CCT College Dublin

Assessment Cover Page

Module Title:	Diploma in Devops
Assessment Title:	Capstone Project: Book Catalog with CI/CD Pipeline
Lecturer Name:	Esteban Garcia
Student Full Name:	Egshiglen Enkhbayar
Student Number:	2024359
Assessment Due Date:	Friday 8th August 2025 – 23:59 (11:59 pm)
Date of Submission:	Friday 8th August 2025

Declaration

Contents

1. Introduction

Demo Video Link: https://drive.google.com/file/d/1NxA__cKcy-rY1JbEnpt1PRFRszDmJ6wl/view?usp=sharing (check here to see the whole deployment and app operation in action)

Github Repo Link: https://github.com/egshiglen2024359/capstone-project

This report is about my final Capstone Project for the DevOps Diploma 2025. When I first started this course, I was confident using basic coding tools but I hadn't done a real end-to-end project with automation, containerization, Kubernetes, or ArgoCD. This Capstone was my chance to take what I learned in class and put it into something that actually works from start to finish.

The goal of the project was to build a Book Catalog API using Django REST Framework and then go much further — containerize it, automate its testing and deployment, and run it in a Kubernetes cluster. That meant learning not just how to write the app, but how to package it with Docker, deploy it with Helm, set up CI/CD pipelines in GitHub Actions, and even configure GitOps with Argo CD so deployments happen automatically when the code changes.

The Book Catalog API is a simple idea - a place to store and manage information about books. Each book has:

- **Title** — The name of the book.
- **Author** — Who wrote it.
- **ISBN** — The book's unique number.
- **Published date** — When it came out.
- **Status** — Whether it's available, checked out or archived.

You can add, view, update, and delete books through the API. The simplicity of the app meant I could focus my energy on the DevOps side, which was the point of this Capstone.

Why these technologies?

- **Django REST Framework** — Fast to set up, great for building APIs
- **PostgreSQL** — A reliable database with good support in Django.
- **Docker** — To package the app so it runs the same everywhere.
- **Helm** — To manage Kubernetes deployments more cleanly than writing YAML files by hand.
- **GitHub Actions** — For CI/CD without needing extra services.
- **Argo CD** — To manage Kubernetes deployments with GitOps, so the cluster always matches the code in my repo.

This project wasn't just about delivering working code; it was about building the entire delivery pipeline from development to production.

2. End-to-End Workflow

From the very start, I wanted a smooth process where I could:

1. Work on my code locally.
2. Push to GitHub.
3. Let automation handle testing, building, and deploying.
4. See the updated version live in Kubernetes without manually running extra commands.

Here's how that works in practice.

Step 1: Local Development

I started by writing my Django app locally. I used Docker Compose so I could run both the Django app and a PostgreSQL database together with one command:

```
docker-compose up --build
```

The --build flag makes sure any code changes are reflected in the running container.

My entrypoint.sh waits for the database to be ready, applies migrations with:

```
python manage.py migrate
```

and then starts the development server.

This means I don't need to run migrations manually in local development, it just works.

If I want to run tests before committing, I can do with:

```
docker-compose exec app pytest
```

This helps me catch issues early before pushing to GitHub.

I also keep my environment variables in a .env file, which Docker Compose reads automatically so the local environment matches production.

Step 2: Version Control

I used Conventional Commits throughout the project (e.g., feat, fix, chore, ci, test with scopes like fix(ingress) and feat(helm)). I committed frequently and kept changes small so the history is easy to follow and semantic-release could infer new versions reliably. I also used pull requests for some changes (e.g., deploy/helm, feat/push-image) to practice review workflows. Release commits are generated automatically by the pipeline (e.g., chore(release): 1.6.3 [skip ci]), followed by a bot commit that updates environments/production/values.yaml with the new image tag. One lesson I learned was to avoid misusing [skip ci] on manual commits because it can prevent releases and deployments from triggering.

- feat(api): add BookView and BookDetailView with URL routes and comments
- build(docker): add Dockerfile and .dockerignore for container setup
- ci(workflow): refactor test pipeline and add Docker build and push job
- fix(django): allow all hosts for Kubernetes testing

This makes my Git history easy to read and allows tools like semantic-release to understand my changes.

Step 3: Continuous Integration

Whenever I push to main or open a pull request, my GitHub Actions workflow starts. The first stage runs all the tests with:

```
pytest ./api/tests/test_views.py
```

This ensures nothing breaks before going further.

The workflow also:

- Runs python manage.py migrate in the runmigrations job to confirm migrations apply cleanly.
- Runs python manage.py makemigrations --check in the migrations-check job to catch any uncommitted model changes.

