MADS - Deployment 2

It works on my machine

Raoul Grouls, 2 December 2024

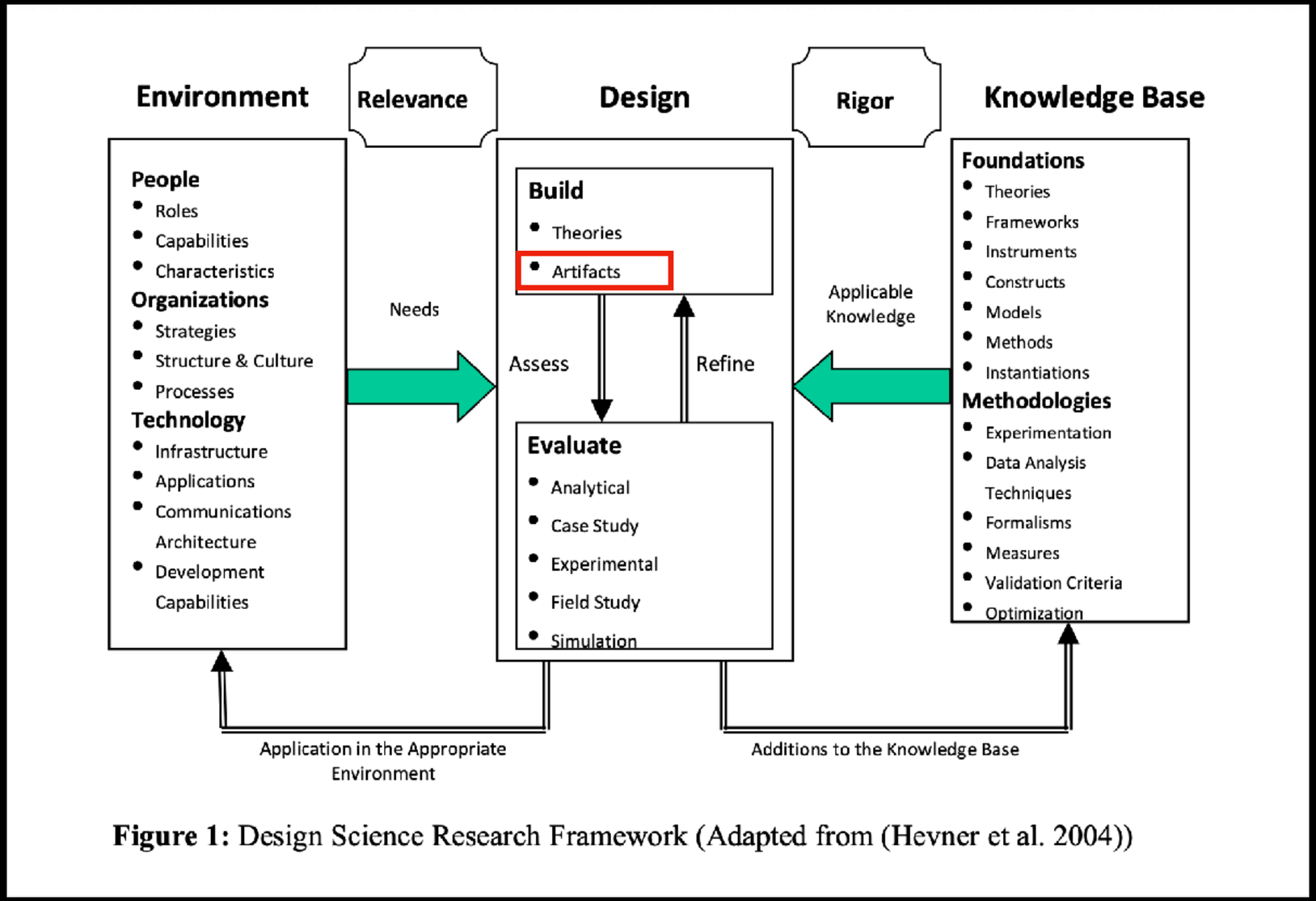


Figure 1: Design Science Research Framework (Adapted from (Hevner et al. 2004))

Vom Brocke, J., Hevner, A., & Maedche, A. (2020). Introduction to design science research. In *Design science research. Cases* (pp. 1-13). Cham: Springer International Publishing.

Ω and λ knowledge

- Descriptive (Ω) knowledge: informs the understanding of a problem, its context, and the underlying design of a solution entity.
- Prescriptive (λ) knowledge: solution design knowledge, in the form of growing design theorie, informs the design of a solution entity, a design process or a design system.

