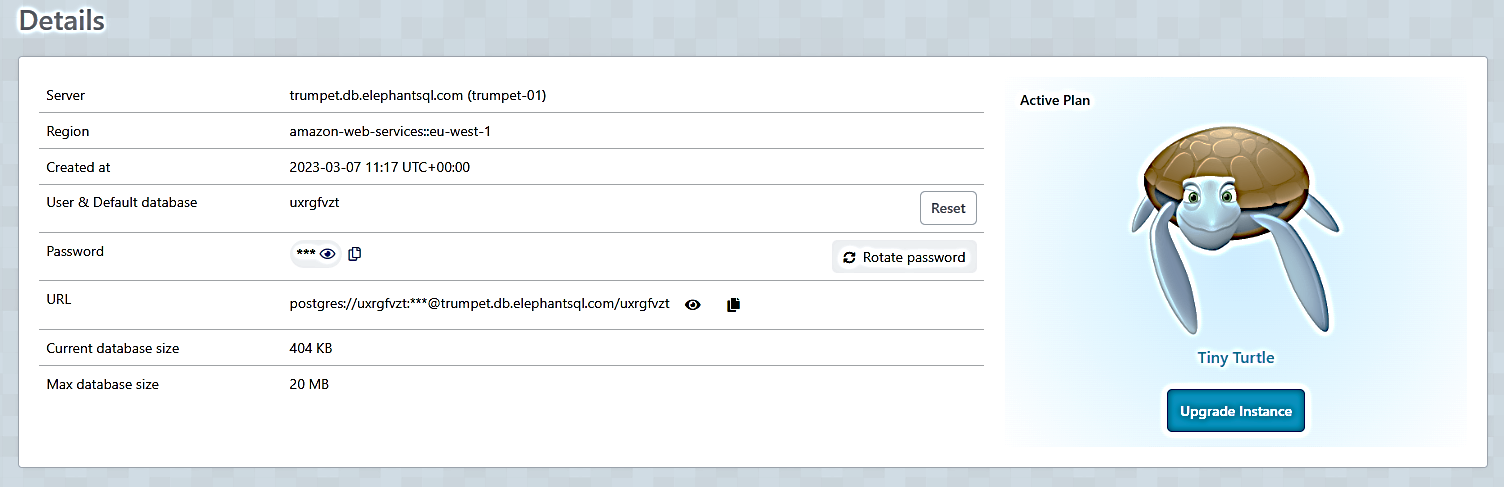
**Бази даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#%D0%B1%D0%B0%D0%B7%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)**PostgreSQL**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#postgresql)[ElephantSQL](https://www.elephantsql.com/) (https://www.elephantsql.com/) - це хмарний сервіс для зберігання та управління реляційними базами даних, заснований на відкритому вихідному коді PostgreSQL. Сервіс надає різні тарифні плани для різних розмірів баз даних і кількості підключень, а також надає можливість створювати кілька баз даних і користувачів для кожної з них.

Мета ElephantSQL - надати простий і зручний спосіб створення та управління базами даних на основі PostgreSQL. Завдяки хмарній архітектурі, користувачі можуть створювати та масштабувати свої бази даних, залежно від потреб свого бізнесу без необхідності встановлення та управління фізичною інфраструктурою.

Виконаємо реєстрацію і створимо безкоштовний інстанс бази даних PostgreSQL у хмарі. Він має розмір 20 Мб, що цілком достатньо для домашнього завдання.



**Екземпляр бази даних Postgres**

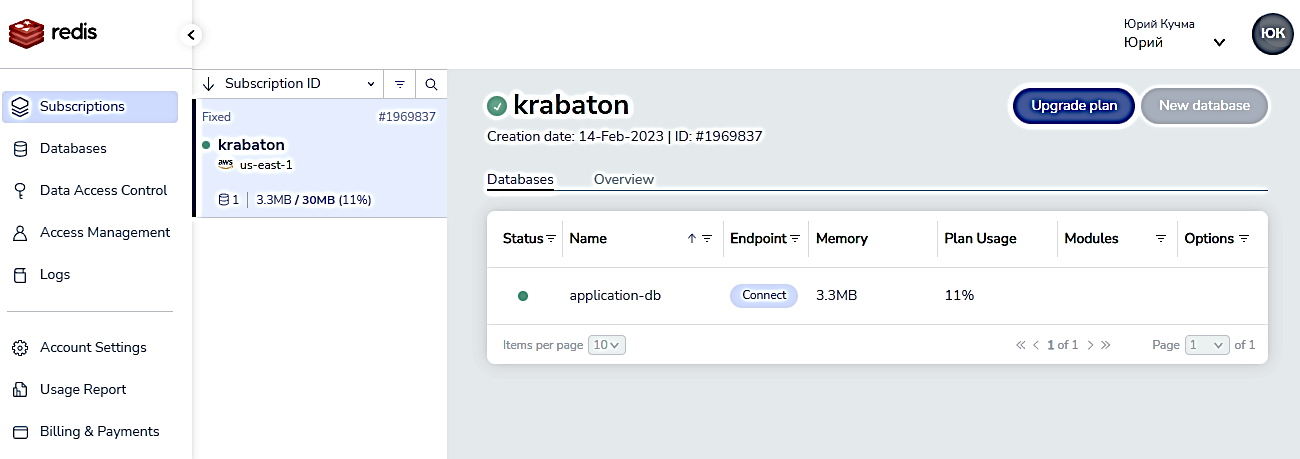
Замініть у .env файлі застосунку всі необхідні значення змінних на отримані від сервісу і виконайте необхідні міграції!

