Build Documentation: Building Information Enhancer (AMOS SS 2024)

Building Information Enhancer (AMOS SS 2024)

License MIT | languages 9 | last commit yesterday | issues 14 open

About the Project

Welcome to the official repository for the `CODE.ING` group, developing an open-source `Building Information Enhancer` software for the `BUILD.ING` partner as a part of the `AMOS 2024` project.

The `BCI Building Information Enhancer` is a platform for individual building owners or industry professionals, allowing them to access information about specific addresses, locations, or regions. This information can be used for a variety of applications, from sustainability certifications for buildings, over calculating the solar power potential, up to aiding in district planning. In general, the BCI building information enhancer offers significant benefits for various stakeholders in the property market.

For our project mission, the team agreed to create an MVP for the BCI Building Information Enhancer, the core functionality will be displaying data from a fixed number of sources, including satellite images, charging stations, and data needed for sustainability certification. Our goal is to build a practical and scalable tool that can grow with our users' needs. Finally, to read about the architecture of our service visit our GitHub Wiki Pages.

Setup & Deployment

`BCI Building Information Enhancer` project is integrated with a full CI-CD pipeline and thus deployed automatically for public use as a BCI website using servers provided by the industry partner. However, production servers might be turned off in the future, so it is also possible to deploy a production-ready system on your local machine using the provided Docker compose file.

Deployment using Docker

To deploy the latest release of the project on your local machine using Docker Engine, follow the instructions below:

1. Clone this repository (https://github.com/amosproj/amos2024ss04-building-information-enhancer).
2. Before deploying the project be sure you have installed the Node.js (>= 20.12.2) and a running Docker Engine.
3. Run the `npm run deploy` command in the root folder of the repository. *
4. Connect to the frontend at port 80 (`localhost:80`).

* This command pulls the newest public release of the images, if you want to build your local files, use `npm run deploy:build` instead.

💻 Developement Deployment

In order to deploy and/or develop one or more services, follow the instructions below:

Setup

1. Clone this repository (https://github.com/amosproj/amos2024ss04-building-information-enhancer).
2. Before deploying the project make sure you have installed the Node.js (>= 20.12.2), Dotnet SDK (>= 8.0.3) and a running Docker Engine.
3. Run the `npm run setup:<os>` command to download all necessary packages and build the services, replacing `<os>` with `windows`, `linux` or `macos` based on your operating system.

Developement

By following the micro-service architecture, each of the services can be deployed and developed separately. However, some of the services depend on each other. Follow instructions below to develop a specific service. Finally, the recommender way to deploy individual services is the docker images.

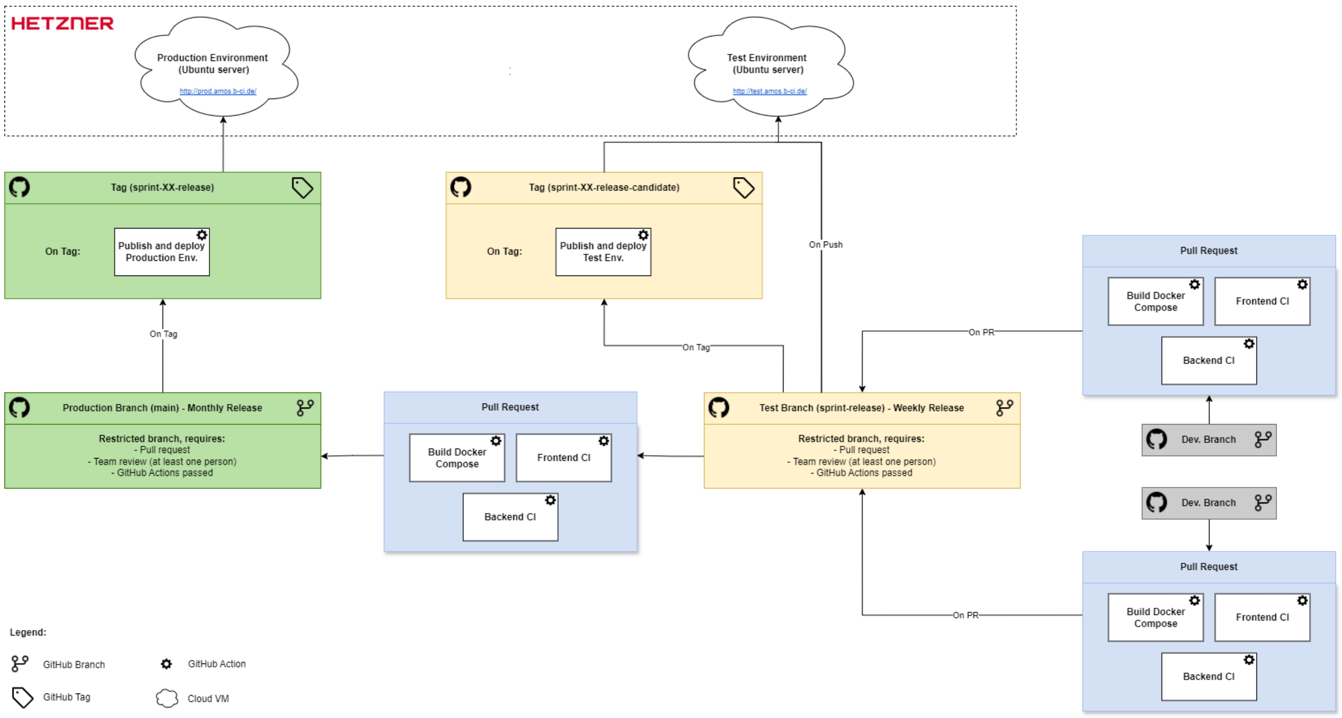
- **Frontend**: To develop the frontend service, one can run the `npm run deploy:frontend:dev` to deploy the VITE development server (with hot reloading). The code is located in the `frontend` folder. At a minimum, the frontend depends on the API Gateway and the MongoDB metadata database. If you want to query the datasets, the API Composer and the Geospatial DB needs to be deployed too. To deploy the service as a container use `docker compose up --build -d frontend`
- **API Gateway**: To develop the C# API Gateway, first make your changes in the `backend/API-Gateway` folder and run inside of it the `dotnet build` command to build the executables. Finally, to run the Gateway run type the `dotnet run` command. The API Gateway can run on its own, however, to use one of its API endpoints, one needs to deploy the corresponding service (for more information about the endpoints, read our System Architecture wiki page). To deploy the service as a container use `docker compose up --build -d api-gateway`.
- **API Composer**: Similarly to the API Gateway, go to the corresponding folder in the `backend` directory, make the changes, and run `dotnet build` and `dotnet run`. To deploy the service as a container use `docker compose up --build -d api-composer`.
- **Data Pipeline**: Similarly to the API Gateway, go to the corresponding folder in the `backend` directory, make the changes, and run `dotnet build` and `dotnet run`. To deploy the service as a container use `docker compose up --build -d datapipeline`.
- **Metadata Database**: To deploy the MongoDB database one needs to use the Docker `docker compose up --build -d metadata-database` command in the root folder. In order to change the metadata,

