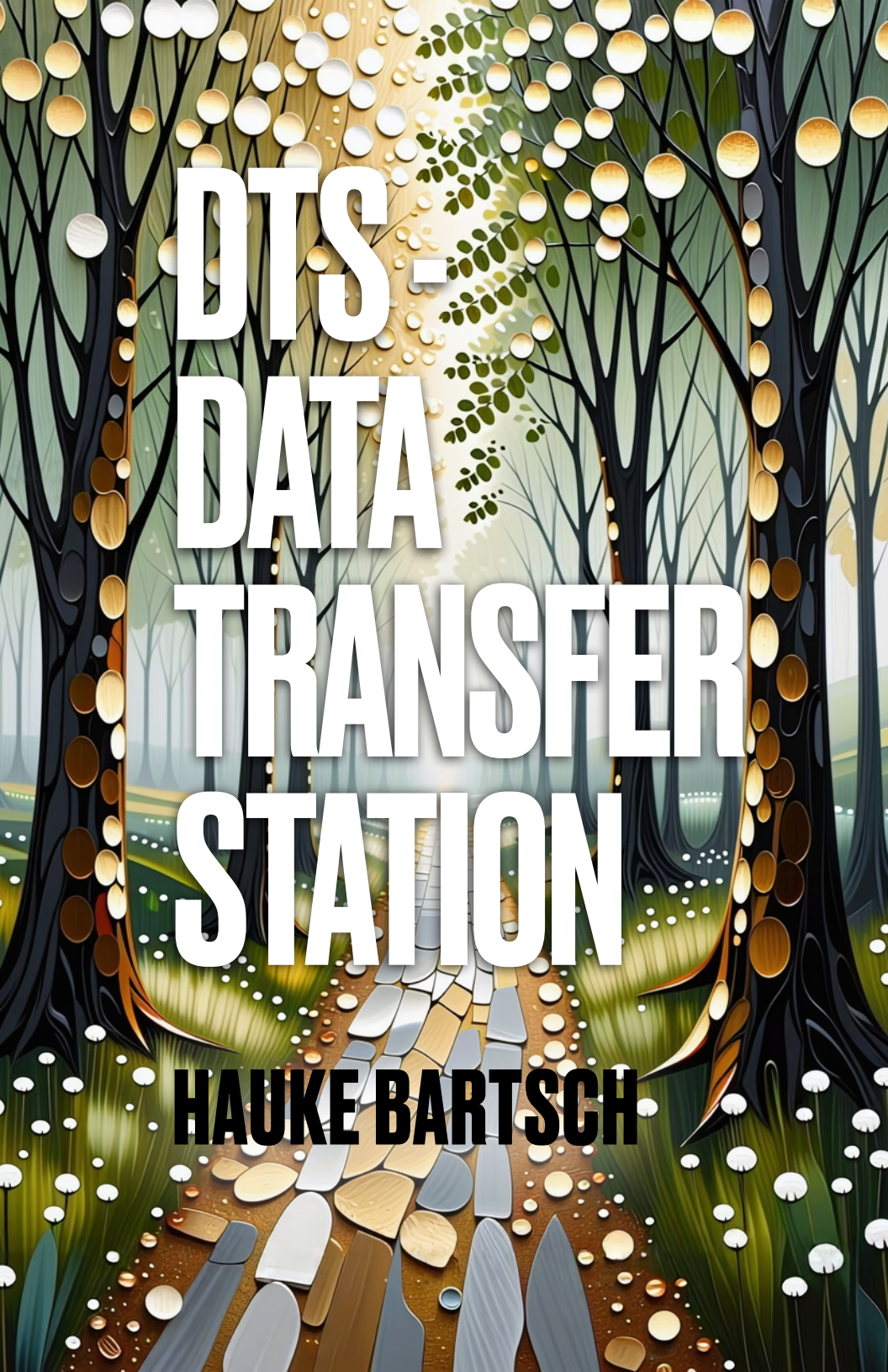

DTS -
DATA
TRANSFER
STATION
HAUKE BARTSCH

DTS - DATA TRANSFER STATION

SYSTEM SPECIFICATION

VERSION 0.2
06/2025

A self-hosted digital platform for the secure image exchange and processing integrated with hospital systems using industry standards.

Hauke Bartsch
Møllendalsbakken 1
5009 Bergen, Norway
github.com/HaukeBartsch/data-transfer-station

System overview

System hardware standard
Main memory: 12GB
Disk space: 512GB
CPU cores: 6
OS: Linux, Ubuntu LTS 24.04
Connectivity: ports 11112 (DICOM), 22 (ssh), 80/443 (configuration)
Software dependencies: DCMTK, python3, docker, apache2, php
System hardware by core
Main memory: based on AI core up to +20GB
Disk space: based on AI core container space requirement (up to 20Gb)
CPU cores: no additional CPU cores needed
GPU: none (only CPU prediction is supported)

INTRODUCTION

DTS is a DICOM-aware platform that allows for in-house image processing using A.I. cores. An A.I. core can be a python-based convolutional neural network or an advanced algorithm. Image data can be forwarded from PACS (Picture Archive and Communication System) and trigger's processing pipelines. After processing a routing component can direct incoming and outgoing images on the network.
DTS participates in a hospital network as a fully automated and configurable medical report service. Radiologists will be sending examinations, processing is triggered and resulting reports bounces back to the sending system.

