SeisLabData

Documentação de desenvolvimento

Equipa NaturalGIS

Índice

Versão online

A versão atualizada deste documento está disponível em https://naturalgis.github.io/seis-lab-data/pt/development/.

1. Desenvolvimento

Este documento é o entregável de projeto *D03 - Documentação orientada ao programador sobre como configurar o projeto*. Contém informação sobre como configurar um ambiente de desenvolvimento local adequado para trabalhar no projeto SeisLabData.

Este projeto é composto por múltiplos serviços, que são orquestrados com `docker compose`. O ficheiro `docker/compose.dev.yaml` contém as instruções adequadas para desenvolvimento.

Dica

Quando a *stack* de desenvolvimento do Docker estiver ativa e em execução, execute os comandos docker compose com esta incantação:

```
docker compose -f docker/compose.dev.yaml <docker-command> <service-name>
```

Isto facilita o ajuste dos comandos ao âmbito deste projeto.

Os serviços mais relevantes são:

- `webapp` – a aplicação web principal, implementada com starlette, sqlmodel, jinja e datastar.
- `processing-worker` – serviço que executa a maior parte do processamento e das modificações à BD. É um *worker* dramatiq.
- `message-broker` – uma instância redis que gere a passagem de mensagens entre a webapp e o processing worker.
- `web-gateway` – uma instância traefik que atua como *reverse proxy* para o sistema.
- `auth-webapp` – uma instância authentik que trata da autenticação de utilizadores.
- `caddy-file-server` – uma instância caddy que serve datasets locais via HTTP.

1.1 Configuração do ambiente

Comece por obter os datasets de exemplo que foram disponibilizados pelo cliente. Estes encontram-se no ficheiro

`ipma/2025-marine-data-catalog/sample-data/20251125_datasample01_restored_data.tar.gz`

que está disponível na plataforma interna de base de conhecimento.

Crie uma diretoria base para os datasets do projeto (por exemplo em `~/data/seis-lab-data`), obtenha o arquivo e extraia-o dentro desta diretoria:

```
mkdir -p ~/data/seis-lab-data
cd ~/data/seis-lab-data

# obtenha o arquivo tar para esta dir e extraia-o
tar -xvf 20251125_datasample01_restored_data.tar.gz

# remova o arquivo após a extração
rm 20251125_datasample01_restored_data.tar.gz
```

Deverá obter algo semelhante a isto (listagem abreviada):

```
ricardo@tygra:~/data/seis-lab-data/$ tree -L 4
.
└── prr_eolicas
    └── base-final
        └── surveys
            └── owf-2025
```

Agora, clone este repositório localmente:

```
cd ~/dev  # ou onde preferir guardar o código

git clone https://github.com/NaturalGIS/seis-lab-data.git
cd seis-lab-data
```

Para simplificar a montagem da diretoria de dados dentro dos serviços docker, o projeto assume que existe uma diretoria `sample-data` na raiz do repositório. Como tal, crie um link simbólico a apontar para a diretoria de dados que criou acima:

```
# assumindo que a sua diretoria sample-data está em `~/data/seis-lab-data`
ln -s ~/data/seis-lab-data sample-data
```

Certifique-se de que tem o docker e o uv instalados na sua máquina.

Utilize o `uv` para instalar o projeto localmente:

```
uv sync --group dev --locked
```

Instale os hooks de pre-commit incluídos:

```
uv run pre-commit install
```

Efetue o *pull* das imagens docker do projeto dos respetivos registos (poderá precisar de fazer login em `ghcr.io`):

```
docker compose -f docker/compose.dev.yaml pull
```

De seguida, lance a stack:

```
docker compose -f docker/compose.dev.yaml up -d
```

Deverá agora conseguir aceder à webapp em

```
http://localhost:8888
```

Continue para a secção de arranque de uma instalação nova.

1.2 Arranque de uma instalação nova

O processo de arranque (*bootstrapping*) consiste em:

- Criar/atualizar a base de dados;
- Carregar as variáveis predefinidas nas tabelas apropriadas da BD;
- Opcionalmente, adicionar alguns projetos, missões de levantamento e registos de exemplo.

