MarkLogic SwarmCare Setup (Docker + Forests + ETL)

This document contains a turnkey implementation prompt and files to set up MarkLogic running in Docker, create forests and databases (schemas/ontology, knowledge graph, staging, final, modules), install server-side transforms, and provide example ETL scripts to ingest and promote SwarmCare EHR/Patient/Doctor data. Drop these files into a project and provide them to your code assistant to implement.

docker-compose.yml

```
version: "3.8"

services:
  marklogic:
    image: store/marklogicdb/marklogic-server:latest
    container_name: marklogic-swarmcare
    restart: unless-stopped
    ports:
      - "8000:8000"   # App-Services
      - "8001:8001"   # Admin UI
      - "8002:8002"   # Management API
      - "8040:8040"   # SwarmCare STAGING REST
      - "8041:8041"   # SwarmCare FINAL REST
    environment:
      MARKLOGIC_ADMIN_USERNAME: admin
      MARKLOGIC_ADMIN_PASSWORD: admin123
    volumes:
      - ./modules:/opt/marklogic/modules
      - ./data:/opt/marklogic/data
      - ./scripts:/opt/scripts
```

scripts/bootstrap-marklogic.sh

```
#!/usr/bin/env bash
set -euo pipefail

ML_HOST="localhost"
ADMIN_USER="${MARKLOGIC_ADMIN_USERNAME:-admin}"
ADMIN_PASS="${MARKLOGIC_ADMIN_PASSWORD:-admin123}"
MGMT="http://${ML_HOST}:8002/manage/v2"
REST="http://${ML_HOST}:8000"
STAGING_PORT=8040
FINAL_PORT=8041

# ---------- helpers ----------
auth=(-u "${ADMIN_USER}:${ADMIN_PASS}" --anyauth -sS)
jhdr=(-H "Content-Type: application/json")

get_json() { curl "${auth[@]}" -X GET "$1?format=json" || true; }
exists_endpoint() { curl "${auth[@]}" -X GET "$1" -o /dev/null -w "%{http_code}" -s; }

wait_for_ml() {
  echo "■ Waiting for MarkLogic Management API..."
  for i in {1..120}; do
    if curl -s "http://${ML_HOST}:8002/version" >/dev/null; then
      echo "■ MarkLogic is up"
      return 0
    fi
    sleep 2
  done
  echo "■ Timed out waiting for MarkLogic"
  exit 1
}

create_db() {
```

```
  local db="$1"
  if [ "$(exists_endpoint "${MGMT}/databases/${db}")" = "200" ]; then
    echo "■■ Database ${db} already exists"
    return 0
  fi
  echo "■ Creating database: ${db}"
  curl "${auth[@]}" "${jhdr[@]}" -X POST "${MGMT}/databases"     -d "{"database-name":"${db}"}"
}

create_forest() {
  local forest="$1"; local host="localhost"
  if [ "$(exists_endpoint "${MGMT}/forests/${forest}")" = "200" ]; then
    echo "■■ Forest ${forest} already exists"
    return 0
  fi
  echo "■ Creating forest: ${forest}"
  curl "${auth[@]}" "${jhdr[@]}" -X POST "${MGMT}/forests"     -d "{"forest-name":"${forest}","host":"${host}"}"
}

attach_forest() {
  local db="$1"; local forest="$2"
  # check attachment
  if get_json "${MGMT}/databases/${db}/forests" | grep -q ""${forest}""; then
    echo "■ Forest ${forest} already attached to ${db}"
    return 0
  fi
  echo "■ Attaching forest ${forest} -> ${db}"
  curl "${auth[@]}" "${jhdr[@]}" -X POST "${MGMT}/databases/${db}/forests"     -d "{"forest-name":"${forest}"}"
}

put_db_props() {
  local db="$1"; shift
  local body="$1"
  echo "■  Updating properties for ${db}"
  curl "${auth[@]}" "${jhdr[@]}" -X PUT "${MGMT}/databases/${db}/properties" -d "${body}"
}

create_rest_api() {
  local name="$1"; local port="$2"; local db="$3"; local modulesDB="$4"