Vom Brocke, J., Hevner, A., & Maedche, A. (2020). Introduction to design science research. In *Design science research. Cases* (pp. 1-13). Cham: Springer International Publishing.

Artefacts we build

- Trained model weights (`.pth` files) and BPE-tokenizer files
- Docker images that serve the model
- Docker compose networks that run the pipeline / dashboard
- Wheel files that publish the code to create and serve the model
- Makefile automate the creation of all the artefacts

Docker Fundamentals: Key Concepts

Dockerfile -> builds -> **Image** -> runs -> **Container**

You should have produced something like this

```dockerfile
FROM python:3.12-slim
WORKDIR /app
RUN pip install --no-cache-dir requests loguru
COPY ingest/ingest.py .

CMD ["python", "ingest.py"]
```

```dockerfile
FROM python:3.11-slim
WORKDIR /app
RUN pip install --no-cache-dir pandas loguru pyarrow
COPY preprocess/preprocess.py .

CMD ["python", "preprocess.py"]
```

```dockerfile
FROM python:3.12-slim
COPY model/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

WORKDIR /app
COPY model/*.py .
EXPOSE 8000

ENTRYPOINT ["uvicorn", "serve:app", "--host", "0.0.0.0", "--port", "8000"]
```

This will work.

However…

```
build:
    docker build -t mads-ingest -f ingest/ingest.Dockerfile .
    docker build -t mads-preprocess -f preprocess/preprocess.Dockerfile .
    docker build -t mads-model -f model/serve.Dockerfile .

run:
    docker run \
        -v ./data:/app/data \
        -v ./logs:/app/logs mads-ingest
    docker run \
        -v ./data:/app/data \
        -v ./logs:/app/logs mads-preprocess
    docker run \
        -v ./data:/app/data \
        -v ./logs:/app/logs \
        -p 8000:8000 \
        mads-model

clean:
    rm -rf data/*
```

- But using docker-compose is better
- It makes it easier to combine multiple docker containers

```yaml
services:
  ingest:
    build:
      context: .
      dockerfile: ingest/ingest.Dockerfile
    volumes:
      - ./data:/app/data
      - ./logs:/app/logs

  preprocess:
    build:
      context: .
      dockerfile: preprocess/preprocess.Dockerfile
    volumes:
      - ./data:/app/data
      - ./logs:/app/logs
    depends_on:
      - ingest

  model:
    build:
      context: .
      dockerfile: model/serve.Dockerfile
    volumes:
      - ./data:/app/data
      - ./logs:/app/logs
    ports:
      - "8000:8000"
    depends_on:
      - preprocess

    healthcheck:
      test: ["CMD-SHELL", "curl -f http://localhost:8000/health && exit 0 || exit 1"]
      interval: 10s
      timeout: 10s
      retries: 3
      start_period: 10s
    restart: unless-stopped

```

Docker compose

Services Architecture:

- Three connected services: ingest, preprocess, and model
- Pipeline flow: ingest → preprocess → model

Common Patterns:

- Each service has dedicated Dockerfile
- Shared volumes for data and logs
- Volume mounts persist data to host machine

Key Features:

- Model service exposes port 8000
- Health checks on model service
- Automatic restart policy (restart: unless-stopped)
- Service dependencies ensure correct startup order

Volume Configuration:

- ./data:/app/data - shared data directory
- ./logs:/app/logs - centralized logging

Health Checks (model):

- Used in `docker ps` to report health
- 10s intervals with 3 retries
- 10s startup grace period
- Curl test to /health endpoint

Docker compose commands

`docker compose up`

- Starts all services defined in compose file
- Shows logs in terminal
- Creates networks/volumes if needed
- Builds images if not present

`docker compose up -d`

- Starts services in detached mode
- Runs in background
- No logs in terminal
- Returns control to shell

`docker compose down`

- Stops all running containers
- Removes containers and networks
- Preserves volumes by default
- Add `-v` flag to remove volumes

`docker ps`

- Shows running containers
- Displays: Container ID, Image, Command, Status, Ports
- Add `-a` flag (for “all”) to show stopped containers
- Useful for checking container health

uv-example

An example where

- `curl` is installed
- `uv` is installed to speed up installation of dependencies

```
18 # Base image
17 FROM python:3.12
16
15 # this updates dependencies, installs curl and
14 # rm -rf /var removes files only necessary during installation
13 RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*
12
11 WORKDIR /app
10
 9 ADD --chmod=755 https://astral.sh/uv/install.sh /install.sh
 8 RUN /install.sh && rm /install.sh
 7
 6 COPY ./requirements.txt .
 5 RUN /root/.local/bin/uv pip install --system --no-cache -r requirements.txt
 4
 3 COPY test.py test.py
 2 ENTRYPOINT ["python", "test.py"]
 1
```