**Redis**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#redis)[RedisLabs](https://app.redislabs.com/#/) (https://app.redislabs.com/#/) - це хмарний сервіс для зберігання та управління даними в базі даних Redis. Сервіс пропонує низку інструментів і функцій, які дають змогу користувачам легко створювати, масштабувати та управляти базами даних Redis у хмарі.

Основна мета RedisLabs - надати користувачам простий і зручний спосіб використання Redis у хмарному оточенні, що може допомогти прискорити розробку застосунків і спростити їхнє масштабування. Він також надає різні інструменти та функції, які допомагають поліпшити продуктивність і безпеку Redis.

Виконаємо реєстрацію та створимо безкоштовний екземпляр бази даних на 30 Мб.



**Екземпляр бази даних Redis**

Сервіс надасть значення хоста, порт і пароль для користувача. Додамо значення змінних оточення:

REDIS\_HOST=

REDIS\_PORT=

REDIS\_PASSWORD=

Тепер, через появу пароля користувача, проведемо зміни коду в необхідних місцях нашого застосунку. Додамо змінну в клас налаштувань Settings.

**src/conf/config.py**

class Settings(BaseSettings):

sqlalchemy\_database\_url: str = 'postgresql+psycopg2://user:password@localhost:5432/postgres'

secret\_key: str = 'secret'

algorithm: str = 'HS256'

mail\_username: str = 'example@meta.ua'

mail\_password: str = 'password'

mail\_from: str = 'example@meta.ua'

mail\_port: int = 465

mail\_server: str = 'smtp.meta.ua'

redis\_host: str = 'localhost'

redis\_port: int = 6379

redis\_password: str = 'password'

cloudinary\_name: str = None

cloudinary\_api\_key: str = None

cloudinary\_api\_secret: str = None

class Config:

env\_file = ".env"

env\_file\_encoding = "utf-8"

settings = Settings()

Змінимо умови підключення до екземпляра Redis у нашій програмі в головному файлі main.py:

**main.py**

@app.on\_event("startup")

async def startup():

r = await redis.Redis(host=settings.redis\_host,

port=settings.redis\_port,

password=settings.redis\_password,

db=0,

encoding="utf-8",

decode\_responses=True)

await FastAPILimiter.init(r)

А також змінимо умови підключення до екземпляра Redis у класі авторизації Auth:

**src/services/auth.py**

class Auth:

...

r = Redis(host=settings.redis\_host, port=settings.redis\_port, password=settings.redis\_password, db=0)

Після всіх проведених операцій повертаємося до процесу розгортання застосунку.

**Розгортаємо застосунок**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#%D1%80%D0%BE%D0%B7%D0%B3%D0%BE%D1%80%D1%82%D0%B0%D1%94%D0%BC%D0%BE-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA)

**Аутентифікація**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#%D0%B0%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F)Встановіть консольну утиліту space.

iwr https://get.deta.dev/space-cli.ps1 -useb | iex

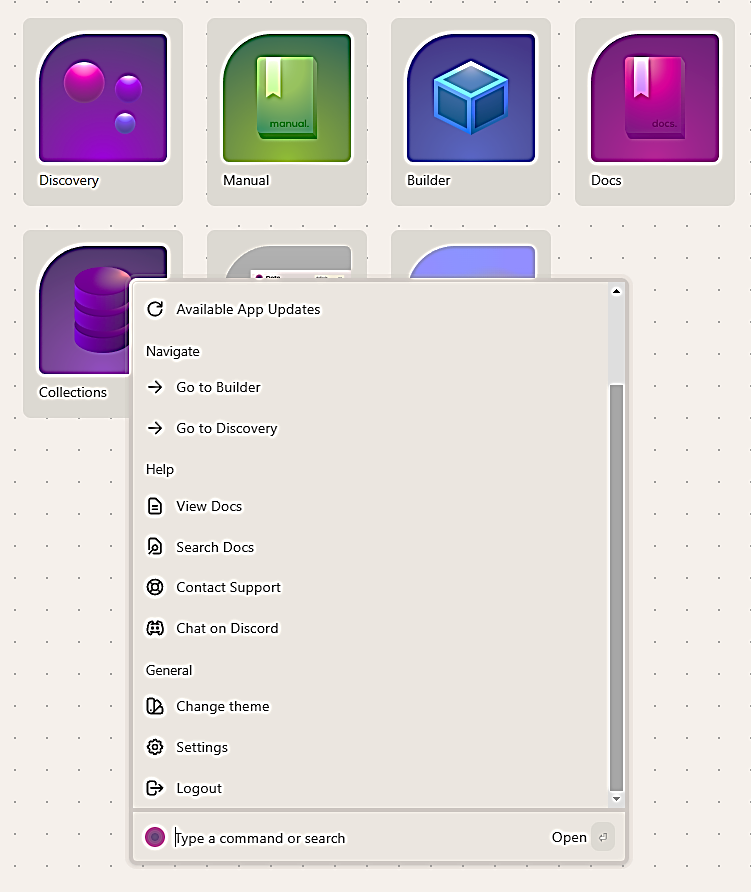
Виконайте аутентифікацію в сервісі командою

