Deployment Concept all.txt

Civic Data Lab Support Project

all.txt

Version
1.0.1
Authors
Rahkakavee Baskaran (&effect), Jan Dix (&effect)
Created at
January 30th, 2024
Updated at
March 19th, 2025
License
CC-BY 4.0

&effect

Content

1 Possible Obstacles

[redacted]

2 Architecture Recommendations

2.1 Authentication and Authorization using FastAPI Users

FastAPI Users[1] simplifies the implementation of user authentication and authorization in web applications built with FastAPI. It provides ready-to-use components for handling user registration, authentication, and token-based authorization, making it easier for developers to integrate user management functionality into their FastAPI projects. With support for OAuth2, JWT, and other authentication methods, FastAPI Users streamlines the process of securing and managing user access within web applications. We recommend using the cookie transport in combination with a database strategy. If the API should be used outside the browser the transport can be complemented using bearer scheme. Additionally, FastAPI Users allows to easily integrate HTTPX OAuth[2] to authenticate users with external services like Google or Microsoft. This could be an additional feature for paying customers.

2.2 Database Integration using SQLAlchemy and Alembic

SQLAlchemy[3] is a Python SQL toolkit and Object-Relational Mapping (ORM) library that facilitates interaction with relational databases. It provides a high-level API for database operations, allowing developers to express database queries using Python code rather than raw SQL. SQLAlchemy supports a variety of database backends, offers a powerful and flexible query language, and simplifies the mapping of database tables to Python objects for seamless integration in applications. SQLAlchemy can be comined with multiple dialects. We recommend using PostgreSQL using the psycopg (psycopg3) driver[4] [5].

Alembic[6] is a database migration tool for SQLAlchemy, designed to automate the process of evolving database schemas over time. It allows developers to manage database changes through Python scripts, making it easy to version and apply schema modifications while keeping track of database schema history.

---

[1] https://fastapi-users.github.io/fastapi-users/latest/

[2] https://frankie567.github.io/httpx-oauth/

[3] https://www.sqlalchemy.org/

[4] https://www.psycopg.org/psycopg3/docs/

[5] https://docs.sqlalchemy.org/en/20/dialects/postgresql.html#module-sqlalchemy.dialects.postgresql.psycopg

[6] https://alembic.sqlalchemy.org/en/latest/

2.3 Dependency Management using Poetry

Python dependency management involves specifying and organizing project dependencies, historically done through tools like pip and a requirements.txt file. However, using requirements.txt alone can lead to issues with version conflicts. Therefore, we recommend using Poetry[7]. Poetry simplifies dependency management by combining package and project configuration in a single file, ensuring consistency across development, testing, and deployment environments, while also offering features like semantic versioning and dependency resolution, enhancing the overall development experience. Additionally, Poetry allows developers to define multiple dependencies groups. We recommend working with a main, the default, and a development group.[8] Development dependencies are only installed in the local environment.

2.4 Code Quality using Linter, Static Type Checker and Git Hooks

Black[9] and Flake8[10] are linters to enforce a unified code format in the project. Black and Flake8 can be installed as development dependencies and can be directly used in all popular development environments (e.g.: PyCharm, Visual Studio Code).

Mypy[11] is a static type checker for Python that allows developers to add type annotations to their code and catch potential errors during development by analysing and enforcing type correctness.

Pre-commit helps managing and maintaining pre-commit hooks for code repositories. These hooks are scripts or commands that run automatically before each commit, allowing tasks like code formatting, linting, and other checks to ensure code quality and consistency.

We recommend combining all these tools to improve the developer experience and ensure a certain code quality on the project. We recommend installing the tools as development dependencies and as pre-commit hooks. Additionally, we recommend installing isort[12] to sort the module imports. The selection is based on our experience and there are other tools that have a similar purpose and will work in a similar way.

2.5 Testing using Pytest

The purpose of writing tests is to systematically verify that your code functions correctly, ensuring that it behaves as expected under different scenarios and edge cases. The tests can be constantly enhanced when errors and edge cases are detected. Pytest[13] is a testing framework offering a simple and readable syntax, making it easy to write and maintain tests.