  # REST API existence check
  if [ "$(exists_endpoint "${REST}/v1/rest-apis/${name}")" = "200" ]; then
    echo "■■ REST API ${name} already exists"
    return 0
  fi
  echo "■ Creating REST API: ${name} on ${port} -> ${db}"
  curl "${auth[@]}" "${jhdr[@]}" -X POST "${REST}/v1/rest-apis"     -d "{
      "rest-api":{
        "name":"${name}",
        "port":${port},
        "database":"${db}",
        "modules-database":"${modulesDB}"
      }
    }"
}

install_transform() {
  local server_port="$1"; local name="$2"; local file="$3"
  echo "■ Installing transform ${name} on port ${server_port}"
  curl "${auth[@]}" -X PUT     -H "Content-Type: application/javascript"     --data-binary @"${file}"     "http:
}

# ---------- START ----------
wait_for_ml

# Databases
SCHEMAS_DB="Schemas-DB"
KG_DB="KnowledgeGraph-DB"
MODULES_DB="Swarmcare-Modules"

STAGE_PAT_DB="Staging-Patient-DB"
STAGE_DOC_DB="Staging-Doctor-DB"
STAGE_EHR_DB="Staging-EHR-DB"
```

```
FINAL_PAT_DB="Final-Patient-DB"
FINAL_DOC_DB="Final-Doctor-DB"
FINAL_EHR_DB="Final-EHR-DB"

for db in "$SCHEMAS_DB" "$KG_DB" "$MODULES_DB"           "$STAGE_PAT_DB" "$STAGE_DOC_DB" "$STAGE_EHR_DB"
  create_db "$db"
done

# Forests (one per DB)
for db in "$SCHEMAS_DB" "$KG_DB" "$MODULES_DB"           "$STAGE_PAT_DB" "$STAGE_DOC_DB" "$STAGE_EHR_DB"
  f="${db}-Forest"
  create_forest "$f"
  attach_forest "$db" "$f"
done

# Set DB properties
# Schemas DB (kept simple)
put_db_props "$SCHEMAS_DB" '{
  "maintain-last-modified": true
}'

# Knowledge Graph DB (triples on)
put_db_props "$KG_DB" '{
  "triple-index": true,
  "collection-lexicon": true
}'

# Content DB template builder
content_props() {
cat <<JSON
{
  "schema-database": "${SCHEMAS_DB}",
  "triple-index": true,
  "collection-lexicon": true,
  "range-element-index": [
    {
      "scalar-type": "string",
      "localname": "patientId",
      "namespace-uri": "",
      "collation": "http://marklogic.com/collation/"
    },
    {
      "scalar-type": "string",
      "localname": "doctorId",
      "namespace-uri": "",
      "collation": "http://marklogic.com/collation/"
    }
  ]
}
JSON
}

for db in "$STAGE_PAT_DB" "$STAGE_DOC_DB" "$STAGE_EHR_DB"           "$FINAL_PAT_DB" "$FINAL_DOC_DB" "$FINAL_EHR_
  put_db_props "$db" "$(content_props)"
done

# Modules DB
put_db_props "$MODULES_DB" '{
  "collection-lexicon": true
}'

# REST APIs
create_rest_api "swarmcare-staging" "${STAGING_PORT}" "$STAGE_EHR_DB" "$MODULES_DB"
create_rest_api "swarmcare-final"   "${FINAL_PORT}"   "$FINAL_EHR_DB"  "$MODULES_DB"

# Install transforms on both endpoints (so they're available regardless of server used)
install_transform "${STAGING_PORT}" "staging-metadata" "/opt/marklogic/modules/transforms/staging-metadata.sjs"
install_transform "${FINAL_PORT}"   "staging-metadata" "/opt/marklogic/modules/transforms/staging-metadata.sjs"
install_transform "${STAGING_PORT}" "final-metadata"   "/opt/marklogic/modules/transforms/final-metadata.sjs"
install_transform "${FINAL_PORT}"   "final-metadata"   "/opt/marklogic/modules/transforms/final-metadata.sjs"

echo "■ Bootstrap complete.
```

```
Admin UI: http://${ML_HOST}:8001
Staging REST: http://${ML_HOST}:${STAGING_PORT}
Final REST:   http://${ML_HOST}:${FINAL_PORT}"
```