space login

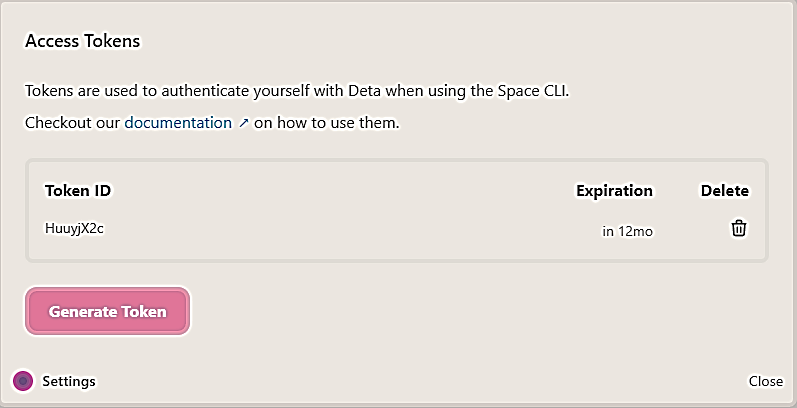
З'явиться запрошення ввести токен аутентифікації:

? Enter access token >

Щоб отримати токен доступу, увійдіть у панель приладів Space, відкрийте телетайп (командний рядок) і натисніть на неї, а далі натисніть 'Settings':



Відкриється модальне вікно Settings, де ви можете натиснути 'Generate Token', щоб згенерувати токен доступу:



Скопіюйте отриманий токен. Вставте його назад у підказку CLI. Після натискання клавіші Enter вас повинен зустріти повідомлення про успіх.

**Ініціалізація**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#%D1%96%D0%BD%D1%96%D1%86%D1%96%D0%B0%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F)Використовуйте команду, щоб створити новий проект за допомогою Space CLI

space new

Вам буде запропоновано ввести ім'я для вашого проєкту. Потім CLI спробує виявити будь-які існуючі застосунки у ваших локальних каталогах. Якщо знайдено, CLI покаже згенеровану конфігурацію для цих застосунків. Щойно ви підтвердите згенеровану конфігурацію, CLI створить новий проект у Builder і згенерує файл Spacefile у вашому локальному каталозі. Spacefile містить конфігурацію вашого застосунку, яка використовується Deta Space, щоб зрозуміти, який вигляд має ваш застосунок і як його запускати.

*# Spacefile Docs: https://go.deta.dev/docs/spacefile/v0*

v : 0

micros :

- name: solution\_rest\_api\_tests

src : .

engine : python3.9

primary : true

Крім того, інтерфейс командного рядка створив прихований каталог .space - він містить інформацію про проект, пов'язуючи його з вашим проєктом Builder. Цей каталог не повинен бути включений у ваш контроль версій і був автоматично доданий у ваш .gitignore. Після того як ви створили проєкт, ви можете переглянути його в застосунку Builder, який можна знайти на вашому дашборді Deta Space.

**Надсилання змін**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#%D0%BD%D0%B0%D0%B4%D1%81%D0%B8%D0%BB%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B7%D0%BC%D1%96%D0%BD)

Коли застосунок готовий, використовуйте команду Space CLI

space push

Ця команда підтвердить правильність вашого Spacefile, а потім упакує і завантажить усі необхідні файли в конвеєр збірки Deta. Виконання команди буде передавати логи на ваш локальний термінал протягом усього процесу, щоб ви знали, що відбувається, і могли усувати проблеми. Після завершення конвеєра збірки буде створено версію.

Версія - це повний пакет вашого застосунку в один момент часу. Нові версії створюються з кожним виконанням команди space push. Думайте про це як аналог комміту в системі git. Ви можете переглянути всі версії, відкривши свій проект у Builder і подивившись їх на вкладці "Develop".

**Запуск застосунку**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/deta#%D0%B7%D0%B0%D0%BF%D1%83%D1%81%D0%BA-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BA%D1%83)Виконайте запуск застосунку за допомогою інтерфейсу командного рядка Space. Використовуйте наступну команду:

space release

Все, деплой застосунку виконано. Ви можете переглянути URL для запущеної версії, відкривши свій проект у Builder і натиснувши кнопку "Open Builder Instance" на вкладці "Develop". Наш застосунок буде розташований за адресою <https://apprestapi-1-r6034909.deta.app/docs>

**Сервіс Fly.io**

[**Fly.io**](https://fly.io/) (https://fly.io/)- це сервіс, який надає інфраструктуру для запуску та розгортання застосунків у хмарі. Він дозволяє запускати віртуальні машини (VM) у різних частинах світу та автоматично масштабувати застосунки, що дає змогу забезпечити високу доступність і швидкодію.

Fly.io підтримує запуск застосунків різними мовами програмування, включно з Python, Ruby, Node.js, Go та іншими. Він також надає інструменти для автоматичного масштабування застосунків, що дає змогу швидко адаптуватися до збільшення навантаження і зберігати високу доступність.

Розберемо приклад деплою застосунку Django у сервісі Fly.io. Спочатку для системи Windows встановіть клієнт Fly.io за допомогою команди:

powershell -Command "iwr https://fly.io/install.ps1 -useb | iex"

Для інших операційних систем встановіть клієнт відповідно до [документації](https://fly.io/docs/hands-on/install-flyctl/) (https://fly.io/docs/hands-on/install-flyctl/)

Ми виберемо варіант розгортання застосунку за допомогою Docker. Сам процес досить простий і описаний у цьому розділі [**документації**](https://fly.io/docs/languages-and-frameworks/dockerfile/) (https://fly.io/docs/languages-and-frameworks/dockerfile/)**.**

Для нашого Django застосунку сформуємо Dockerfile.