Smaller torch builds

Dockerfile:

```
FROM raoulgrouls/torch-python-slim:py3.12-torch2.8.0-arm64-uv0.9.8
```

pyproject.toml:

```
[[tool.uv.index]]
name = "pytorch"
url = "https://download.pytorch.org/whl/cpu"
explicit = true
```

Makefile

- **Target**: File to be created or action to perform
- **Prerequisites**: Files or targets needed before recipes can run
- **Recipe**: Commands (indented with TAB) that make runs to create target or perform action
- `.PHONY` tells `make` these targets don't create actual files

```
target: prerequisites
    recipe
    recipe
    ...
```

Makefile

```
29 img/clustering.png: ../img/clustering.png
30     mkdir -p img
31     cp $< $@
32
33 build: img/clustering.png
34     @echo "$(YELLOW)Building Docker image ... $(NC)"
35     docker build -t $(IMAGE_NAME):$(IMAGE_TAG) .
36     @echo "$(GREEN)Build complete!$(NC)"
37
```

- Checks if target ($@) is missing or older than source ($<)
- Creates directory and copies only if needed
- Automatic variables:
  - $< (first prerequisite),
  - $@ (target name)

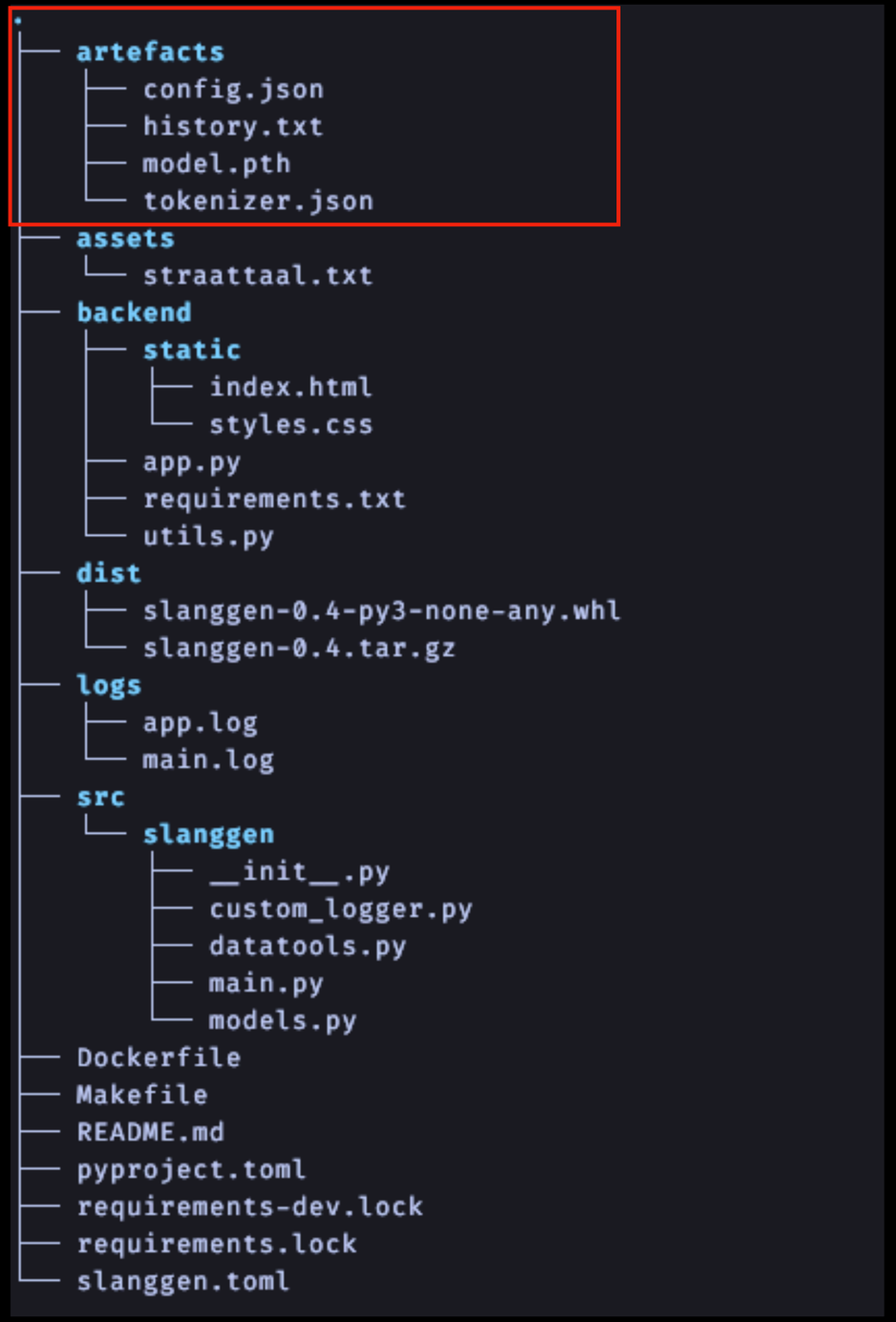
```
35 run: clean build
36     @echo "$(YELLOW)Starting container ... $(NC)"
37     docker run -d -p $(PORT):$(PORT) --name $(IMAGE_NAME) $(IMAGE_NAME):$(IMAGE_TAG)
38     @echo "$(GREEN)Service running on http://localhost:$(PORT)$(NC)"
39     @echo "$(GREEN)View image on on http://localhost:$(PORT)/show$(NC)"
40
```

