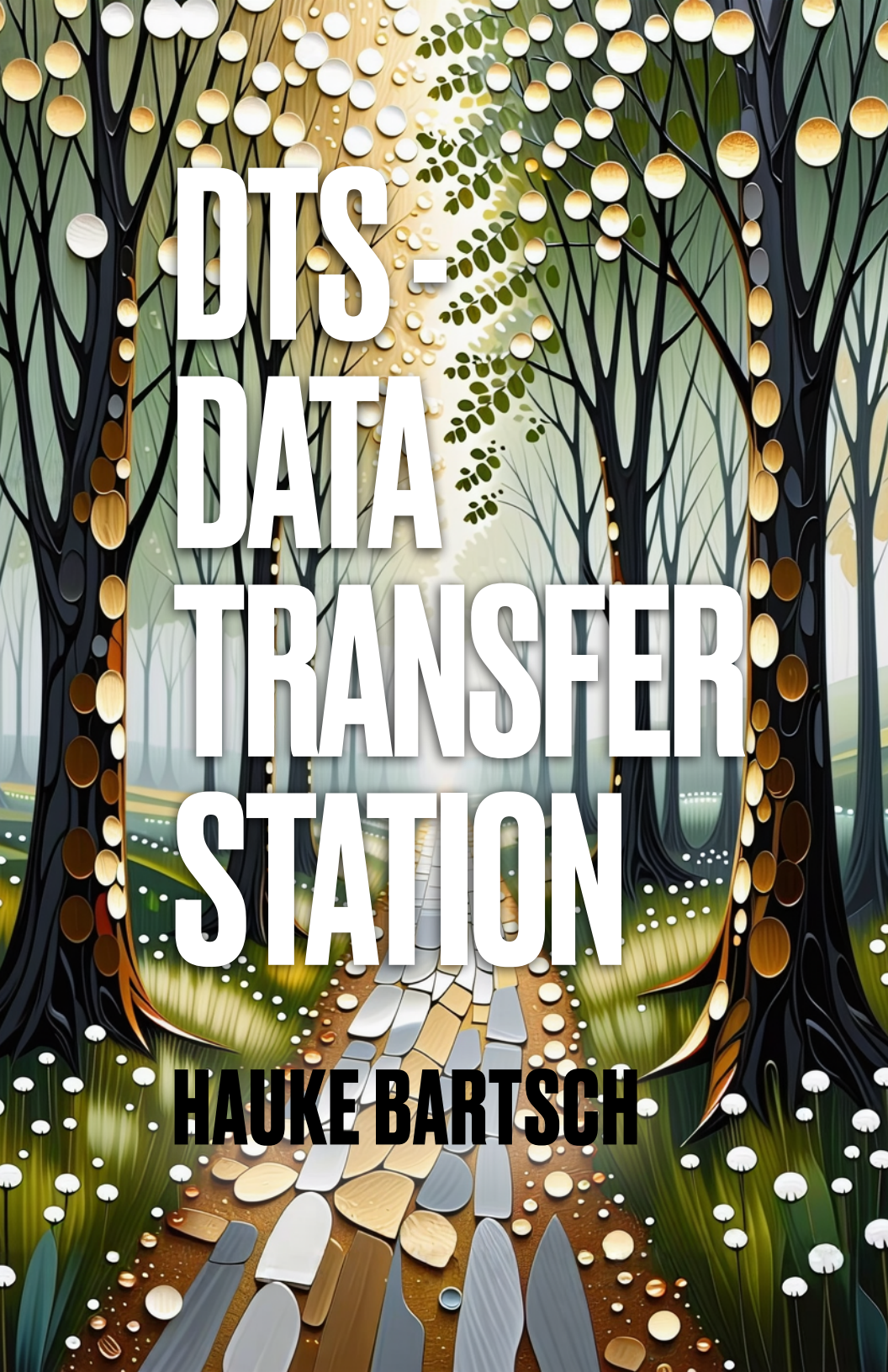
DTS -
DATA
TRANSFER
STATION
HAUKE BARTSCH

DTS - DATA TRANSFER STATION

SYSTEM SPECIFICATION

VERSION 0.1
03/2025

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, podman, 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. Image data can be forwarded from PACS and trigger 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 starts A.I. cores
Viewer	Log viewer with configuration/setup screen
A.I. core	Containerized advanced processing core

Configuration

RECEIVER

The DICOM receiver service is configured as a systemd service in /etc/systemd/system/docker.receiver.service. The port for the service as well as the location of the incoming data is configured inside the systemd file. Here is an example for such a configuration where image data is accepted on port 11112 and data is stored in /data.

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

The service will start a containerized image “receiver:latest” that needs to be present on the hosting computer. Data is stored outside of the container in three folders “${DATADIR}/site/archive/“ (DICOM by study instance), “${DATADIR}/site/raw/“ (links to DICOM by

participant and date), and “${DATADIR}/site/.arrived/“ (job trigger files).

TRIGGER

The trigger service is configured as a systemd service in /etc/systemd/system/trigger.service. The service is implemented as a python (>3.0) script 'trigger.py' located in /data/code/trigger/. All configuration options are stored in a JSON formatted file in the same directory.

```
{
  "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@" ]
                }
            ],
            "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.