---

[7] https://python-poetry.org/

[8] https://python-poetry.org/docs/managing-dependencies/#dependency-groups

[9] https://black.readthedocs.io/en/stable/

[10] https://flake8.pycqa.org/en/latest/

[11] https://mypy-lang.org/

[12] https://pycqa.github.io/isort/

[13] https://docs.pytest.org/en/8.0.x/

By creating and running tests with Pytest, developers can catch and address bugs early in the development process, improve code reliability, and facilitate code maintenance and refactoring with confidence.

3 Hosting Concepts

3.1 Docker

Docker[14] is a platform for developing, shipping, and running applications in containers, which are lightweight, portable, and self-sufficient units that encapsulate everything needed to run the application. A Dockerfile is a script defining the steps and configuration for building a Docker image, specifying the application's environment, dependencies, and execution instructions. The Dockerfile packs the Python runtime and all dependencies to run the API. A Docker image is a lightweight, standalone, and executable package that encapsulates all the necessary components, including code, runtime, libraries, and system tools, to run a software application. A Docker image can usually be executed in any Docker runtime on any operating system that uses the same underlying CPU architecture (e.g.: amd64). Hence, developers can develop locally on a MacBook, but the image can also be started on a Debian server. A Docker container is a runnable instance of a Docker image, encapsulating an application along with its dependencies and providing an isolated and consistent execution environment.

Docker containers can be started in several ways in server environments. Docker can be used in production by directly running containers using the Docker runtime, where individual containers are managed and orchestrated manually, suitable for smaller-scale deployments or simpler use cases. Docker Swarm Mode[15] is a built-in orchestration tool in Docker that allows clustering and managing a group of Docker hosts, providing native support for service discovery and load balancing, making it a lightweight and easy-to-use solution for orchestrating containers in production. Kubernetes[16] is a powerful, open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications, providing advanced features like declarative configuration, automated scaling, and extensive ecosystem support, making it suitable for complex and large-scale production environments.

We recommend using Docker Swarm Mode as it provides useful features like service discovery and load balancing but is relatively easy to manage for a small team.

[14] https://www.docker.com/

[15] https://docs.docker.com/engine/swarm/

[16] https://kubernetes.io/

3.2 Provisioning Infrastructure with Terraform

Terraform[17] is an open-source infrastructure as code (IaC) tool used for provisioning and managing cloud infrastructure resources in a declarative and version-controlled manner. The Terraform state represents the current state of deployed infrastructure and is used to track resource attributes and manage changes. Providers are plugins that interface with different cloud or infrastructure platforms, allowing users to define and manage resources across various environments using a consistent Terraform configuration language. Both Scaleway[18] and Hetzner[19] provide and maintain their own Terraform providers directly interacting with their APIs.

We recommend using Terraform to configure the infrastructure, like virtual machines, databases, etc., in a reproducible way.

3.3 Configuring Infrastructure with Ansible

Ansible[20] is an open-source automation tool that simplifies configuration management, application deployment, and task automation in IT environments through a declarative language and agentless architecture. Ansible modules extend the capabilities of Ansible by providing pre-built, reusable components that enable users to interact with various systems, services, and resources, allowing for seamless automation across diverse infrastructure components.

We recommend using Ansible installing dependencies, Docker runtime, and configuring the virtual machines (e.g.: setting up Docker Swarm Mode).

3.4 Deployments using Stack Deploy

A Compose file can be used to deploy application to a Docker runtime running in Swarm Mode. The file describes the application's structure and settings in a simple YAML file. This file acts as a blueprint, defining how the different parts of the application should run and communicate. The file can also be tested locally using Docker Compose.

When the user deploys this configuration to Docker Swarm, the platform takes care of distributing and managing the application across multiple servers, ensuring it runs reliably and efficiently. This simplifies the deployment process, allowing non-technical individuals to describe their application's requirements without diving into technical details.

We recommend using Stack Deploy in a combination with Docker Context allowing us to directly start applications from GitLab workers.