Chain of dependencies:

- `Run` depends on `clean` and `build`
- `Build` depends on `img/clustering.png`
- Make resolves these automatically in correct order

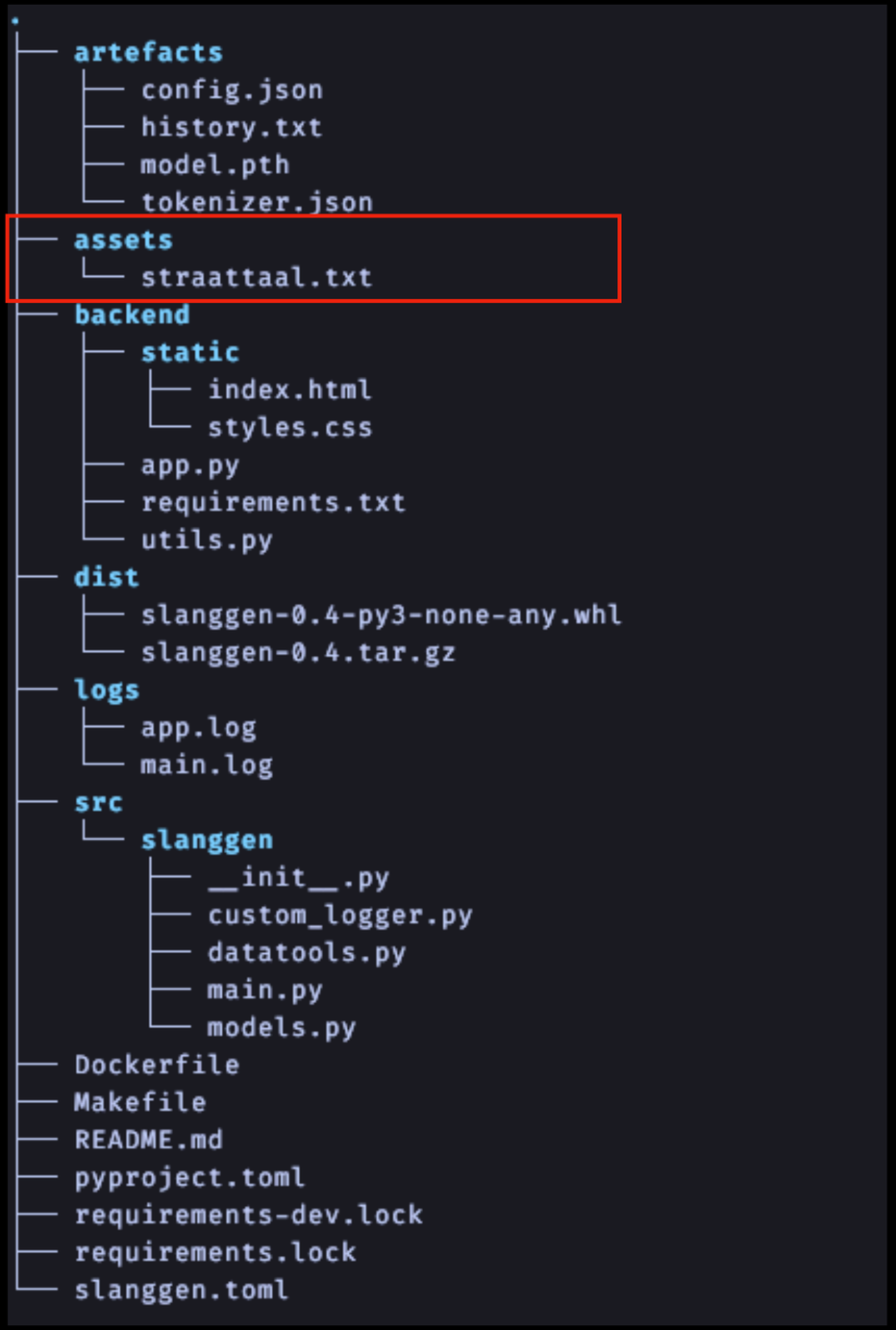
Straattaal