**Dockerfile**

*# Docker-команда FROM вказує базовий образ контейнера*

*# Наш базовий образ - це Linux з попередньо встановленим python-3.10*

FROM python:3.10

*# Встановимо змінну середовища*

ENV APP\_HOME /app

*# Встановимо робочу директорію всередині контейнера*

WORKDIR $APP\_HOME

COPY poetry.lock $APP\_HOME/poetry.lock

COPY pyproject.toml $APP\_HOME/pyproject.toml

*# Встановимо залежності всередині контейнера*

RUN pip install poetry

RUN poetry config virtualenvs.create false && poetry install --only main

*# Скопіюємо інші файли в робочу директорію контейнера*

COPY . .

*# Позначимо порт, де працює програма всередині контейнера*

EXPOSE 8000

*# Запустимо нашу програму всередині контейнера*

CMD ["python", "instagram/manage.py", "runserver", "0.0.0.0:8000"]

Тепер потрібно виконати команду в корені проекту:

fly launch

Команда запуску повинна виявити ваш Dockerfile, щоб створити контейнер для деплою в сервісі. Після виконання команда fly launch запитає ім'я застосунку або запропонує згенерувати випадкове, що, можливо, буде краще, так воно точно буде унікальним. Виберіть регіон для розгортання. Потім команда запуску генерує файл fly.toml для вашого проекту з налаштуваннями.

В отриманий файл fly.toml додайте змінні оточення з файлу .env у розділ [env]. Як і у випадку з Deta Space, використовуйте хмарну базу даних для Postgres.

*# fly.toml file generated for lively-night-7091 on 2023-03-11T20:48:05+02:00*

app = "lively-night-7091"

kill\_signal = "SIGINT"

kill\_timeout = 5

primary\_region = "ams"

processes = []

[env]

SECRET\_KEY = "\*\*\*\*\*\*\*"

DB\_NAME = "\*\*\*\*\*\*\*"

DB\_USER = "\*\*\*\*\*\*\*"

DB\_PASSWORD = "\*\*\*\*\*\*\*"

DB\_HOST = "\*\*\*\*\*\*\*"

DB\_PORT = 5432

EMAIL\_HOST\_USER = "example@meta.ua"

EMAIL\_HOST\_PASSWORD = "password"

[experimental]

auto\_rollback = true

[[services]]

http\_checks = []

internal\_port = 8000

processes = ["app"]

protocol = "tcp"

script\_checks = []

[services.concurrency]

hard\_limit = 25

soft\_limit = 20

type = "connections"

[[services.ports]]

force\_https = true

handlers = ["http"]

port = 80

[[services.ports]]

handlers = ["tls", "http"]

port = 443

[[services.tcp\_checks]]

grace\_period = "1s"

interval = "15s"

restart\_limit = 0

timeout = "2s"

Потім файли fly.toml і .env додайте у файли .gitignore і .dockerignore.

fly.toml

\*\*/.env

NOTE

Можна також використовувати для секретних змінних команду

flyctl secrets set MY\_SECRET=romance

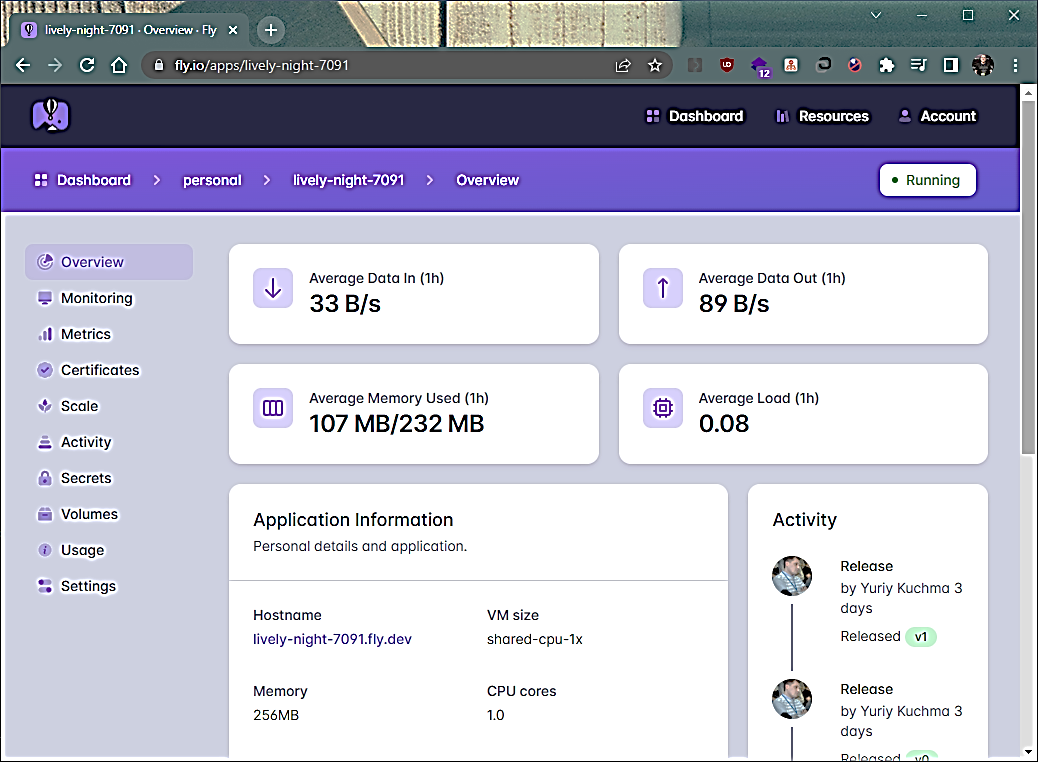
Тепер ми можемо розгорнути наш застосунок командою.

fly deploy

Після чого ваш контейнер буде створений і розгорнутий. Щоб подивитися розгорнутий застосунок у браузері, виконайте

fly open

Або можете перейти в [дашборд](https://fly.io/dashboard) (https://fly.io/app/sign-in?return\_to=%2Fdashboard) і знайти та відкрити посилання всередині.