modules/transforms/staging-metadata.sjs

```
/**
 * Attach staging metadata & collections.
 * Params (transform-params):
 *   entityType: patient|doctor|ehr
 *   source: string (e.g., EPIC, Cerner)
 *   ingestDate: yyyy-mm-dd (optional; defaults to today)
 */
'use strict';

var xdmp = require('/MarkLogic/xdmp.js');

function today() {
  const d = new Date();
  return d.toISOString().slice(0,10);
}

exports.transform = function(context, params, content) {
  const entity = (params['entityType']||'ehr').toLowerCase();
  const src = params['source'] || 'unknown';
  const dateStr = params['ingestDate'] || today();

  // Collections
  const cols = new Set(content.collections || []);
  ['stage', entity, 'swarmcare', `date:${dateStr}`].forEach(c => cols.add(c));

  // Metadata (properties)
  const meta = content.metadata || {};
  meta['swarmcare:stage'] = 'staging';
  meta['swarmcare:entityType'] = entity;
  meta['swarmcare:source'] = src;
  meta['swarmcare:ingestedAt'] = new Date().toISOString();

  return {
    uri: content.uri,
    value: content.value,
    collections: Array.from(cols),
    metadata: meta
  };
};
```

modules/transforms/final-metadata.sjs

```
/**
 * Attach final metadata & collections when promoting curated docs.
 * Params:
 *   entityType: patient|doctor|ehr
 *   provenance: free-text or job id
 *   version: semantic version string
 */
'use strict';

exports.transform = function(context, params, content) {
  const entity = (params['entityType']||'ehr').toLowerCase();
  const prov = params['provenance'] || 'unknown';
  const version = params['version'] || '1.0.0';

  const cols = new Set(content.collections || []);
  // ensure final flags
  ['final', entity, 'swarmcare', 'curated'].forEach(c => cols.add(c));

  const meta = content.metadata || {};
```

```
  meta['swarmcare:stage'] = 'final';
  meta['swarmcare:entityType'] = entity;
  meta['swarmcare:provenance'] = prov;
  meta['swarmcare:version'] = version;
  meta['swarmcare:promotedAt'] = new Date().toISOString();

  return {
    uri: content.uri,
    value: content.value,
    collections: Array.from(cols),
    metadata: meta
  };
};
```

scripts/etl-ingest.sh

```
#!/usr/bin/env bash
set -euo pipefail

HOST="${HOST:-localhost}"
PORT="${PORT:-8040}"            # staging REST
ADMIN_USER="${MARKLOGIC_ADMIN_USERNAME:-admin}"
ADMIN_PASS="${MARKLOGIC_ADMIN_PASSWORD:-admin123}"

auth=(-u "${ADMIN_USER}:${ADMIN_PASS}" --anyauth -sS)

# Ingest helper: (file, uri, entityType, source)
ingest() {
  local file="$1"; local uri="$2"; local entity="$3"; local source="$4"
  echo "■ Ingesting ${file} -> ${uri} (entity=${entity}, source=${source})"
  curl "${auth[@]}" -X PUT     -H "Content-Type: application/json"     --data-binary @"${file}"     "http://${HO
}

# EHR
for f in ./data/staging/ehr/*.json; do
  [ -e "$f" ] || continue
  base=$(basename "$f")
  ingest "$f" "/swarmcare/stage/ehr/${base}" "ehr" "EHR-SYSTEM"
done

# Patient
for f in ./data/staging/patient/*.json; do
  [ -e "$f" ] || continue
  base=$(basename "$f")
  ingest "$f" "/swarmcare/stage/patient/${base}" "patient" "PATIENT-INTAKE"
done

# Doctor
for f in ./data/staging/doctor/*.json; do
  [ -e "$f" ] || continue
  base=$(basename "$f")
  ingest "$f" "/swarmcare/stage/doctor/${base}" "doctor" "PROVIDER-REG"
done

echo "■ Staging ingest complete."
```