- artefacts: contains all artefacts we create



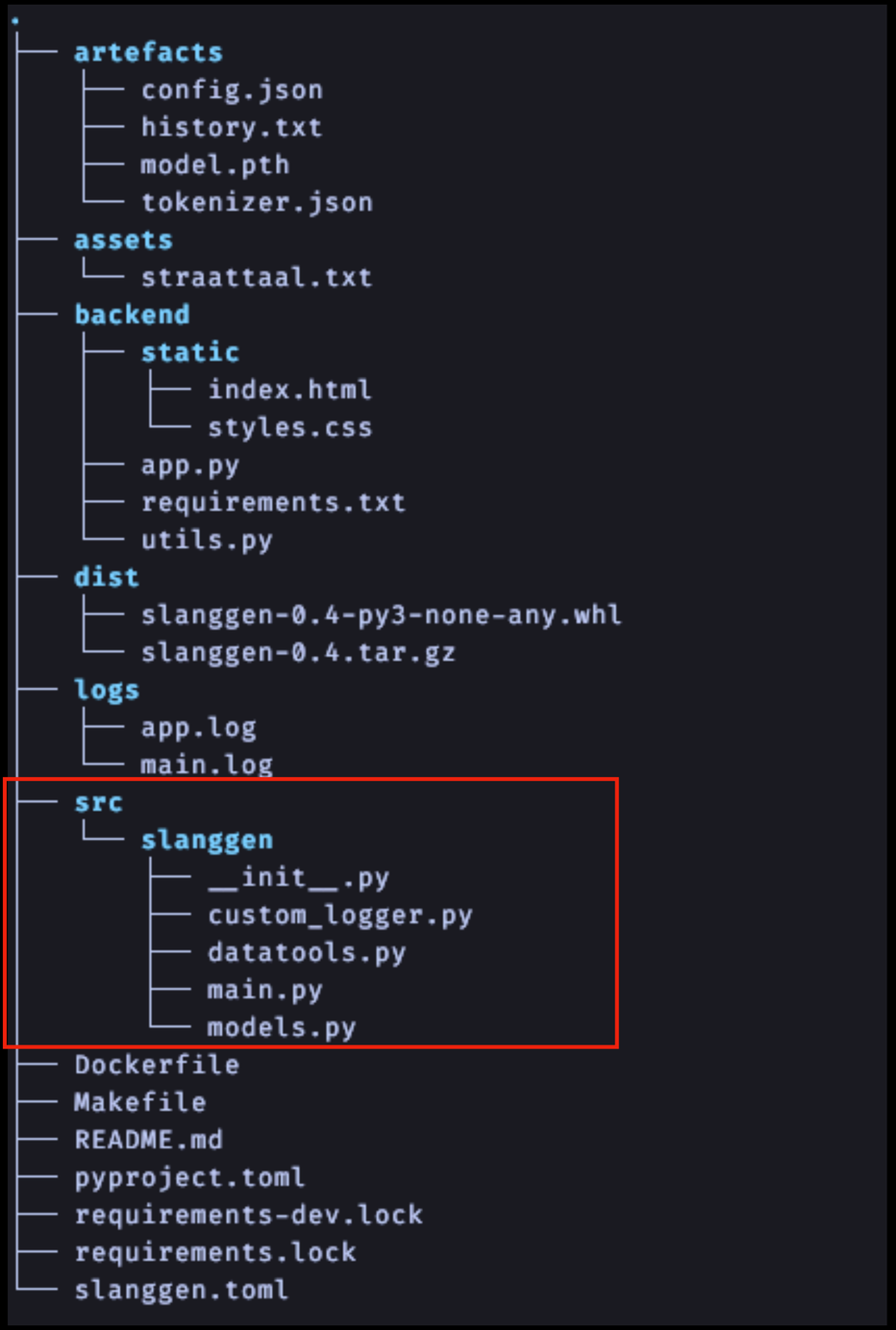
Straattaal

- assets: data we obtained



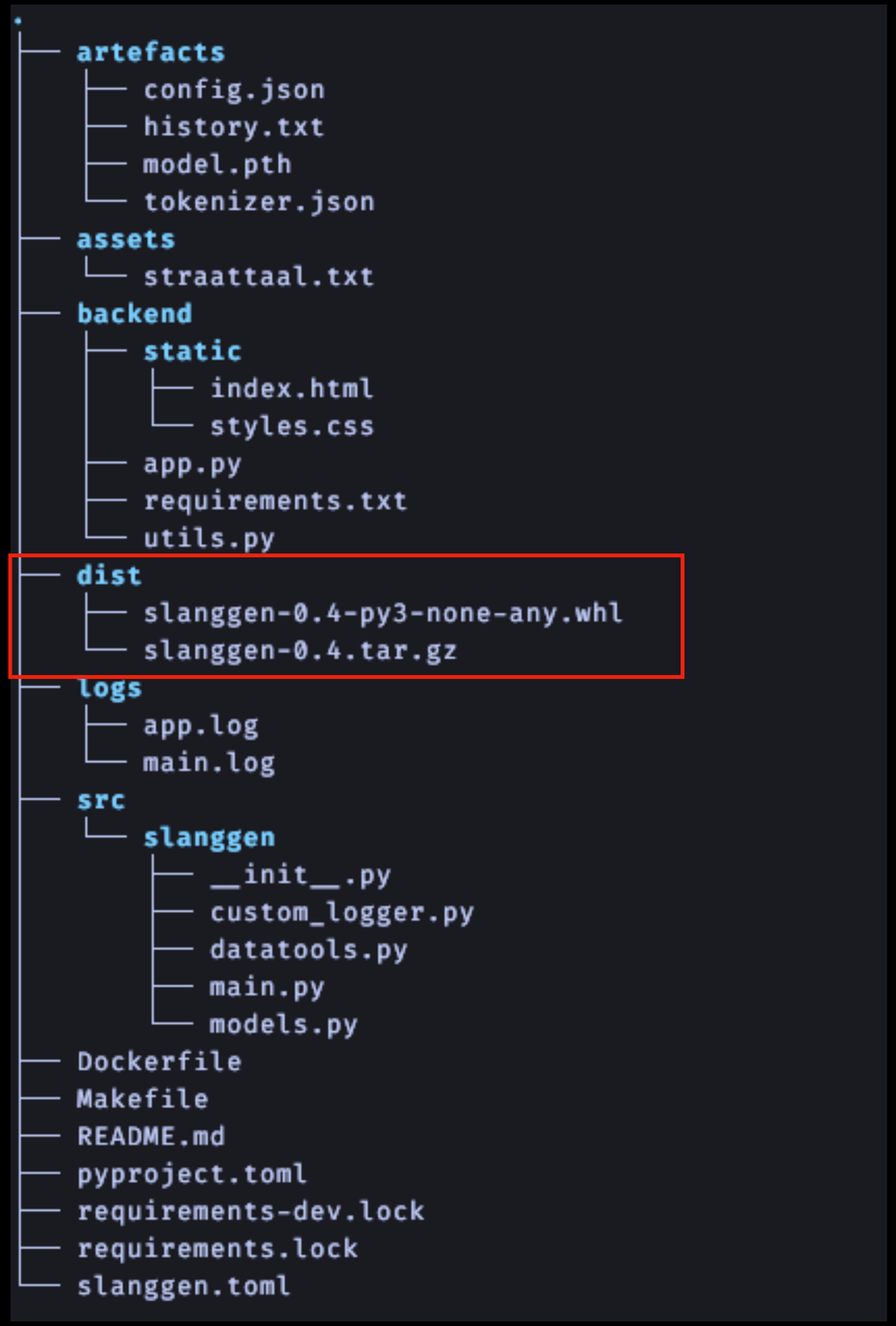
Straattaal

- src/slanggen: python src code for scraping, preprocessing and training a PyTorch model



Straattaal

- We can publish the src/slanggen code as a wheel file.
- We can install the wheel in the Dockerfile
- This makes it easier to import the model from `models.py` without worrying about dependencies