**Fly.io dashboard**

**Сервіс Koyeb**

[**Koyeb**](https://www.koyeb.com/) (https://www.koyeb.com/) - це потужний і гнучкий інструмент для розробників, який полегшує розгортання і масштабування застосунків, дозволяючи їм зосередитися на розробці коду, а не на управлінні інфраструктурою.

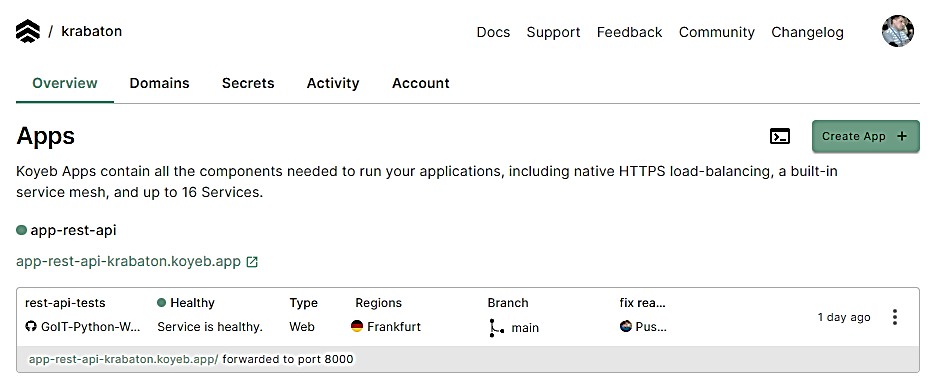
Має підтримку декількох мов програмування, включно з Python, Node.js, Go, Ruby, PHP та іншими. Дає змогу інтегруватись із сервісом GitHub. Koyeb забезпечує безпеку ваших даних за допомогою шифрування та інших заходів безпеки. Легкий у використанні, оскільки має простий та інтуїтивно зрозумілий інтерфейс, який дає змогу швидко створювати та управляти застосунками.

Під час реєстрації нам буде надано кредит у розмірі 5.5$ на місяць. Цей умовно-безкоштовний тариф включає 512 Мб оперативної пам'яті для вашого застосунку.

INFO

Get started for free! 🎉 Deploy two nano services for free with $5.5 of free credit on us every month.

Після реєстрації ми побачимо адміністративну панель (дашборд) з нашими застосунками, якщо, звісно, вони у вас уже є.



**Koyeb dashboard**

**Підготовка коду**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/koyeb#%D0%BF%D1%96%D0%B4%D0%B3%D0%BE%D1%82%D0%BE%D0%B2%D0%BA%D0%B0-%D0%BA%D0%BE%D0%B4%D1%83)Ми розглянемо процес розгортання застосунку з Github репозиторію. Але спочатку необхідно внести кілька змін до репозиторію з нашим кодом. Сервіс Koyeb має [набір репозиторіїв](https://github.com/koyeb?q=example&type=all&language=&sort=) (<https://github.com/koyeb?q=example&type>= all&language=&sort=) із прикладами для деплою. Для фреймворка FastAPI існує свій репозиторій [**example-fastapi**](https://github.com/koyeb/example-fastapi) (https://github.com/koyeb/example-fastapi)**.** Проаналізувавши репозиторій із прикладом, нам необхідно у свій репозиторій додати файл Procfile із наступним вмістом:

web: uvicorn main:app --port ${PORT:-8000} --host 0.0.0.0

Це команда, що запускає вебзастосунок за допомогою ASGI-сервера Uvicorn:

* uvicorn команда для запуску сервера Uvicorn;
* main:app тут вказуємо модуль Python і об'єкт, який буде використовуватися для застосунку. У цьому випадку використовується модуль з ім'ям main і об'єкт з ім'ям app;
* --port ${PORT:-8000} задаємо номер порту, який повинен прослуховувати сервер. Ми використовуємо змінну оточення PORT, і якщо вона не встановлена, то за замовчуванням використовується порт 8000.
* --host 0.0.0.0.0 вказуємо адресу хоста, до якого повинен прив'язатися сервер. У цьому випадку він прив'язується до усіх доступних мережевих інтерфейсів, що дає змогу отримати доступ до сервера з будь-якої IP-адреси.