Step 4: Semantic Release

If all tests and migration checks pass, semantic-release looks at my commits and decides whether to make a patch, minor, or major version bump. It follows Semantic Versioning (MAJOR.MINOR.PATCH). It then updates the changelog, tags the release in GitHub, and sets the new version number.

Step 5: Build and Push Docker Image

The next stage logs into GHCR (GitHub Container Registry) using the built-in GITHUB_TOKEN (no manual login needed in CI) and builds the Docker image:

```
docker build -t ghcr.io/egshiglen2024359/capstone-project:<version> .
```

Then it pushes that image to GHCR, where Kubernetes can pull it later.

Step 6: Update Helm Values

The workflow then edits environments/production/values.yaml to set:

```
image:
  tag: <new version>
```

This is committed back to the main branch. This step is important because my Helm chart uses image.tag to know exactly which image to deploy.

Step 7: Continuous Deployment with Argo CD

Since Argo CD is connected to my repo and watching books-catalog-chart/ with the production values file, it sees that values.yaml changed. It automatically syncs the Kubernetes cluster to match the repo, pulling the new Docker image and rolling out the updated app.

3. Docker Image Build Process

The Dockerfile for this project is designed to be small, efficient, and easy to understand.

1. FROM python:3.12-slim

I started with a lightweight Python image. This helps reduce image size and avoids shipping unnecessary tools.

2. ENV

```
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
```

These make Python skip .pyc files and log output immediately, which is useful for debugging.

3. **WORKDIR /app**

This sets /app as the working directory for all commands.

4. **Install Python Dependencies**

I copied requirements.txt:

```
COPY requirements.txt .
```

5. **System Dependencies**

I installed the system dependencies and Python packages in a single RUN instruction to reduce image layers and improve build efficiency:

```
RUN apt-get update && \
    apt-get install -y gcc libpq-dev netcat-traditional && \
    pip install --upgrade pip && \
    pip install -r requirements.txt && \
    rm -rf /var/lib/apt/lists
```

- gcc and libpq-dev are needed to compile Python packages like psycopg2.
- netcat-traditional is used in entrypoint.sh to check if the database is ready before starting Django.
- rm -rf /var/lib/apt/lists cleans up cached package lists to keep the image size smaller.

6. **Copy Application Code**

```
COPY . .
```

This puts the rest of the app into the image.

7. **Entrypoint**

```
ENTRYPOINT ["/app/entrypoint.sh"]
```

This script runs migrations and then starts Django.

Why containerization matters:

- The app runs exactly the same in development, CI/CD, and production.
- Kubernetes just needs the image - no worrying about environment differences.
- It makes scaling easier if needed in the future.

4. CI/CD Pipeline Explanation

The GitHub Actions workflow is broken into several jobs:

1. **test**
   - Checks out the code from the repo.

- Installs Python dependencies from requirements.txt.
- Runs all unit tests with pytest ./api/tests/test_views.py.
- Ensures the application logic works before continuing.

2. **runmigrations**
    - Runs python manage.py migrate inside the CI environment to confirm that the current migrations can be applied successfully to the database.
3. **migrations-check**
    - Runs python manage.py makemigrations --check to detect any untracked model changes that haven't been committed. This prevents schema drift between code and database.
4. **semantic-release**
    - Scans commit messages that follow the **Conventional Commits** format to determine the new version number using **Semantic Versioning** (MAJOR.MINOR.PATCH).
    - Updates the CHANGELOG.md file, creates a new GitHub release, and tags the repository with the new version.
5. **build-docker-image**
    - Logs in to the GitHub Container Registry (**GHCR**) using the built-in GITHUB_TOKEN.
    - Builds a Docker image from the application code with the new semantic version as the tag.
    - Pushes the image to GHCR at:

      ghcr.io/egshiglen2024359/capstone-project:<version>
6. **deploy-application**
    - Uses the fjogeleit/yaml-update-action to automatically update environments/production/values.yaml and set:

      ```
      image:
        tag: <new version>
      ```
    - Commits this change back to main. This update is the trigger for Argo CD to deploy the new version.
7. **Argo CD auto-sync** *(outside the workflow file)*
    - Argo CD is configured to monitor the Helm chart directory (books-catalog-chart/) with the production values.yaml.
    - When it detects the updated image tag, it automatically syncs the Kubernetes cluster to match the repo.
    - This includes pulling the new Docker image from GHCR and rolling out the updated deployment.
    - **Self-heal** mode is enabled, so if someone changes something directly in the cluster, Argo CD will revert it to match the repo.

Why Argo CD?
Before using Argo CD, I would have had to run:

helm upgrade --install books-catalog-app ./books-catalog-chart manually after every change. With Argo CD's GitOps model, I simply push code to GitHub, and the deployment happens automatically. The repo is the single source of truth, ensuring the cluster always matches the version in source control.

