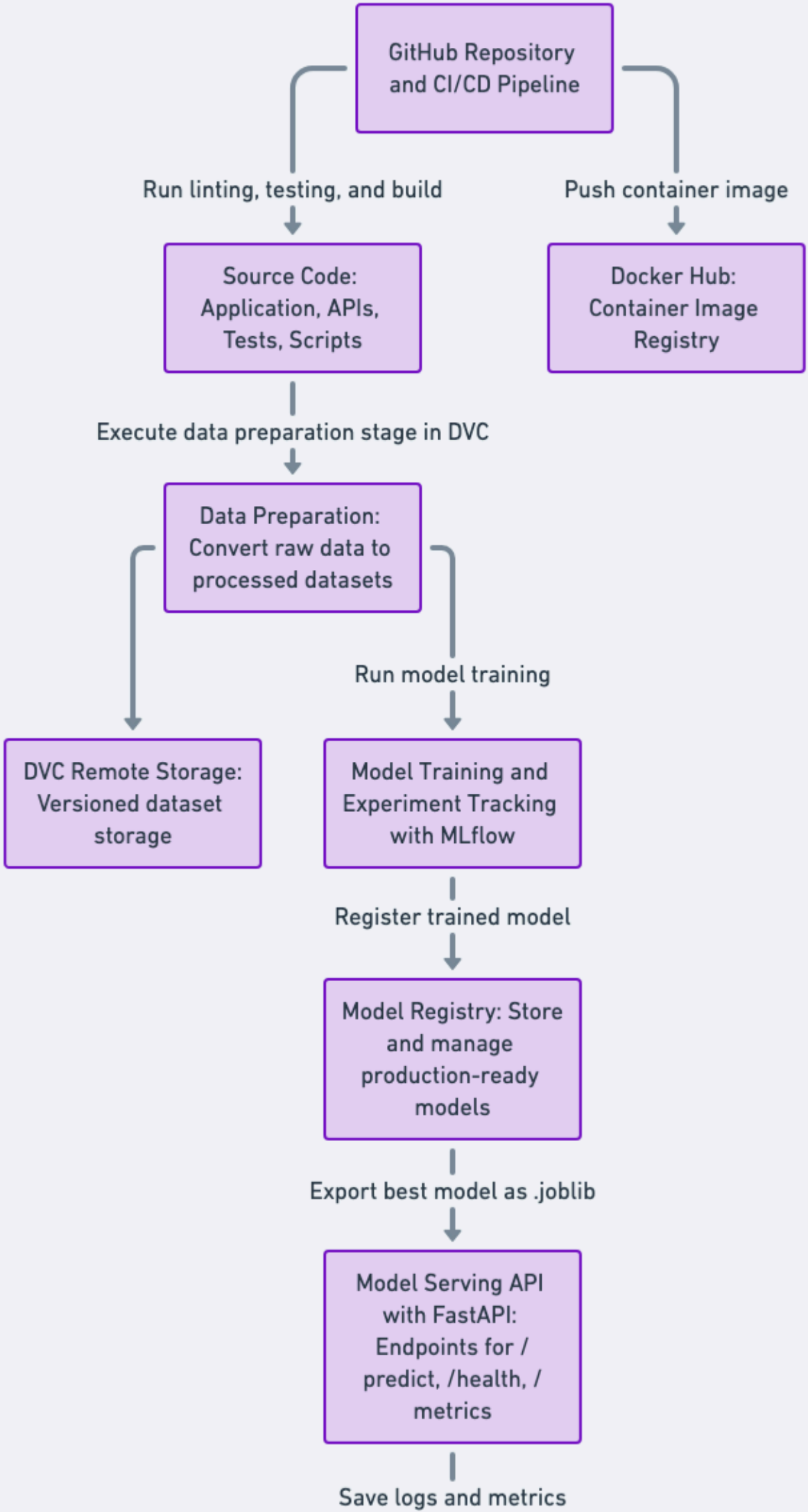
# MLOps Iris — Assignment Summary

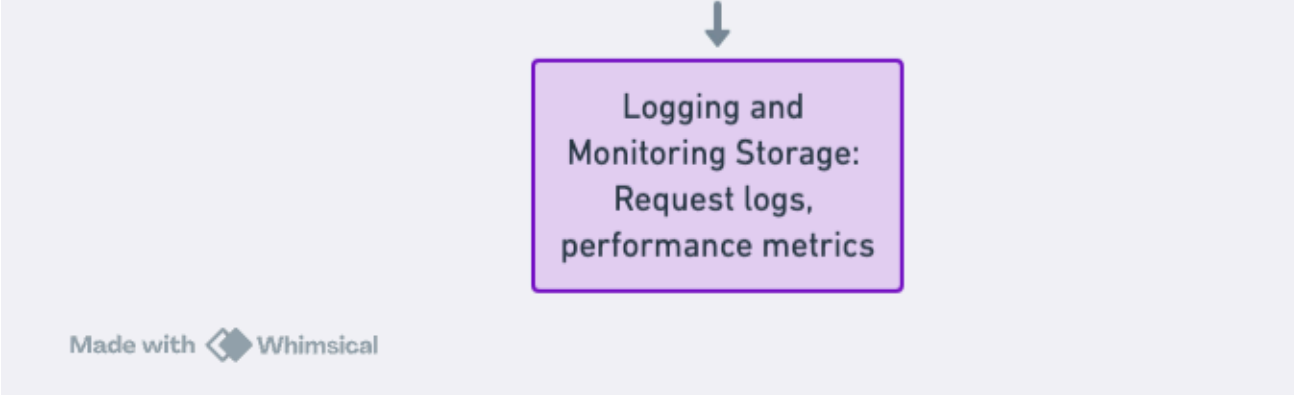
**Goal:** Build, track, package, deploy, and monitor a minimal end-to-end ML system using the **Iris** dataset.

**Stack:** Git/GitHub · DVC · MLflow · FastAPI · Docker · GitHub Actions · Logging+SQLite · Prometheus

---

## 1. Architecture (at a glance)


GitHub Repository and CI/CD Pipeline
Run linting, testing, and build
Push container image
Source Code: Application, APIs, Tests, Scripts
Docker Hub: Container Image Registry
Execute data preparation stage in DVC
Data Preparation: Convert raw data to processed datasets
Run model training
DVC Remote Storage: Versioned dataset storage
Model Training and Experiment Tracking with MLflow
Register trained model
Model Registry: Store and manage production-ready models
Export best model as .joblib
Model Serving API with FastAPI: Endpoints for /predict, /health, /metrics
Save logs and metrics




---

## 2. Repository Layout

```
api/                FastAPI app (main.py, schemas)
artifacts/model/    Deployed model (model.joblib)
data/               Raw/processed CSV + schema/metadata