edit the `init-db.js` file inside `backend/metadata-database` folder.
- **Geospatial Databases**: To deploy the Geospatial databases one needs to use the Docker `docker compose up --build -d sql-database` command in the root folder.

👥 The Team

🔗 CI-CD Pipeline Architecture

To follow the agile development principles (thus making the deployment quicker, easier, and fully automatic) we have developed a CI/CD pipeline visible in the diagram below.


HETZNER
Production Environment (Ubuntu server)
http://prod.amos.b-ci.de/
Test Environment (Ubuntu server)
http://test.amos.b-ci.de/
Tag (sprint-XX-release)
On Tag:
Publish and deploy Production Env.
Tag (sprint-XX-release-candidate)
On Tag:
Publish and deploy Test Env.
On Push
On Tag
On Tag
On PR
On PR
Production Branch (main) - Monthly Release
Restricted branch, requires:
- Pull request
- Team review (at least one person)
- GitHub Actions passed
Pull Request
Build Docker Compose
Frontend CI
Backend CI
Test Branch (sprint-release) - Weekly Release
Restricted branch, requires:
- Pull request
- Team review (at least one person)
- GitHub Actions passed
Dev. Branch
Dev. Branch
Legend:
GitHub Branch
GitHub Action
GitHub Tag
Cloud VM

▶ GitHub Actions

We are taking full advantage of the following GitHub Actions workflows:

- Pull requests to the restricted branches (`main` and `sprint-release`):
  - Frontend CI - Runs linting and test checks for the frontend service.
  - Backend CI - Runs linting, formatting, and test checks for the backend service.
  - Build Docker Compose - Test if the Build of the whole Docker Compose file works.
- On new `sprint-XX-release` or `sprint-xx-release-candidate` tags:
  - Publish and deploy Production Env. - Build and publish the newest Docker images and deploy them to the production environment Ubuntu server.
  - Publish and deploy Test Env. - Build and publish the newest Docker images and deploy them to the test environment Ubuntu server.

- On push to the `sprint-release` branch
  - Publish and deploy Test Env. - Build and publish the newest Docker images and deploy them to the test environment Ubuntu server.

🚀 Automatic Deployment

For the deployment of our service (both production and testing), we are using Ubuntu VMs hosted by the Hetzner provider. When a deploy action is started, all Docker Images are built and published to the `ghcr.io` container images repository of the `amosproj/amos2024ss04-building-information-enhancer` project. Then, an SSH connection is made, and the appropriate server pulls and deploys the newest container images using the Docker Compose file. The servers are restricted to only allow incoming connections for HTTP, HTTPS, and SSH protocols.

🖌 Sprint Release Candidate

During each sprint development, new features are created on separate development branches and pull-requested to the `sprint-release` branch. This branch is restricted to only accepting pull requests, including the need for all passed actions and at least a single peer review. When a PR is merged into the `sprint-release` branch, the workflow for deployment into the test environments is run. This is done to minimize the surprises and test the software inside the Ubuntu servers, before the official releases.

Additionally, the Release Manager creates a new `release-candidate` tag, automatically starting the aforementioned deploy workflow to the test environment. Therefore, the `sprint-release` branch contains the newest and latest changes to the product and is tagged as a `sprint-XX-release-candidate` each sprint. The latest test environment can be viewed here.

One downside of this setup is that when new changes are pushed to the `sprint-release` branch before the sprint-candidate is released, they can overwrite the tag and push the newest changes to the test environment. This was done by purpose in order to allow the software developers to test their newest changes inside the Ubuntu environment, without needing to re-run the CI-CD pipeline and tagging the candidate.

💻 Production Release

The production system is located in the `main` branch. Similarly to the release candidate, when a new `sprint-xx-release` tag is created, an appropriate workflow is called and the newest code is deployed to the production server. Again, the `main` branch is restricted to only accepting pull requests, including the need for all passed actions and at least a single peer review. The production system can be viewed here.