5. Helm Chart Deployment

My Helm chart (books-catalog-chart/) defines everything needed to run the Book Catalog API on Kubernetes. It allows me to deploy the app with a single helm upgrade --install command and keeps all configuration in one place.

1. **Deployment**
   - Specifies how many replicas (pods) to run.
   - Defines the container image using:

```
image:
  repository: ghcr.io/egshiglen2024359/capstone-project
  tag: <version>
```

   - Sets the container port (8000 for Django).
   - Mounts environment variables from a ConfigMap and Secret.
   - Uses imagePullSecrets so Kubernetes can pull the private image from GHCR.

2. Service

- Creates a stable internal DNS name for the pods.
- Exposes the app on a ClusterIP so the Ingress controller can forward traffic to it.
- Maps port 80 (external) to port 8000 (container).

3. Ingress

- Routes HTTP requests from outside the cluster to the Service.
- Configured for the Traefik ingress controller.
- Uses the path /api/books/ so requests like /api/books/1/ go to Django.
- Includes annotations to control behavior (e.g., disable SSL redirect for local testing).

4. ConfigMap

- Stores non-sensitive environment variables such as:
  - DATABASE_HOST
  - DATABASE_NAME
  - DATABASE_USER
- These values are injected into the pod so Django can connect to the database.

5. Secret

- Stores sensitive values like:
  - DATABASE_PASSWORD
- Mounted as environment variables in the pod, never stored in plain text in Git.

6. Migration Job Hook

- A Kubernetes Job that runs before the Deployment is created or updated.
- Executes:
    - python manage.py migrate
- Ensures the PostgreSQL schema is up-to-date before the app starts serving requests.
- Also uses imagePullSecrets so it can pull the private image.

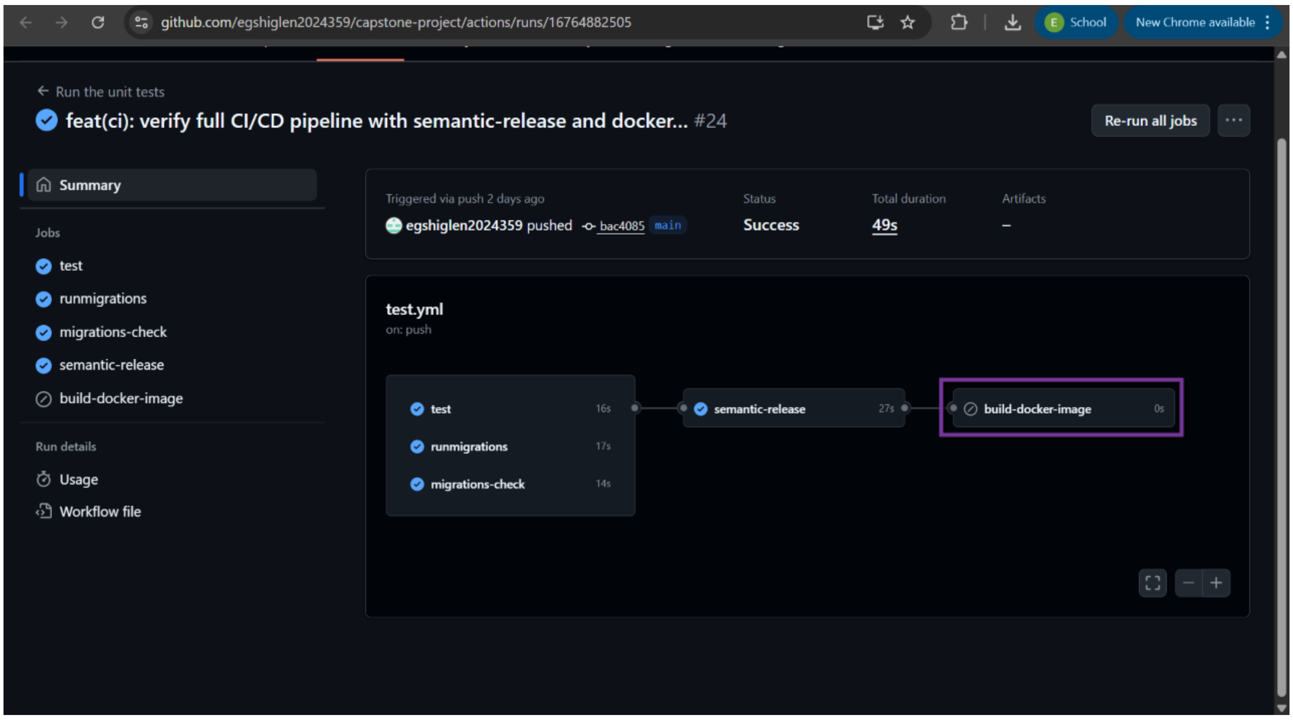
7. values.yaml

- The main configuration file for the chart.
- Allows overriding defaults for:
    - image.repository
    - image.tag
    - replicaCount
    - Environment variables and secrets.
- In CI/CD, this file is updated automatically with the latest image.tag so Argo CD knows which image to deploy.