scripts/            run\_mlflow\.sh, deploy.sh, bootstrap.sh
src/                data.py (ETL), train.py, utils.py
tests/              unit tests (data, API, metrics)
.github/workflows/  CI (build/test/push)
dvc.yaml            DVC pipeline (prepare\_data stage)
Makefile            Common tasks (dev/CI parity)
```

---

## 3. What we built (per assignment parts)

### Part 1 — Repository & Data Versioning (4/4)

- **ETL:** `src/data.py` writes:
  - `data/raw/iris.csv`, `data/processed/iris.csv`
  - `data/schema.json` (contract), `data/metadata.json` (checksum, provenance)
- **DVC:** `dvc.yaml` stage `prepare_data` + idempotent remote setup.
- **Make targets:** `make data`, `make dvc-init`, `make dvc-remote`, `make dvc-repro`, `make dvc-push/pull`.

### Part 2 — Model Dev & Experiment Tracking (6/6)

- **Models:** LogisticRegression (with StandardScaler), RandomForest.
- **Tracking:** MLflow logs params, metrics (acc, precision/recall/f1 macro), confusion matrix artifact, signature + input example.
- **Selection:** Best by accuracy → exported to `artifacts/model/model.joblib`.
- **Registry:** Registers `iris_clf` and assigns alias `production` (falls back to stage "Production" if aliases unsupported).