scripts/etl-promote.sh

```
#!/usr/bin/env bash
set -euo pipefail

HOST="${HOST:-localhost}"
STAGING_PORT="${STAGING_PORT:-8040}"
FINAL_PORT="${FINAL_PORT:-8041}"
ADMIN_USER="${MARKLOGIC_ADMIN_USERNAME:-admin}"
ADMIN_PASS="${MARKLOGIC_ADMIN_PASSWORD:-admin123}"
auth=(-u "${ADMIN_USER}:${ADMIN_PASS}" --anyauth -sS)
```

```
# Promote helper: read from staging, write to final with metadata
promote() {
  local stageUri="$1"; local finalUri="$2"; local entity="$3"; local provenance="$4"; local version="$5"

  echo "■ Promoting ${stageUri} -> ${finalUri} (entity=${entity})"

  # 1) GET from staging
  tmp="$(mktemp)"; trap 'rm -f "$tmp"' EXIT
  curl "${auth[@]}" -X GET "http://${HOST}:${STAGING_PORT}/v1/documents?uri=${stageUri}" -o "$tmp" -H "Accept: a

  # TODO: invoke your harmonizer here (map to FHIR, enrich with ontology, etc.)
  # e.g. run a jq transform or call /v1/eval to server-side process.

  # 2) PUT to final with final-metadata transform
  curl "${auth[@]}" -X PUT     -H "Content-Type: application/json"     --data-binary @"$tmp"     "http://${HOST}
}

# Example promotions (adjust patterns as needed)
for src in $(curl "${auth[@]}" -G "http://${HOST}:${STAGING_PORT}/v1/search"     --data-urlencode 'q=collection:
  base="${src##*/}"
  promote "${src}" "/swarmcare/final/ehr/${base}" "ehr" "harmonizer-v1" "1.0.0"
done

echo "■ Promotion complete."
```

Data folder layout

```
data/
  staging/
    ehr/        # drop initial EHR/Episode/Report JSON here
    patient/    # drop patient registration JSON here
    doctor/     # drop provider JSON here
```

Run instructions

```
# 0) bring up MarkLogic
docker compose up -d

# 1) bootstrap (creates forests, databases, properties, REST servers, installs transforms)
docker exec -it marklogic-swarmcare bash -c "chmod +x /opt/scripts/bootstrap-marklogic.sh && /opt/scripts/bootst

# 2) (from your host) ingest staged docs
bash scripts/etl-ingest.sh

# 3) promote curated docs to final
bash scripts/etl-promote.sh
```

Architecture notes (summary)

```
- Schemas / Ontology
  - Schemas-DB (schema-database for all content DBs)
  - KnowledgeGraph-DB with triple-index enabled for RDF/triples (SNOMED, LOINC, RxNorm, payer policies, KG edges

- Staging vs Final (Data Hub–style separation)
  - Staging DBs: Staging-EHR-DB, Staging-Patient-DB, Staging-Doctor-DB
  - Final DBs: Final-EHR-DB, Final-Patient-DB, Final-Doctor-DB

- Forests
  - One forest per DB at first run (e.g., Staging-EHR-DB-Forest)

- Collections & Metadata
  - Staging transform adds: stage, {entityType}, swarmcare, date:YYYY-MM-DD + metadata (swarmcare:stage, source,
  - Final transform adds: final, {entityType}, swarmcare, curated + provenance/version/timestamps.

- REST Servers
```

  - swarmcare-staging on 8040 -> defaults to Staging-EHR-DB
  - swarmcare-final on 8041 -> defaults to Final-EHR-DB

- ETL
  - Landing: etl-ingest.sh puts raw docs into staging with transform-based metadata & collections.
  - Promote: etl-promote.sh copies docs to final with curated flags; insert harmonization/ontology enrichment in

Additional notes

Notes: - This package is intended to be dropped directly into a project folder. Place the files under the matching paths (docker-compose.yml at repo root, modules/, scripts/, data/). Transforms are installed into the modules/transforms path and referenced by the bootstrap script. If you'd like, I can also generate an archive (zip) containing these files exactly in the correct directory structure so you can hand it to your code assistant.