# DCONAP End-to-End Docker Platform Documentation

This comprehensive documentation details the entire DCONAP (Development & Continuous Operations for AI/ML Platform) Docker-based setup from scratch: all files, services, rationale, configurations, commands, and best practices.

## 1. Prerequisites

1. **Host Requirements**:
   * Docker Engine (v20.10+)
   * Docker Compose plugin (v2+ recommended)
   * Linux/macOS/Windows (Docker Desktop)
   * Minimum 16 GB RAM, 4‑core CPU
2. **Repositories & Files** (root of repo):

├── Dockerfile

├── docker-compose.yml

├── .env

├── entrypoint.sh

├── requirements.txt

├── baseline/ # Your Jupyter notebooks & code

└── docker/ # Additional configs (dags/, grafana\_provisioning/, prometheus.yml)

## 2. File-by-File Breakdown

### 2.1 .env

Centralizes all environment and build arguments.

# Network

DOCKER\_NETWORK=dconap-network

# Postgres

POSTGRES\_USER=dconap\_user

POSTGRES\_PASSWORD=dconap\_pass

POSTGRES\_DB=dconap\_db

POSTGRES\_PORT=5432

POSTGRES\_VOLUME=postgres\_data

# Redis

REDIS\_PORT=6379

REDIS\_VOLUME=redis\_data

# GitLab CE

GITLAB\_HOSTNAME=gitlab.example.com

GITLAB\_PORT\_HTTP=80

GITLAB\_PORT\_HTTPS=443

GITLAB\_PORT\_SSH=22

GITLAB\_CONFIG\_VOLUME=gitlab\_config

GITLAB\_LOGS\_VOLUME=gitlab\_logs

GITLAB\_DATA\_VOLUME=gitlab\_data

# Spark

SPARK\_MASTER\_PORT=7077

SPARK\_MASTER\_UI\_PORT=8088

SPARK\_WAREHOUSE\_VOLUME=spark\_warehouse

SPARK\_LOGS\_VOLUME=spark\_logs

SPARK\_WORKER\_MEMORY=1G

SPARK\_WORKER\_CORES=2

# Airflow

AIRFLOW\_PORT=8081

AIRFLOW\_\_CORE\_\_EXECUTOR=CeleryExecutor

AIRFLOW\_\_CORE\_\_SQL\_ALCHEMY\_CONN=postgresql+psycopg2://dconap\_user:dconap\_pass@postgres:5432/dconap\_db

AIRFLOW\_\_CELERY\_\_BROKER\_URL=redis://redis:6379/0

AIRFLOW\_\_CELERY\_\_RESULT\_BACKEND=db+postgresql://dconap\_user:dconap\_pass@postgres:5432/dconap\_db

AIRFLOW\_\_CORE\_\_FERNET\_KEY=your\_fernet\_key\_here

# Kafka & Zookeeper

KAFKA\_BROKER\_ID=1

KAFKA\_PORT=9092

ZOOKEEPER\_PORT=2181

# Flink

FLINK\_JOBMANAGER\_PORT=6123

FLINK\_JOBMANAGER\_UI\_PORT=8082

FLINK\_TASK\_SLOTS=2

FLINK\_TASKMANAGER\_SCALE=2

# Superset

SUPERSET\_PORT=8085

SUPERSET\_ADMIN\_USER=admin\_user

SUPERSET\_ADMIN\_PASSWORD=admin\_password

SUPERSET\_ADMIN\_FIRSTNAME=Superset

SUPERSET\_ADMIN\_LASTNAME=Admin

SUPERSET\_SECRET\_KEY=some\_secret\_key

SUPERSET\_VOLUME=superset\_data

# Prometheus

PROMETHEUS\_PORT=9090

PROMETHEUS\_VOLUME=prometheus\_data

# JupyterLab / Dev Container

USER\_ID=1000

GROUP\_ID=1000

VENV\_PATH=/opt/venv

JUPYTER\_WORKSPACE\_DIR=/home/jupyteruser/work

JUPYTER\_PORT=8888

JUPYTER\_TOKEN=dconap-token

# JUPYTER\_PASSWORD=admin # Use only one of token/password

LOG\_DIR=${JUPYTER\_WORKSPACE\_DIR}/logs

**Why parameterize?**

* Decouple code from environment
* Easy overrides per developer or CI/CD pipeline
* Single source of truth for all ports, volumes, credentials

### 2.2 Dockerfile (Multi-Stage)

Optimized build producing a slim runtime image.

# Stage 1: builder

ARG PYTHON\_IMAGE=python:3.9-slim

FROM ${PYTHON\_IMAGE} AS builder

...

RUN addgroup --gid ${GROUP\_ID} jupytergroup && useradd -ms /bin/bash -u ${USER\_ID} -g jupytergroup jupyteruser

WORKDIR ${VENV\_PATH}

RUN python3 -m venv . && . bin/activate && pip install --upgrade pip

COPY requirements.txt /tmp/

RUN . ${VENV\_PATH}/bin/activate && pip install --no-cache-dir -r /tmp/requirements.txt

# Stage 2: runtime

FROM ${PYTHON\_IMAGE} AS runtime

RUN apt-get update && apt-get install -y tini jq && rm -rf ...

COPY --from=builder ${VENV\_PATH} ${VENV\_PATH}

RUN addgroup ... usermod -aG sudo jupyteruser && ...

USER jupyteruser

WORKDIR ${JUPYTER\_WORKSPACE\_DIR}

ENV PATH="${VENV\_PATH}/bin:$PATH" ...