6. Lessons Learned and Challenges Faced

A) CI/CD & Versioning

1. **Docker build job not running after release**
    - Wrong semantic-release output variable names and missing needs: meant the build never started.


github.com/egshiglen2024359/capstone-project/actions/runs/16764882505

← Run the unit tests
feat(ci): verify full CI/CD pipeline with semantic-release and docker... #24

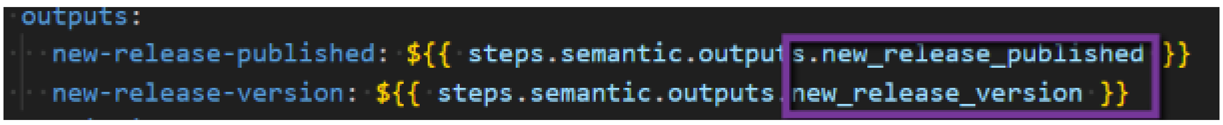
Triggered via push 2 days ago — egshiglen2024359 pushed bac4085 main — Status: Success — Total duration: 49s — Artifacts: –

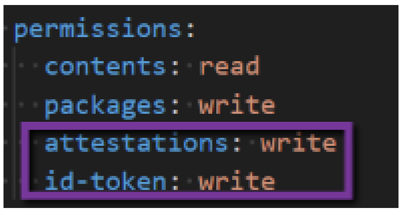
Jobs: test, runmigrations, migrations-check, semantic-release, build-docker-image

test.yml
on: push

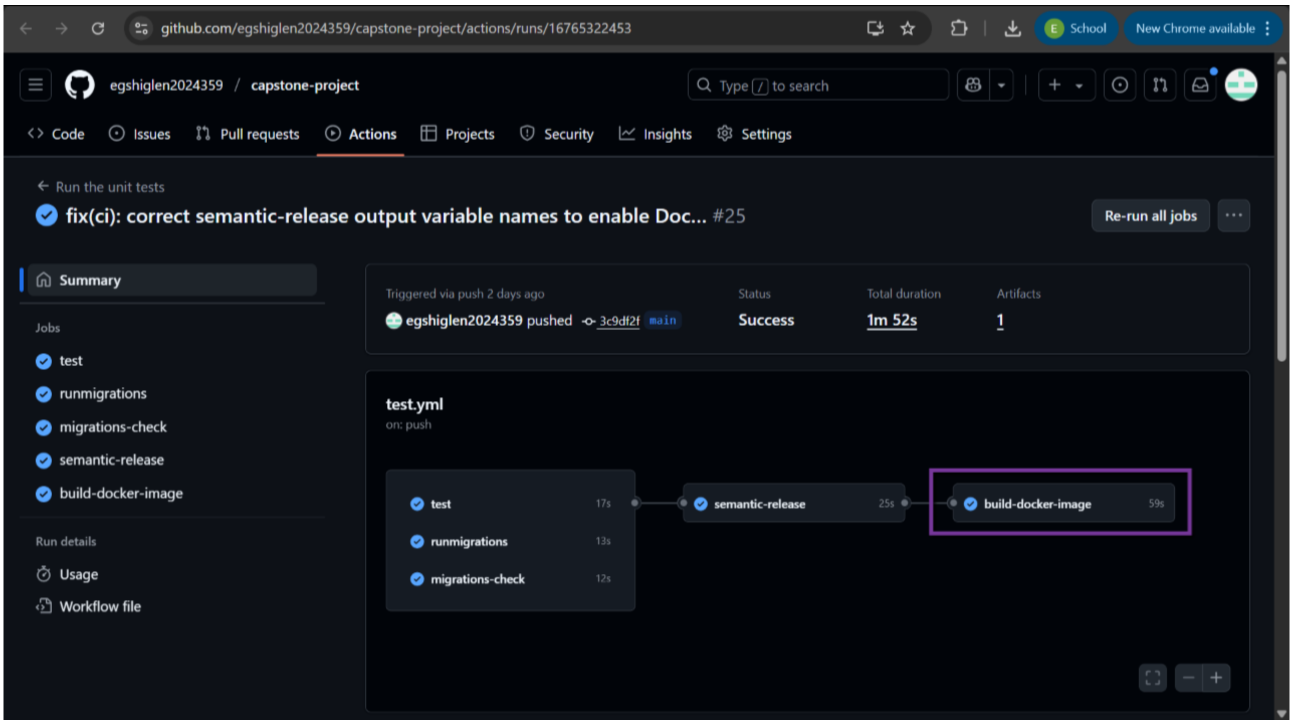
test 16s, runmigrations 17s, migrations-check 14s → semantic-release 27s → build-docker-image 0s

