🏘️ Real Estate Scraper Application – Cloud-Native

A modern, production-grade backend system for scraping and storing real estate listings using Python, FastAPI, PostgreSQL, Docker, and Kubernetes. Designed with observability, CI/CD, high availability, and disaster recovery in mind.

🎯 Project Goals

- Build a **scraper backend** for real estate listings (non-commercial, demo data).
- Provide an **API** for accessing and managing scraped data.
- Host with **cloud portability** in mind (AWS for now, Azure/GCP compatible).
- Ensure **resilience, scalability, and maintainability** using Docker, K8s, CI/CD, and backups.
- Serve **frontend and backend under the same domain** via a reverse proxy (nginx or ingress).

🔧 Tech Stack

- **Python 3.10+**, FastAPI
- **SQLAlchemy**, PostgreSQL
- **Docker**, Docker Compose
- **GitHub Actions** (CI/CD)
- **Kubernetes** (Minikube / AWS EKS / Azure AKS / GCP GKE)
- **Prometheus + Grafana (AGPL!)** (Monitoring)
- **AWS Free Tier** (current deployment)
- Disaster Recovery & High Availability ready
- **nginx / Ingress** for serving frontend + backend on one domain

---

📦 Project Structure

```
real-estate-scraper/
├── app/                    # FastAPI backend
│   ├── scrapers/           # scraping logic
│   ├── models/             # SQLAlchemy models
│   ├── api/                # API routes
├── frontend/               # React frontend app
├── tests/                  # unit & integration tests
├── Dockerfile
├── docker-compose.yml
├── kubernetes/             # K8s manifests incl. ingress
├── .github/workflows/      # GitHub Actions CI/CD
├── scripts/                # backup/restore scripts
├── requirements.txt
└── README.md
```

🚀 Features

- `/scrape` – trigger real estate scraping task (mock or live HTML)
- `/listings` – query stored listings with filters
- `/health`, `/metrics` – health & metrics endpoints
- Job scheduler (optional) for recurring scrapes
- Clean separation between scraping & API layer
- React frontend served through same domain via reverse proxy

---

🧪 DevOps & Cloud Features

- Dockerized microservice
- CI/CD pipeline (test → build → push → deploy)
- K8s-ready manifests (`deployment.yaml`, `service.yaml`, `hpa.yaml`, `ingress.yaml`)
- Prometheus metrics & Grafana dashboard
- Backup and restore scripts for PostgreSQL
- Simulated failure scenarios (for DR testing)
- AWS deployment via EC2/EKS and S3 for backups
- Portable to Azure & GCP

🛡️ High Availability & Disaster Recovery

- Replicated pods via `replicas: 2` in K8s
- Horizontal Pod Autoscaler (`hpa.yaml`)
- External database backup to S3 (`backup.sh` + cronJob)
- Simulated pod/database crash recovery guides

📈 Monitoring

- `/metrics` for Prometheus scraping
- Service metrics: request count, response times, uptime
- Grafana dashboard with alerts for scraper failures

🧹 Static Code Analysis (Python & React)

- `ruff` – ultra-fast linter (Python, replaces flake8, black, isort, etc.)
- `mypy` – static typing and type checking
- `bandit` – Python security analyzer
- `eslint` – JS/React/TS linter
- `prettier` – auto-code formatter
- `typescript` – type-safe frontend codebase

All tools are open-source and run locally or via GitHub Actions (CI/CD).

📑 To-Do / Roadmap

- □ Core scraper with BeautifulSoup/Playwright/Selenium
- □ Asynchronous FastAPI routes
- □ PostgreSQL schema for listings
- □ Docker Compose setup
- □ CI pipeline with test + lint + build
- □ Kubernetes manifests
- □ AWS deployment
- □ DR simulation (backup & restore)
- □ Monitoring and metrics
- □ React frontend integration
- □ Serve frontend + backend under single domain
- □ Helm & Terraform support (optional)

⚠ Legal & Ethical Disclaimer

This project uses **mocked or publicly available HTML** for demonstration purposes. It is designed strictly for **educational use** and does **not access or store any private or proprietary data**.

📦 Versioning and Release Notes

- We maintain a `CHANGELOG.md` file at the root of the repository to track all notable changes per release.
- Each release is tagged using **Semantic Versioning (MAJOR.MINOR.PATCH)**.
- GitHub **Releases** are created for every new version, linking to the changelog entries.
- Pull Requests and commit messages follow conventional commits style (`feat:`, `fix:`, etc.) to help generate changelog entries.
- This process ensures clarity and smooth collaboration in a team environment.

A link to the full changelog can be found here.