Straattaal

`dist/` is the distribution directory containing packaged versions

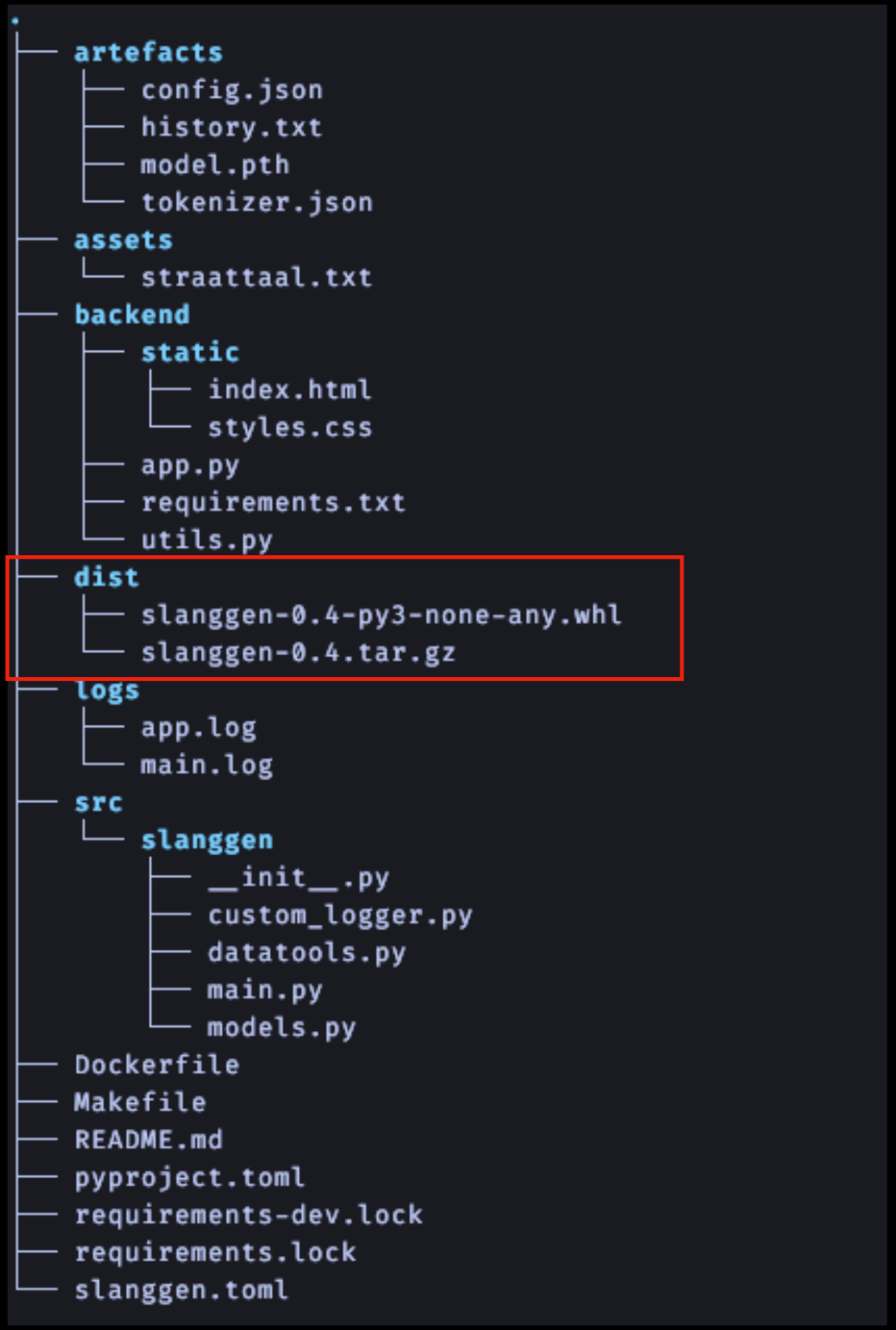
slanggen-0.4-py3-none-any.whl

- Wheel format (.whl) - faster to install than source distributions
- py3: Python 3 compatible
- any: Works on any platform