    - Fix: Corrected variable names and job dependencies.


```
outputs:
  new-release-published: ${{ steps.semantic.outputs.new_release_published }}
  new-release-version: ${{ steps.semantic.outputs.new_release_version }}
```



```yaml
permissions:
  contents: read
  packages: write
  attestations: write
  id-token: write
```



github.com/egshiglen2024359/capstone-project/actions/runs/16765322453

egshiglen2024359 / capstone-project

← Run the unit tests

fix(ci): correct semantic-release output variable names to enable Doc... #25

Triggered via push 2 days ago: egshiglen2024359 pushed 3c9df2f main

Status: Success

Total duration: 1m 52s

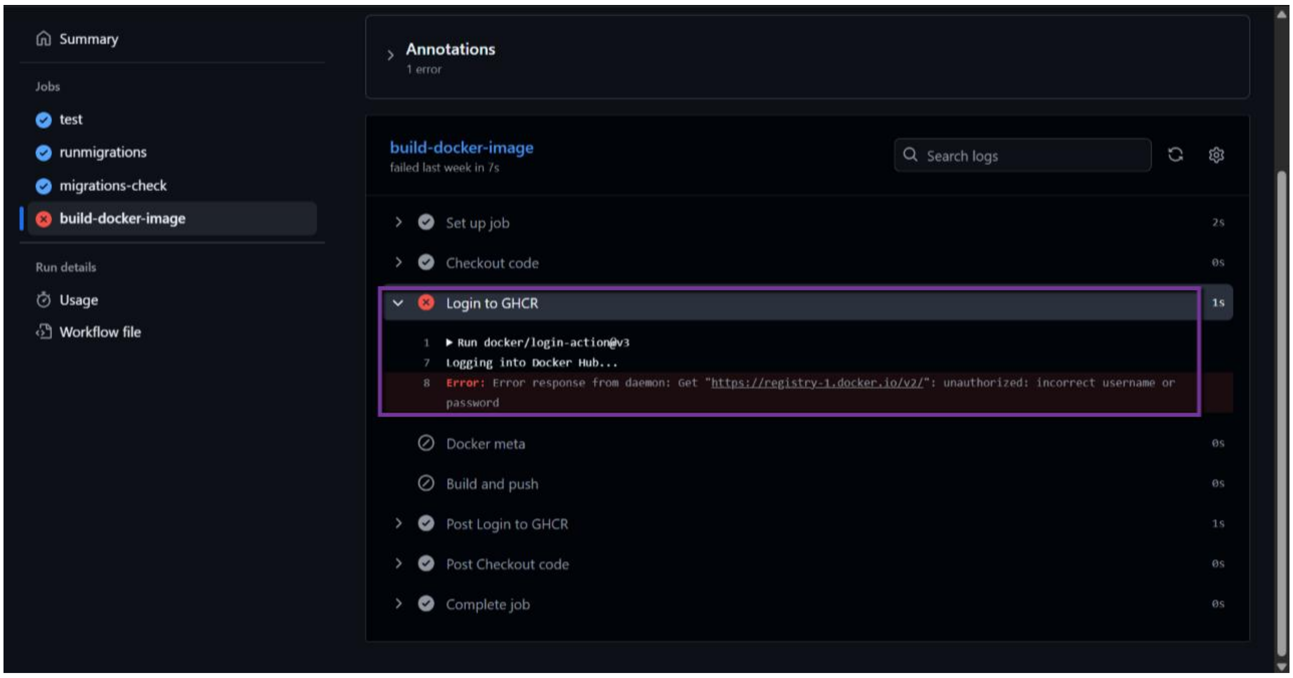
Artifacts: 1

Jobs: test, runmigrations, migrations-check, semantic-release, build-docker-image

test.yml
on: push

```
test 17s
runmigrations 13s
migrations-check 12s
semantic-release 25s
build-docker-image 59s
```

2. **GHCR login failed in build job**
    - Workflow couldn't push images because GHCR login wasn't configured.