Як бачимо, ця команда запускає вебзастосунок за допомогою Uvicorn, а, отже, пакет uvicorn повинен бути обов'язково доданий у файл requirements.txt.

fastapi

psycopg2-binary

python-jose[cryptography]

passlib[bcrypt]

python-multipart

libgravatar

fastapi-mail

redis

fastapi-limiter

cloudinary

sqlalchemy

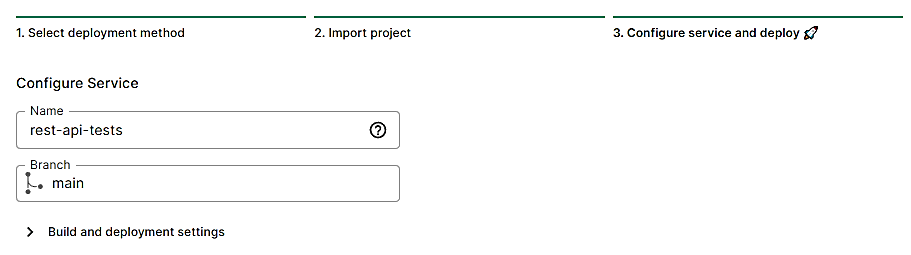
pydantic[dotenv]

uvicorn

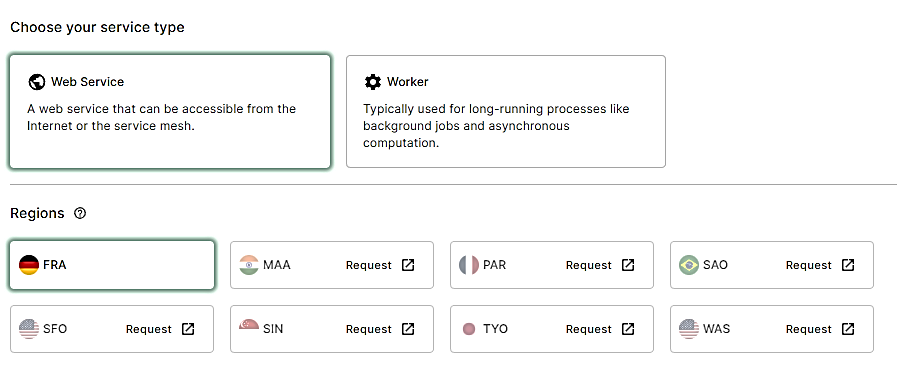
**Розгортаємо застосунок**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-14/module-14-02/koyeb#%D1%80%D0%BE%D0%B7%D0%B3%D0%BE%D1%80%D1%82%D0%B0%D1%94%D0%BC%D0%BE-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA)Натисніть кнопку "Create App". Виберіть варіант розгортання застосунку з Github. Додайте застосунок сервісу Koyeb у свій Github, щоб сервіс зміг бачити наші репозиторії. Після дозволу для сервісу бачити наші репозиторії на другому етапі Import project виберемо репозиторій із нашим проектом.

Ми потрапимо на сторінку конфігурації проекту. Насамперед нам потрібно вказати, з якої гілки репозиторію відбуватиметься розгортання застосунку. За замовчуванням це гілка main.

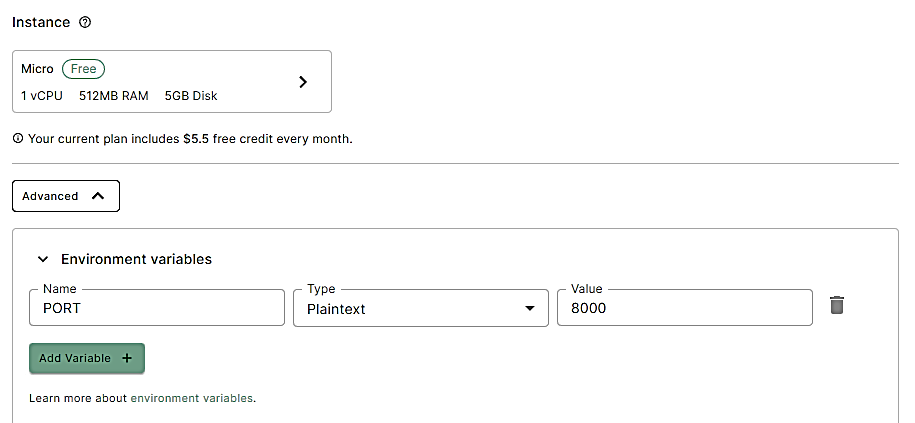


Вибираємо, що ми збираємося розгортати "Web Service" і регіон Frankfurt, для безкоштовного використання.

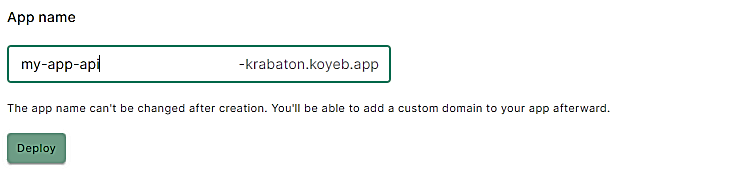


Вибираємо екземпляр застосунку "Micro", а у вкладці "Advanced", що розкривається, нам необхідно до існуючої змінної оточення "PORT" додати наші змінні оточення з файлу