`slanggen-0.4.tar.gz`

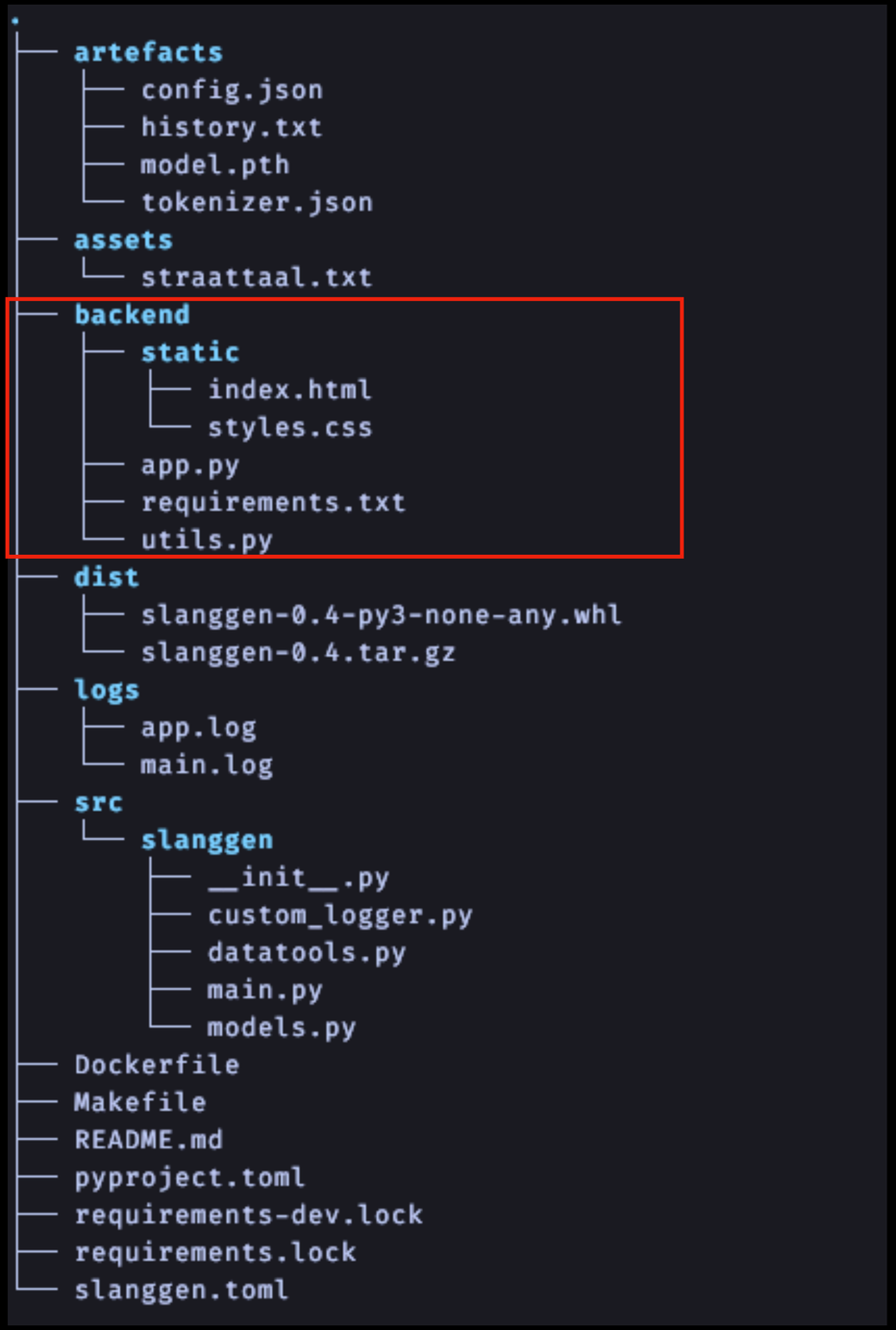
- Source distribution (sdist)
- Contains raw source code and build instructions
- Fallback if wheel installation fails
- Required for PyPI distribution

Both files represent version 0.4 of the slanggen package, just in different formats.



Straattaal

- backend: python code that uses the trained model for inference
- We can access slanggen (`from slanggen import models`) because we installed the package from the wheel
- Includes static folder with css/ html ; you could separate this as a stand alone docker container for the frontend



Installing dist

```
RUN --mount=source=dist,target=/dist PYTHONDONTWRITEBYTECODE=1 pip install --no-cache-dir /dist/*.whl
```

- RUN: Dockerfile instruction that executes commands during image build
- `--mount=source=dist,target=/dist`: Temporarily mounts local `dist` directory to `/dist` in container during build (why is that?)
- `PYTHONDONTWRITEBYTECODE=1`: Prevents Python from creating `.pyc` files
  - Container images should be immutable and `.pyc` files create varying builds
  - Adds unnecessary size to container images
  - Can cause permission issues in some container setups
- `pip install --no-cache-dir /dist/*.whl`: Installs all wheel files from mounted /dist directory without caching