### Part 3 — API & Docker (4/4)

- **API:** FastAPI with Pydantic v2.
  - `POST /predict` → class id/name, optional probabilities
  - `GET /health` → load status + model path
  - `GET /metrics` → Prometheus text format
- **Docker:** Slim image, lockfile install, exposes `8000`.

### Part 4 — CI/CD (6/6)

- **Actions:** Install from **requirements.lock.txt**, lint, tests.
- **Docker:** Build & push multi-arch image on `main` (+ tag `latest` & short SHA).
- **Smoke test:** Container `/health` check in workflow.

### Part 5 — Logging & Monitoring (4/4)

- **Logs:** Rotating file logs `logs/app.log`.
- **SQLite:** `logs/predictions.db` stores request/response audit.
- **Metrics:** Prometheus counters + latency histogram at `/metrics`.

### Part 6 — Summary + Demo (2/2)

- You're reading the 1-page summary ✅
- 5-min walkthrough script below.

**Bonus**

- ✅ Input validation (Pydantic v2).
- ✅ Prometheus endpoint.
- ❇️ Grafana/Prometheus compose + retrain trigger: stubs can be added.

---

## 4. How to run (local, reproducible)

```
# Setup (Python 3.11.x)
make setup-lock

# Part 1: data artifacts
make data
make test

# MLflow (auto-port, prints URL)
./scripts/run_mlflow.sh
export MLFLOW_TRACKING_URI=http://127.0.0.1:<printed-port>

# Part 2: train & register
python src/train.py

# Part 3: API
make api
# In another shell:
curl -s http://127.0.0.1:8000/health | jq
curl -s -X POST http://127.0.0.1:8000/predict -H 'content-type:
application/json' \
  -d
'{"sepal_length":5.1,"sepal_width":3.5,"petal_length":1.4,"petal_width":0.2}'
| jq
curl -s http://127.0.0.1:8000/metrics | head
```

> One-command demo: `./scripts/bootstrap.sh` (or `make bootstrap`) Does venv → deps → data → tests → MLflow → train → API.

---

## 5. CI/CD Overview

- Trigger: push/PR to `main`.
- Steps:
  1. Install from `requirements.lock.txt` (reproducible).
  2. Lint (flake8) + run tests (pytest).
  3. Build Docker image → tag `latest` and `${{ github.sha::7 }}`.
  4. Push to Docker Hub.
  5. Smoke test `/health`.

**Local parity:** `make ci-install` installs from lockfile like CI.

---

## 6. Monitoring & Logging

- **Metrics** (`/metrics`):
  - `predict_requests_total`
  - `predict_latency_seconds` (histogram)
- **Logs:**
  - App logs → `logs/app.log` (rotating).
  - SQLite audit → `logs/predictions.db` (`log_prediction()` in `src/utils.py`).

---

## 7. Validation & Tests

- `tests/test_part1_validation.py` → schema, checksum, numeric dtypes, label domain.
- `tests/test_api.py` → startup health, happy-path predict, validation errors, `/metrics`.
- Run locally: `make test` (ensures `make data` first).

---

## 8. Deliverables

- **GitHub repo:** `https://github.com/gregariousgovind/mlops-iris`
- **Docker Hub image:** `gregariousgovind/mlops-iris:latest`
- **Summary doc:** `SUMMARY.md`
- **5-min screen recording:** `demo.mp4`

---

## 9. Decisions & Notes

- **Dataset:** Iris (small, deterministic; DVC used to demonstrate reproducible data pipeline).
- **Model choice:** Simple baselines (LogReg, RF) are enough to show tracking/selection.
- **Reproducibility:** Python 3.11.x, pinned lockfile, seeds, run tags with data checksum.
- **Safety:** Validation via Pydantic; API returns 422 for schema errors.

---

## 10. Future Extensions

- Prometheus + Grafana docker-compose with a basic dashboard.
- Scheduled retraining on new data (GitHub Actions workflow + DVC tracked dataset).

- Canary or shadow deployments; model-versioned traffic splitting.
- Great Expectations or Pandera data quality checks in CI.