O *bootstrapping* é feito utilizando a CLI `seis-lab-data`, que está disponível no serviço `webapp`. Contém muitos comandos e pode ser invocada assim:

```
docker compose -f docker/compose.dev.yaml exec -ti webapp uv run seis-lab-data --help
```

Execute os seguintes comandos:

```
# inicializar a BD
docker compose -f docker/compose.dev.yaml exec -ti webapp uv run seis-lab-data db upgrade

# adicionar dados predefinidos
docker compose -f docker/compose.dev.yaml exec -ti webapp uv run seis-lab-data bootstrap all

# opcionalmente, carregar registos de exemplo
docker compose -f docker/compose.dev.yaml exec -ti webapp uv run seis-lab-data dev load-all-samples

# opcionalmente, gerar um grande número de registos sintéticos
# (mais útil quando se trabalha na interface web)
docker compose -f docker/compose.dev.yaml exec -ti webapp uv run seis-lab-data dev generate-many-projects --num-projects=50
```

1.3 Notas adicionais

A imagem docker de desenvolvimento usa a tag `latest` e é reconstruída em cada commit no ramo `main` do repositório. Assim sendo, deverá executar

```
docker compose -f docker/compose.dev.yaml pull webapp
docker compose -f docker/compose.dev.yaml up -d
```

sempre que souber que houve *merges* recentes.

Criar a imagem docker localmente

Na maior parte das vezes irá utilizar uma imagem docker pré-construída. No entanto, existe um caso especial em que será necessário criá-la localmente: quando adicionar uma nova dependência Python ao projeto. Nesse caso, crie a imagem com:

```
docker build \
  --tag ghcr.io/naturalgis/seis-lab-data/seis-lab-data:$(git branch --show-current) \
  --file docker/Dockerfile \
  .
```

Depois, reinicie a stack com:

```
CURRENT_GIT_BRANCH=$(git branch --show-current) docker compose -f docker/compose.dev.yaml up -d --force-recreate
```

Traduções no ambiente de desenvolvimento local

Como o ficheiro docker compose de desenvolvimento monta a diretoria `src` inteira via bind mount, os ficheiros `*.mo` compilados do contentor são mascarados pelos ficheiros presentes no disco local. Isto significa que após executar `seis-lab-data translations compile` é necessário reiniciar o serviço `webapp` para as alterações terem efeito.

1.4 Serviços auxiliares de desenvolvimento

A stack de desenvolvimento inclui alguns serviços adicionais relevantes:

DOZZLE

Instância dozzle, útil para monitorizar os *logs* dos vários serviços da stack. Acessível em http://localhost:8888/monitoring

JUPYTER

Instância jupyter, útil para escrever notebooks ou interagir com um REPL Python. Acessível em http://localhost:5002

PG-ADMIN

Instância pg-admin, útil para inspecionar as bases de dados da stack. Acessível em http://pgadmin.localhost:8888 com as credenciais:

- utilizador: `dev@dev.dev`
- password: `dev`

2. Execução de testes

Os testes normais correm dentro do contentor `webapp`, após instalar as dependências necessárias:

```
docker compose --file docker/compose.dev.yaml exec -ti webapp uv sync --locked --group gdal --group dev
docker compose --file docker/compose.dev.yaml exec -ti webapp uv run pytest
```

Os testes de integração correm com:

```
docker compose --file docker/compose.dev.yaml exec webapp uv run pytest -m integration
```

2.1 Testes end-to-end (E2E)

Os testes E2E correm fora da stack docker e requerem a instalação do playwright localmente:

```
uv run playwright install --with-deps chromium
```

Os testes podem então ser executados com:

```
uv run pytest \
    tests/e2e/ \
    -m e2e \
    --confcutdir tests/e2e \
    --user-email akadmin@email.com \
    --user-password admin123 \
    --base-url http://localhost:8888
```

A incantação anterior executa todos os testes E2E em modo *headless*. Para os executar em modo *headed*:

```
uv run pytest \
    tests/e2e/ \
    -m e2e \
    --confcutdir tests/e2e \
    --user-email akadmin@email.com \
    --user-password admin123 \
    --base-url http://localhost:8888 \
    --headed \
    --slowmo 1500
```