[17] https://www.terraform.io/

[18] https://registry.terraform.io/providers/scaleway/scaleway/latest/docs

[19] https://registry.terraform.io/providers/hetznercloud/hcloud/latest/docs

[20] https://www.ansible.com/

3.5 Optional: Monitoring and Alerting

We recommend using the Grafana stack to monitor the infrastructure and optionally also your application. Grafana[21] is an open-source dashboard for monitoring and observability, providing a customizable and feature-rich interface to visualize and analyse metrics, logs, and other data. Prometheus[22] is a monitoring and alerting toolkit designed for collecting and storing time-series data, enabling the retrieval and analysis of detailed performance metrics from various systems. Loki[23] is a log aggregation system that complements Prometheus, allowing efficient storage and querying of logs while seamlessly integrating with Grafana for comprehensive observability across metrics and logs. Scaleway automatically stores system related metrics and logs, like CPU time, memory usage, etc., in their internal cockpit which is based on the Grafana stack.[24]

Additionally, we recommend to setup an error monitoring and tracking tool to help you identifying and resolving issues in your applications. You are automatically informed if your application runs into any errors. Sentry[25] and Honeybadger[26] both provide a free tier for a single developer. The team tier has a similar price. Both providers provide an easy integration for FastAPI. However, we recommend Sentry as it provides a larger variety for different Python tools and more programming languages.

Furthermore, we recommend to setup an external downtime monitor to keep track of your application uptime. We recommend using UptimeRobot[27] because the free tier is sufficient for small teams.

3.6 Optional: Managed Database

We recommend using a managed databased. A managed database is a database service provided by a cloud provider that handles administrative tasks such as setup, maintenance, scaling, and backups, allowing users to focus on application development rather than database management. The advantages of a managed database are simplified operations, automatic updates, enhanced security, and scalability, reducing the operational overhead for businesses while ensuring efficient and reliable database performance.

3.7 Optional: Virtual Private Network

A Virtual Private Network (VPN) is a technology that establishes a secure and encrypted connection over the internet, allowing users to access a private network. It is commonly used to create a secure connection between remote users and corporate networks, ensuring confidentiality and protecting data from unauthorized access or interception. We recommend

[21] https://grafana.com/grafana/

[22] https://prometheus.io/

[23] https://grafana.com/products/cloud/logs/

[24] https://www.scaleway.com/en/docs/observability/cockpit/

[25] https://sentry.io/welcome/

[26] https://www.honeybadger.io/

[27] https://uptimerobot.com/pricing/

using Tailscale[28]. Tailscale allows to create a direct connection to virtual machines without opening any ports. Additionally, Tailscale can be used to authenticate and authorize SSH connections. However, the premium tier costs $18 per user per month. Hence, we recommend implementing a VPN at later stage.

4 CI/CD Pipeline

The Continuous Integration/Continuous Deployment (CI/CD) pipeline is an automated process in software development that combines continuous integration, where code changes are regularly integrated and tested, with continuous deployment, automating the delivery of applications to production environments. It streamlines the development lifecycle, enhancing collaboration, code quality, and the speed at which software updates can be delivered to end-users. The CI/CD pipeline implements the hosting concept. It combines testing of code, the provisioning of infrastructure elements and the deployment of the services to different environments. Each of these steps is separated into a single stage. Each stage depends on the execution of the previous stage to ensure that only working code is shipped to the production environment.

4.1 Build Stage

During the build stage the Docker image is built and uploaded to a *Docker registry*. The registry allows to keep different versions of the image, caching for faster build times, and easy rollbacks to previous versions of the application in the case of an incident or problem. We recommend using the built-in GitLab container registry because it allows to keep code and images in the same place, and it does not create any additional costs.[29]

4.2 Testing Stage

During the testing stage the code is tested using *pytest* based on pre-defined tests. The tests are executed using the Docker image that is built in the previous stage.

4.3 Development Deployment Stage

During the development deployment stage the Docker image is pulled on the development environment and a container is started. This step can be automated for a certain branch, e.g.: develop (see Git Flow), or alternatively manually executed. In the beginning, we recommend to manually execute this step. This step requires the build stage successfully executed. Additionally, we can also configure this step to be dependent on the testing stage.

[28] https://tailscale.com/

[29] https://docs.gitlab.com/ee/user/packages/container_registry/

4.4 Acceptance Deployment Stage (Optional)

During the acceptance deployment stage the Docker image is pulled on the acceptance environment and a container is started. The acceptance stage is optionally and can be used to verify the release before deploying to the production environment. Alternatively, the stage can also be used to release beta features that should only be available to smaller group of testers. This step can be automated for a certain branch, e.g.: main (see Git Flow), or alternatively manually executed. In the beginning, we recommend to manually execute this step. This step requires the build stage successfully executed. We recommend configuring this step to be dependent on the testing stage.