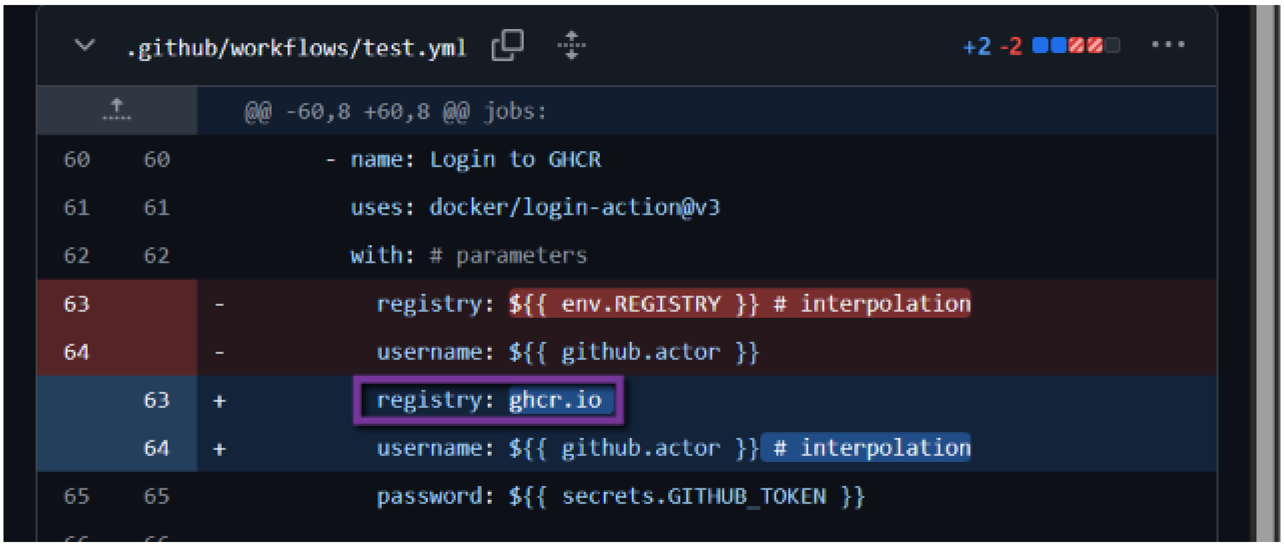


Annotations
1 error

build-docker-image
failed last week in 7s

Jobs: test, runmigrations, migrations-check, build-docker-image

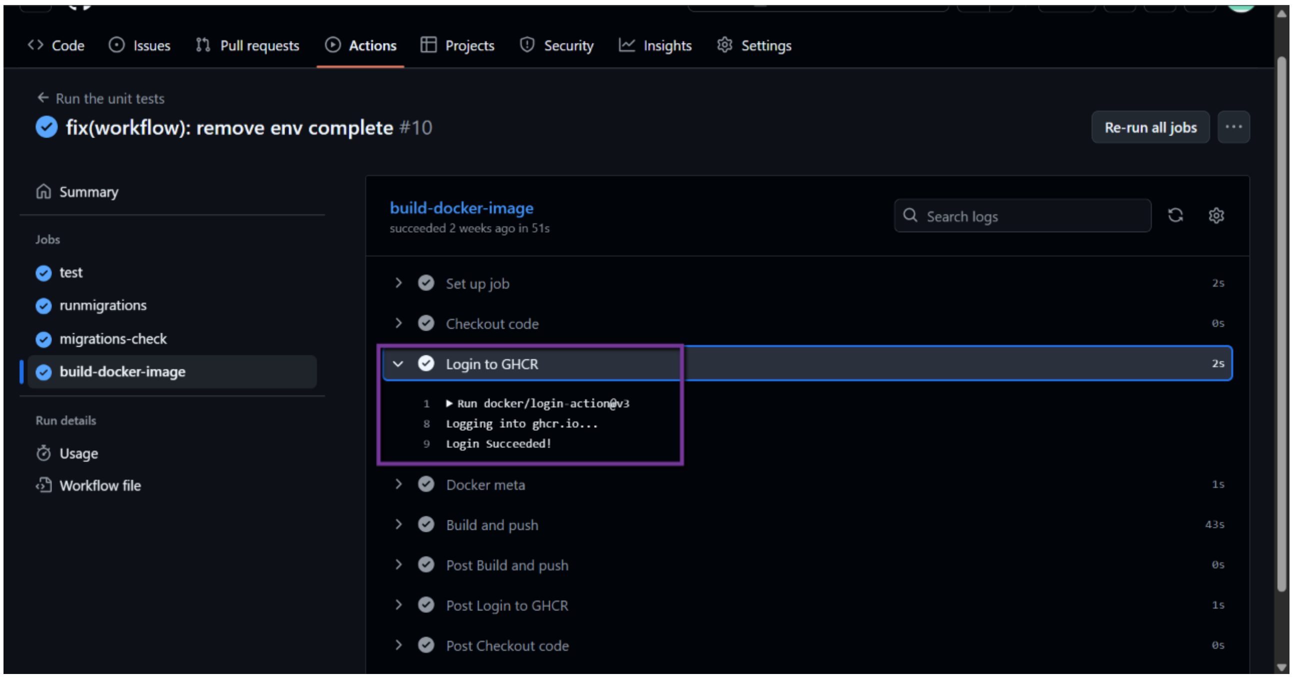
```
Set up job            2s
Checkout code         0s
Login to GHCR         1s
  ▶ Run docker/login-action@v3
  Logging into Docker Hub...
  Error: Error response from daemon: Get "https://registry-1.docker.io/v2/": unauthorized: incorrect username or
  password
Docker meta           0s
Build and push        0s
Post Login to GHCR    1s
Post Checkout code    0s
Complete job          0s
```