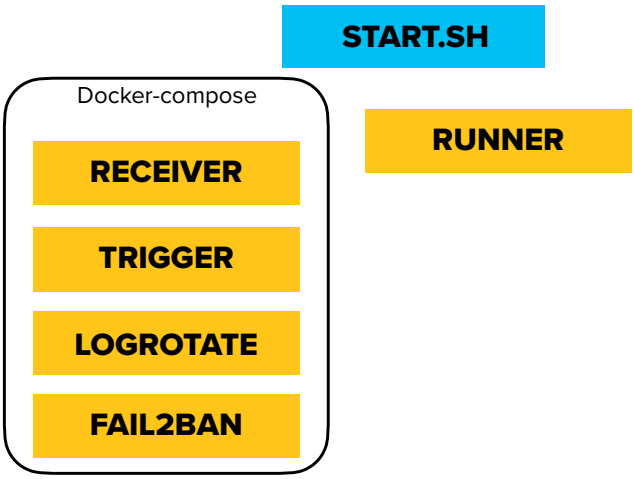
Features:

- DICOM receive using DCMTKs storescp
- Hosting of several A.I. cores (dev/prod)
- Auto-forwarding based on DICOM tags

DTS Components

Receiver	Listen to incoming DICOM requests
Trigger	Detects processing jobs and sets up data
Runner	Starts processing and forwards generated data
Viewer	Log viewer with configuration/setup screen (opt)
A.I. core	Containerized advanced processing core

Architecture



RECEIVER

The DICOM receiver service runs as a docker container (under docker-compose). The configuration for data (/data) and the docker internal port (11112) are contained in a configuration file inside the container.

```
{
  "DICOMIP": "localhost",
  "DICOMPORT": "11112",
  "DICOMAETITLE": "FIONA",
  "DATADIR": "/data",
  "LOCALTIMEZONE": "Europe/Copenhagen",
  "SITES": {
    "PROJ": {
      "DICOMIP": "localhost",
      "DICOMPORT": "11112",
      "DATADIR": "/data"
```

```
        }
      }
    }
```

Docker compose can build and start “receiver:latest”. Data is stored outside of the container in three folders “/data/site/archive/“ (DICOM by study instance), “/data/site/raw/“ (links to DICOM by participant and date), and “/data/site/.arrived/“ (job trigger files).

Test this container by a) check if storescpFIONA and processSingleFile3.py are running inside (started by cron). You can connect using ‘docker exec -it receiver bash’ and use ‘ps aux’.

You can also test the container by sending DICOM images to port 11112. They should appear in the /data/site/archive folder visible on the host.

TRIGGER

The trigger container is configured in a /root/data-transfer-station/configuration/ folder. A job created by trigger will appear in /data/code/workflow_joblist.jobs as a json-encoded line.

```
{
  "arrived": "/data/site/.arrived",
  "raw": "/data/site/raw",
  "archive": "/data/site/archive",
  "timeout": 16,
  "log": "/data/logs/trigger.log",
  "logging": [
        { "host": "localhost", "port": 3306,
          "dbname": "", "driver": ""
        }
  ],
  "Streams": [
      {
        "log": "/data/logs/trigger.log",
        "name": "AIcore",
        "description": "AIcore",
        "trigger": {
             "AETitleCalled": "^AICORE1$",
             "AETitleCaller": ".*"
         },
         "destination": [
            {
              "AETitle": "FIONA",
              "IP": "fiona.ihelse.net",
              "PORT": "7280",
              "filter": [
                  { "SeriesDescription": "*" }
              ]
            }
```

```
            ],
            "trigger-study": [
                 {
                    "type": "exec",
                    "cmd": [ "echo", “triggered",
                       “@StudyInstanceUID@",
                       "@PATH@",
                       “@DESCRIPTION@“,
                       “@StreamName@“]
                    }
            ],
            "trigger-series": []
        }
      ]
    }
```

Logging supports on-system logs in /data/logs/ as well as off-site SQL logging configured in “.logging” based on the python library sqlalchemy.
Each active A.I. core is expected to match to an entry in “.Streams” where trigger is configured based on incoming application entity (AE) title. This option allows several A.I. cores using the same IP/port configuration (dev/prod).
The A.I. core functionality is in either ‘trigger-study’ (default) or ‘trigger-series’ based on when a A.I. core should be run. The order of operations for an A.I. core consists of starting the containerized prediction and generating reports.
All images in the output folder will be forwarded based on the configuration in “.destination”. Note that additional filtering is possible in this step.

RUNNER

The runner component starts jobs created by the trigger service (workflow_joblist.jobs). Because it needs to startup docker containers (A.I. cores) it needs to run on the host. By default the start.sh script will create a cron job that starts the runner every minute to check for work.

Setup

Start the service with ‘./start.sh’ a script located in the root directory of the data-transfer-station github code. In case of issues you can force the docker containers to be build again locally using ‘docker compose build --no-cache’.

Verify that services are running correctly by inspecting the log files created in /data/logs/.