4.5 Production Deployment Stage

During the production deployment stage, the Docker image is pulled on the production environment and a container is started. This step can be automated for a certain branch, e.g.: main (see Git Flow), or alternatively manually executed. In the beginning, we recommend to manually execute this step. This step requires the build stage successfully executed. We recommend configuring this step to be dependent on the testing stage.

5 Cost Estimation

5.1 Scaleway

Scaleway is a cloud computing service provider based in France, offering a range of cloud services, including virtual servers, storage, and networking solutions. Known for its competitive pricing and flexibility, Scaleway targets developers, startups, and enterprises seeking scalable and cost-effective cloud infrastructure. The offered services are GDPR conform, and the data centers are in Paris (France), Amsterdam (Netherlands), and Warsaw (Poland). A data processing agreement (DPA) is automatically in effect and can be downloaded from the website.[30]

Service	Costs in € (per Month)
Virtual Machine Production Environment (PLAY2-MICRO, 4 vCPUs, 8 GB RAM, PARIS 1)	44.06€
Virtual Machine Acceptance Environment (PLAY2-NANO, 2 vCPUs, 4 GB RAM, PARIS 1)	24.35€

[30] https://www.scaleway.com/en/contracts/

Virtual Machine Development Environment (PLAY2-NANO, 2 vCPUs, 4 GB RAM, PARIS 1)	24.35€
Database Production Environment (DB-PLAY2-NANO, 2 vCPUs, 4 GB RAM, Standalone Mode, PARIS)	36.50€
Database Acceptance and Development Environment (DB-PLAY2-PICO, 1 vCPUs, 2 GB RAM, Standalone Mode, PARIS)	21.97€
Total	**151.32€**

5.2 Hetzner

Hetzner is a German web hosting and data center company providing a variety of hosting services, dedicated servers, and cloud solutions. Known for its reliable infrastructure and competitive pricing, Hetzner caters to a broad range of clients, from individual developers to large enterprises, offering hosting solutions to meet diverse needs. The offered services are GDPR conform, and the data centers are in Nuremberg (Germany), Falkenstein (Germany), Helsinki (Finland), Ashburn (Virginia, USA), and Hillsboro (Oregon, USA). We recommend hosting the application in European data centers. A DPA can be separately signed.[31]

Service	Costs in € (per Month)
Virtual Machine Production Environment (CPX31, 4 vCPUs, 8 GB RAM)	13.60€
Virtual Machine Acceptance Environment (CX21, 2 vCPUs, 4 GB RAM)	5.35€
Virtual Machine Development Environment (CX21, 2 vCPUs, 4 GB RAM)	5.35€
Database All Environments* (VSERVER MC60, 8 vCPUs, 32GB RAM)	54.00€

[31] https://docs.hetzner.com/general/general-terms-and-conditions/data-privacy-faq

Total	**78.30€**

* The database is not comparable to the managed database instance offered by Scaleway. Hetzner offers managed servers for webhosting[32]. These offers include a managed database. However, this database does not provide fine-graded security options like creating multiple users or restricted users with read-only access. Additionally, the database is also reachable from the internet and cannot be used with private networks offered as part of the cloud offering.

5.3 Recommendation

We recommend using Scaleway, because it provides a variety of managed services like managed databases, transactional emails, and monitoring. This allows the all.txt team to focus on the product instead of focussing on the infrastructure. Scaleway has significantly higher entry costs, but Hetzner does not provide any services beyond virtual machines. The

6 Git Flow

We recommend to early adopt and define a way to structure the work with Git. As all.txt is product-focused organisation, we recommend using Git Flow. Git Flow is based on Vincent Driessen's branching model.[33] It defines a set of branching conventions and best practices. It introduces a structured workflow with two main branches, "**main**" for production-ready code and "**develop**" for ongoing development. Feature branches, release branches, and hotfix branches are created to facilitate organized feature development, release preparation, and bug fixes, making it easier for teams to collaborate and manage the software development life cycle. Daniel Kummer's cheatsheet provides a good introduction and overview on the different operations.[34]

[32] https://www.hetzner.com/managed-server/

[33] https://nvie.com/posts/a-successful-git-branching-model/

[34] https://danielkummer.github.io/git-flow-cheatsheet/index.html

About &effect:

Our goal is to make data science an integral part of decision-making in the public and social sector. To this end, we develop impact-oriented data products at the intersection between social sciences, data science and software development.

Contact:

Jan Dix

jan.dix@and-effect.com