.github/workflows/test.yml +2 -2

```diff
@@ -60,8 +60,8 @@ jobs:
          - name: Login to GHCR
            uses: docker/login-action@v3
            with: # parameters
-             registry: ${{ env.REGISTRY }} # interpolation
-             username: ${{ github.actor }}
+             registry: ghcr.io
+             username: ${{ github.actor }} # interpolation
              password: ${{ secrets.GITHUB_TOKEN }}
```

- Fix: Used docker/login-action@v3 with GITHUB_TOKEN and correct permissions.



← Run the unit tests

fix(workflow): remove env complete #10

Summary

Jobs
- ✓ test
- ✓ runmigrations
- ✓ migrations-check
- ✓ build-docker-image

Run details
- Usage
- Workflow file

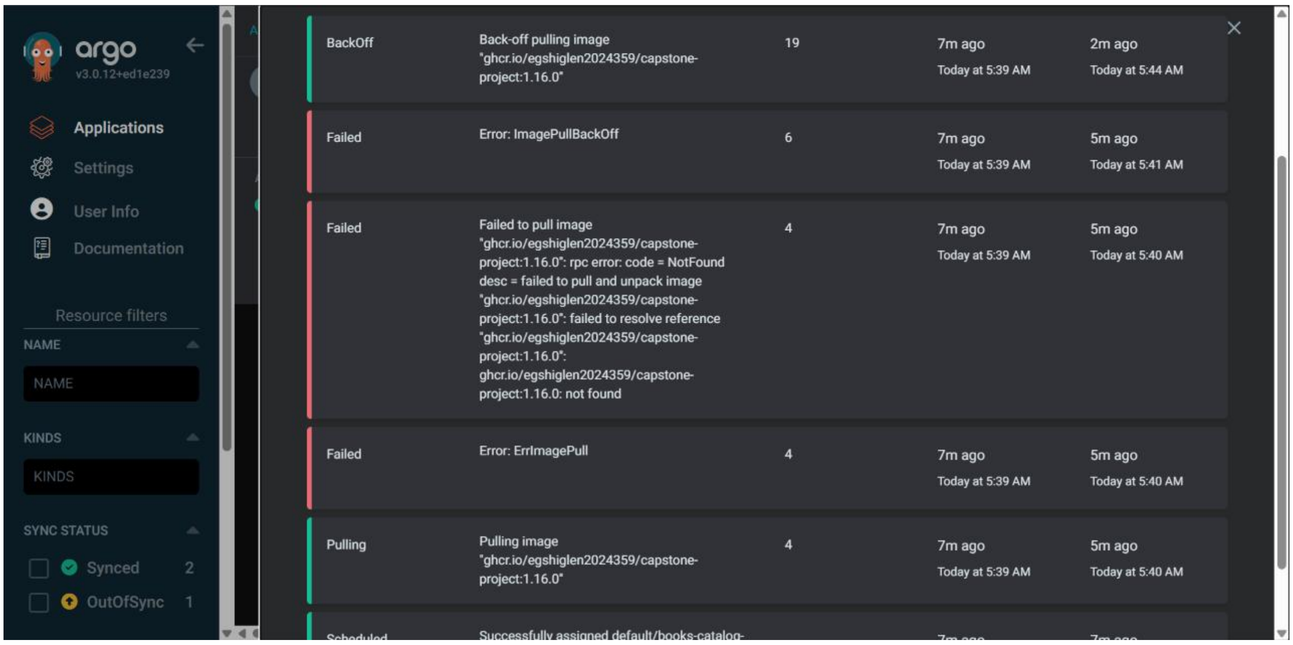
build-docker-image
succeeded 2 weeks ago in 51s

✓ Set up job	2s
✓ Checkout code	0s
✓ Login to GHCR	2s

```
▶ Run docker/login-action@v3
Logging into ghcr.io...
Login Succeeded!
```