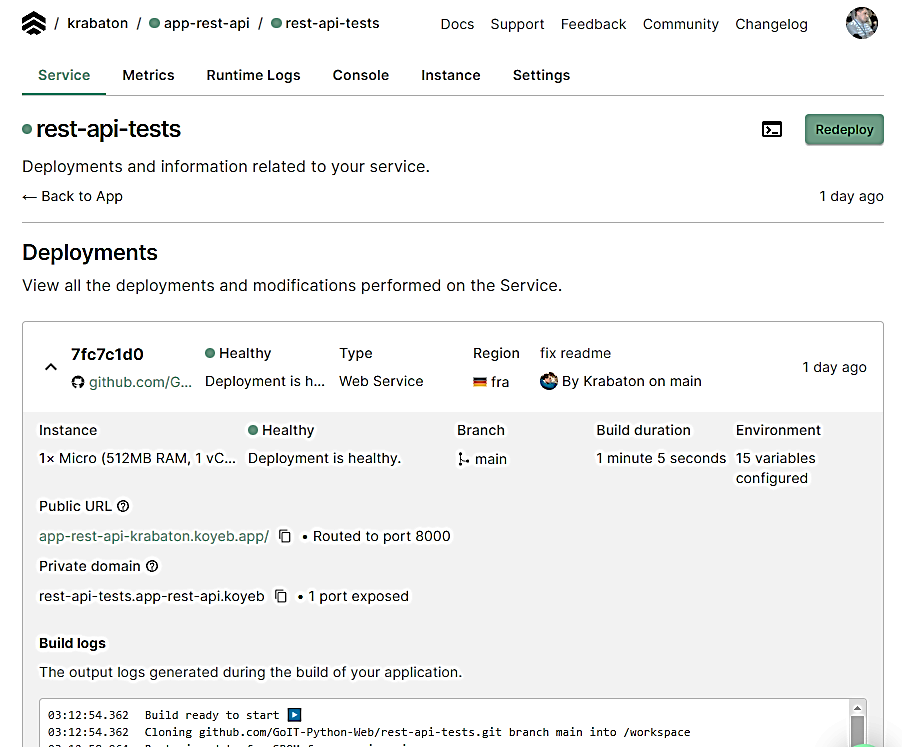
.env з актуальними значеннями.



Останнім етапом налаштування — ми вказуємо доменне ім'я для нашого застосунку. Все, наш застосунок готовий до деплою і ми можемо почати процес розгортання застосунку, натиснувши кнопку "Deploy".



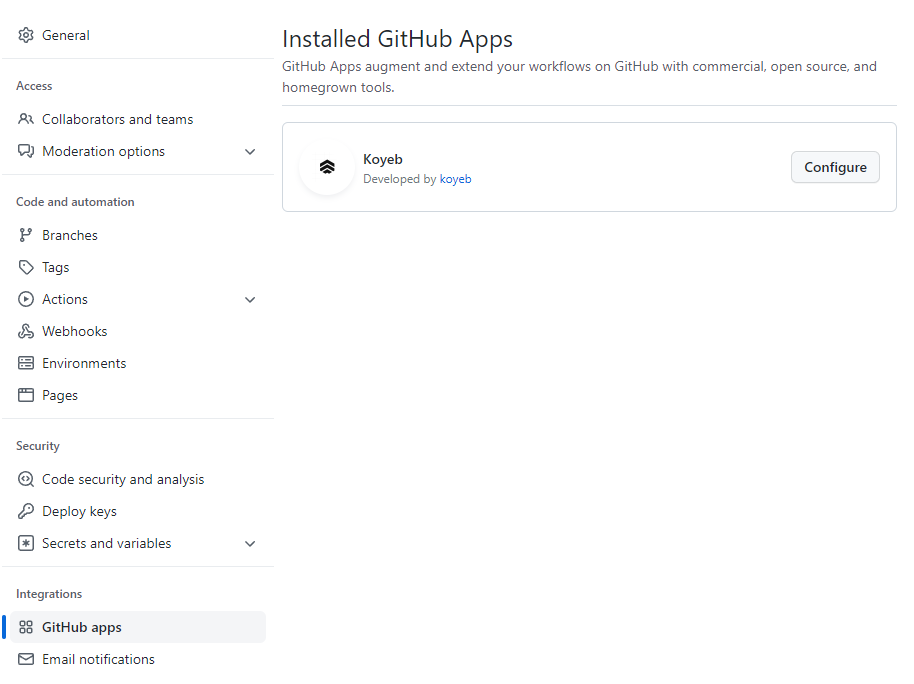
Ми запустимо процес, де можемо бачити результат деплою. Якщо він буде успішним, то в кінці ми побачимо в "Build logs" повідомлення "Build succeeded ✅". Якщо виникнуть помилки, уважно прочитайте та усуньте їх.



Після успішного деплою ми отримаємо робочий екземпляр нашого застосунку

h[ttps://app-rest-api-krabaton.koyeb.app](https://app-rest-api-krabaton.koyeb.app/docs)

. Головною особливістю тепер буде те, що при кожному git push або PR (Pull Request) у гілку main, деплой застосунку відбуватиметься автоматично. Це можливо через те, що ми додали в наш репозиторій додаток Koyeb.



Тепер розробку можна вести в окремій гілці, а всі остаточні зміни для деплою відправляємо через PR в гілку main.

INFO

Актуальна і детальна документація з розгортання застосунку FastAPI у сервісі Koyeb знаходиться за наступним [**посиланням**](https://www.koyeb.com/docs/deploy/fastapi) (https://www.koyeb.com/docs/deploy/fastapi)**.**

**Сервіс Replit**

[**Replit**](https://replit.com/) (https://replit.com/)- це онлайн-платформа, яка дозволяє користувачам створювати, запускати і розгортати різні проекти, включно з застосунками, вебсайтами, іграми та багато іншого, прямо в браузері. Основна мета сервісу - спростити процес розробки програмного забезпечення, зробивши його доступним для всіх, незалежно від рівня досвіду в програмуванні.

**Replit**надає користувачеві віртуальне середовище розробки (IDE), яке містить всі необхідні інструменти для написання, відлагодження та тестування коду. Користувачі можуть вибрати з різних мов програмування і фреймворків, включно з Python, Java, JavaScript, Ruby, PHP і багатьма іншими.