COPY entrypoint.sh /entrypoint.sh

COPY baseline ${JUPYTER\_WORKSPACE\_DIR}

EXPOSE ${JUPYTER\_PORT}

ENTRYPOINT ["tini","--","/entrypoint.sh"]

CMD jupyter-lab --ip=0.0.0.0 --port=${JUPYTER\_PORT} --no-browser

**Why multi-stage?**

* Builder contains heavy build tools & caches
* Runtime only has minimal dependencies & virtualenv
* Final image size reduced by ~300 MB

### 2.3 docker-compose.yml

Orchestrates all services with named networks, volumes, and logging.

version: '3.9'

networks:

dconap\_network:

name: ${DOCKER\_NETWORK}

volumes:

postgres\_data: {name: ${POSTGRES\_VOLUME}}

...

services:

postgres:

image: postgres:latest

environment: {POSTGRES\_USER: ${POSTGRES\_USER}, ...}

volumes: [postgres\_data:/var/lib/postgresql/data]

ports: ["${POSTGRES\_PORT}:5432"]

logging: {driver: json-file, options: {max-size: "10m", max-file: "3"}}

profiles: [base,dev]

spark-master:

image: bitnami/spark:3.5.0

ports: ["${SPARK\_MASTER\_UI\_PORT}:8080","${SPARK\_MASTER\_PORT}:7077"]

healthcheck: {...}

profiles: [spark]

development:

build:

context: .

args: {USER\_ID: ${USER\_ID}, ...}

volumes: ["./baseline:${JUPYTER\_WORKSPACE\_DIR}"]

ports: ["${JUPYTER\_PORT}:${JUPYTER\_PORT}"]

profiles: [dev]

logging: {...}

**Why Compose?**

* Single CLI to manage 12+ containers
* Profiles let you spin up subsets (dev, spark, monitoring, full)
* Named volumes persist data across container restarts
* Built-in healthchecks and logging policies

### 2.4 entrypoint.sh

Standardized startup for Jupyter with logging & auth.

1. **Init dirs**: ${LOG\_DIR}, ${JUPYTER\_CONFIG\_DIR}
2. **Ownership**: chown -R ${UID}:${GID}
3. **Logging**: tee -a {timestamped logfile}
4. **Auth**:
   * If JUPYTER\_PASSWORD, generate hash via Python and write jupyter\_server\_config.py
   * Else if JUPYTER\_TOKEN, write token config
   * Else rely on default Jupyter token
5. **Exec**: Launch $\* (shell-form CMD ensures ${JUPYTER\_PORT} expansion)

## 3. Services and Why They’re Used

| Service | Role | Why Chosen / Config Highlights |
| --- | --- | --- |
| PostgreSQL | Relational metadata & Airflow metadata database | Persistent volume, default Postgres image, port from ${ENV} |
| Redis | Celery broker for Airflow | Light-weight, in-memory, persisted via volume |
| GitLab CE | On-premise Git + CI/CD | Omnibus image, ports 80/443/22, external URL config |
| Spark | Batch & interactive analytics | Bitnami image, master & workers, healthchecks, warehouse dir |
| Airflow | DAG orchestration & scheduling | CeleryExecutor, Postgres+Redis backend, Fernet key |
| Kafka & ZK | Event streaming backbone | Confluent images, single broker, ZK coordination |
| Flink | Stateful stream processing | Apache Flink image, jobmanager & taskmanager split |
| Superset | BI & dashboarding | Apache Superset, admin creds, secret key, port from ${ENV} |
| Prometheus | Metrics collection | Prometheus v2, custom prometheus.yml, volume for DB |
| Grafana | Metrics visualization | Grafana OSS, provisioning folder, admin creds |
| Development | Interactive Python & ML dev (JupyterLab) | Custom multi-stage image, token/password, logs, notebooks |

## 4. Step-by-Step Setup

1. **Clone repo**:

git clone <repo-url> && cd DCONAP

1. **Populate** .env: Edit values or keep defaults.
2. **Build & start dev stack**:

docker compose --env-file .env up --build -d development postgres redis

1. **Access Jupyter**: http://localhost:${JUPYTER\_PORT} (token from logs or ${JUPYTER\_TOKEN})
2. **Start Spark**:

docker compose --env-file .env up -d spark-master spark-worker

Access Spark UI: http://localhost:${SPARK\_MASTER\_UI\_PORT}

1. **Start monitoring**:

docker compose --env-file .env up -d prometheus grafana

* + Prometheus UI: http://localhost:${PROMETHEUS\_PORT}
  + Grafana UI: http://localhost:${GRAFANA\_PORT} (default admin/admin)

1. **Optional: Airflow & Superset**:

docker compose up -d airflow superset

* + Airflow UI: http://localhost:${AIRFLOW\_PORT}
  + Superset UI: http://localhost:${SUPERSET\_PORT}

## 5. Best Practices & Troubleshooting

* **Volume Cleanup**: docker volume prune to remove unused volumes.
* **Logs**: Check logs via docker logs <container> or in workspace logs/ directory.
* **Scaling Spark**: docker compose up --scale spark-worker=3
* **Resource Limits**: Add deploy.resources.limits.memory in Compose for production.
* **Upgrading Images**: Bump versions in .env, then docker compose pull && up -d --build.
* **Profiles** (v2+): --profile dev, --profile spark, --profile monitoring, --profile full.

Document last updated: April 26, 2025