Однією з головних можливостей **Replit** є можливість спільної роботи над проектами в режимі реального часу. Це означає, що кілька розробників можуть працювати над одним проектом одночасно і бачити зміни, що вносяться іншими учасниками команди.

**Replit**також надає потужні інструменти для розгортання проектів, які дають змогу користувачам легко опублікувати свої застосунки та вебсайти в Інтернеті. Крім того, Replit має велику спільноту розробників, де користувачі можуть обмінюватися ідеями, ставити запитання та отримувати допомогу.

TIP

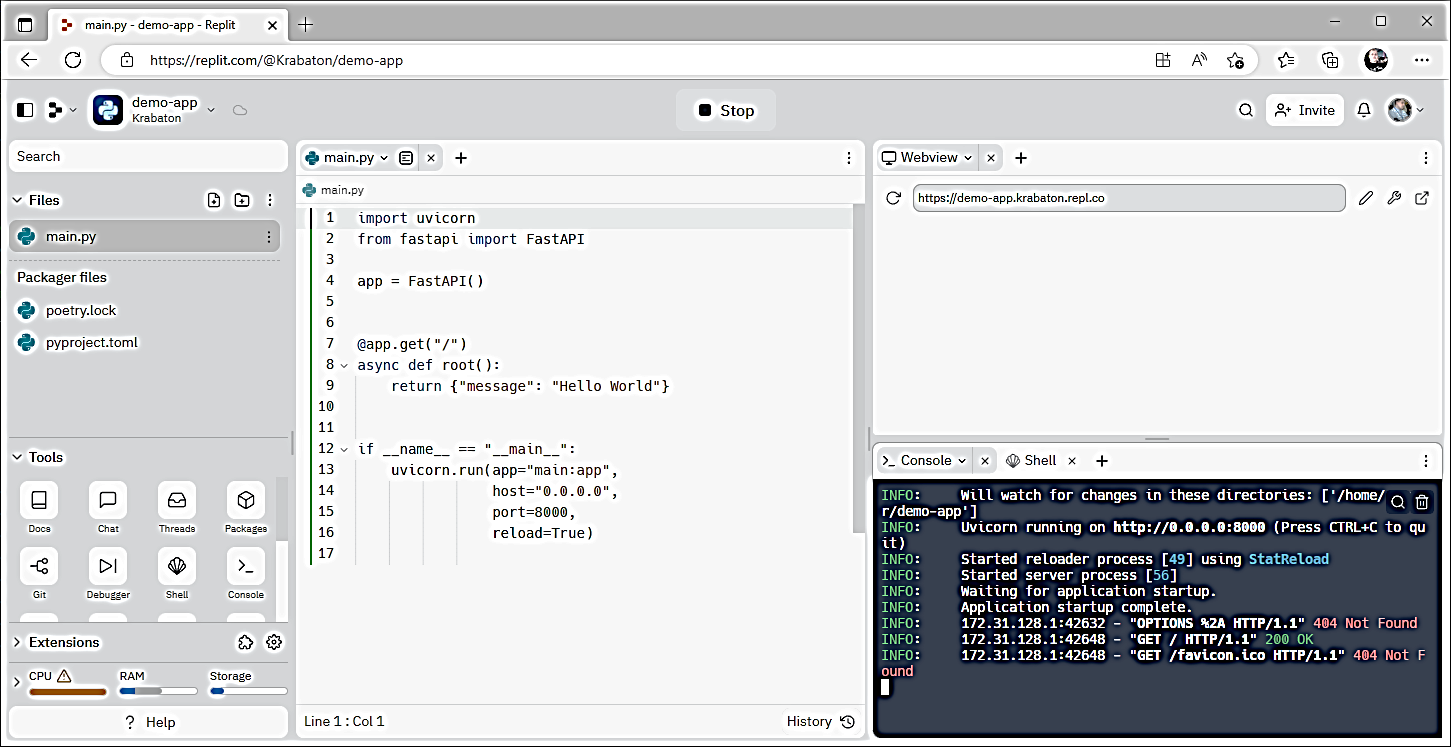
Загалом, Replit - це потужний інструмент для розробки програмного забезпечення, який робить процес розробки доступнішим і зручнішим для всіх.

Але, все ж таки, сервіс [**Replit**](https://replit.com/) (https://replit.com/) не можна вважати повноцінним сервісом для продуктових рішень, але в цілях розробки і для перевірки роботи прототипів він досить хороший. Оскільки надає можливість перевірити роботу вашого застосунку в робочому середовищі, максимально наближеному до реального.

Як ви помітили, всі живі приклади в конспекті використовували сервіс Replit. Як і для AWS Elastic Beanstalk, увесь код застосунку можна завантажити архівом.

Також можна виконати прив'язку до Git репозиторію за принципом Heroky або Koyeb. Але головна відмінність сервісу - те, що він має онлайн редактор для редагування коду застосунку у самому сервісі. Також перевагою сервісу є використання poetry для створення віртуальних оточень.

Фактично, це онлайн IDE для розробки ваших проектів з можливістю запуску застосунку в хмарі.



**Replit IDE online**

Сервіс має багатий функціонал, який постійно розширюється. Та має дуже хорошу і детальну [документацію](https://docs.replit.com/) (https://docs.replit.com/) для новачка. Рекомендуємо з нею ознайомитися, якщо збираєтеся використовувати сервіс для своїх проектів.

Під час розробки застосунку вам буде надано доменне ім'я 4-го рівня для демонстрації в інтернеті <https://demo-app.krabaton.repl.co/>. Домен буде як назва проекту та вашого **username** на сервісі.