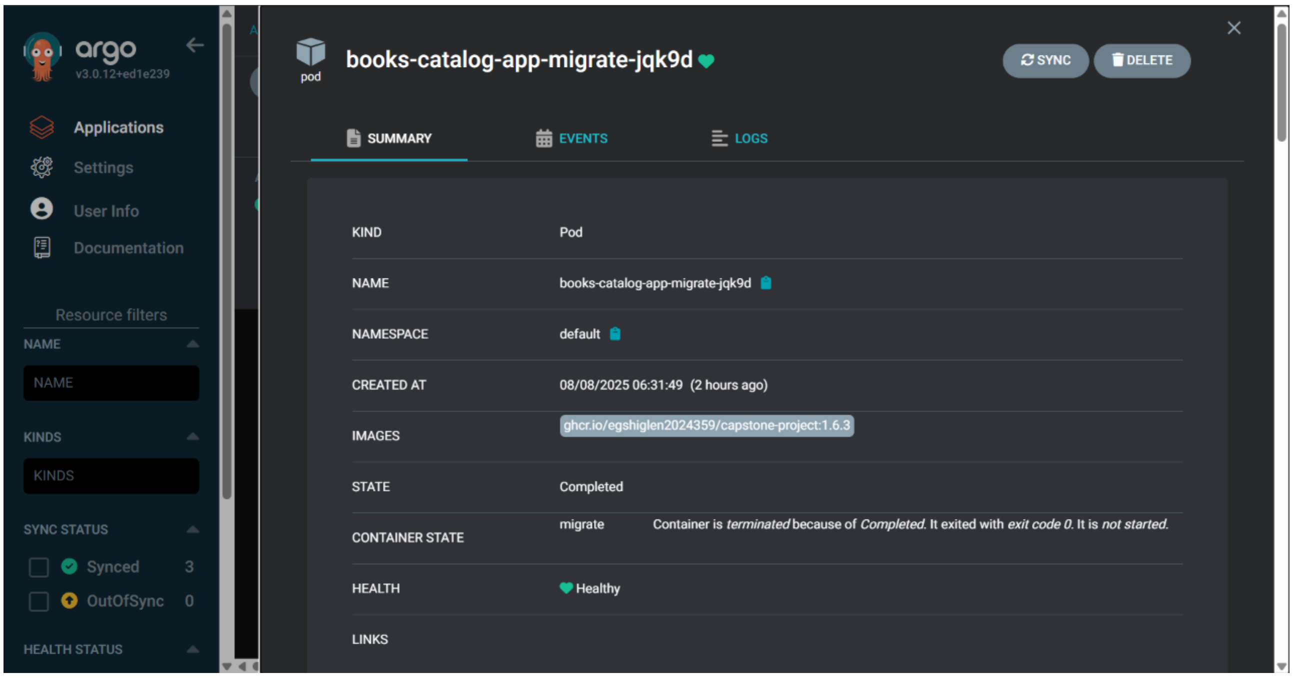
✓ Docker meta	1s
✓ Build and push	43s
✓ Post Build and push	0s
✓ Post Login to GHCR	1s
✓ Post Checkout code	0s

3. **Image tag mismatch / wrong version deployed**
   - Helm fell back to .Chart.AppVersion instead of the new semantic version.



BackOff	Back-off pulling image "ghcr.io/egshiglen2024359/capstone-project:1.16.0"	19	7m ago Today at 5:39 AM	2m ago Today at 5:44 AM
Failed	Error: ImagePullBackOff	6	7m ago Today at 5:39 AM	5m ago Today at 5:41 AM
Failed	Failed to pull image "ghcr.io/egshiglen2024359/capstone-project:1.16.0": rpc error: code = NotFound desc = failed to pull and unpack image "ghcr.io/egshiglen2024359/capstone-project:1.16.0": failed to resolve reference "ghcr.io/egshiglen2024359/capstone-project:1.16.0": ghcr.io/egshiglen2024359/capstone-project:1.16.0: not found	4	7m ago Today at 5:39 AM	5m ago Today at 5:40 AM
Failed	Error: ErrImagePull	4	7m ago Today at 5:39 AM	5m ago Today at 5:40 AM
Pulling	Pulling image "ghcr.io/egshiglen2024359/capstone-project:1.16.0"	4	7m ago Today at 5:39 AM	5m ago Today at 5:40 AM
Scheduled	Successfully assigned default/books-catalog-		7m ago	7m ago

- Fix: CI updates image.tag in values.yaml so Argo CD always deploys the intended image.



books-catalog-app-migrate-jqk9d

KIND	Pod
NAME	books-catalog-app-migrate-jqk9d
NAMESPACE	default
CREATED AT	08/08/2025 06:31:49 (2 hours ago)
IMAGES	ghcr.io/egshiglen2024359/capstone-project:1.6.3
STATE	Completed
CONTAINER STATE	migrate Container is *terminated* because of *Completed*. It exited with *exit code 0*. It is *not started*.
HEALTH	Healthy
LINKS	

B) Kubernetes, Helm & Jobs

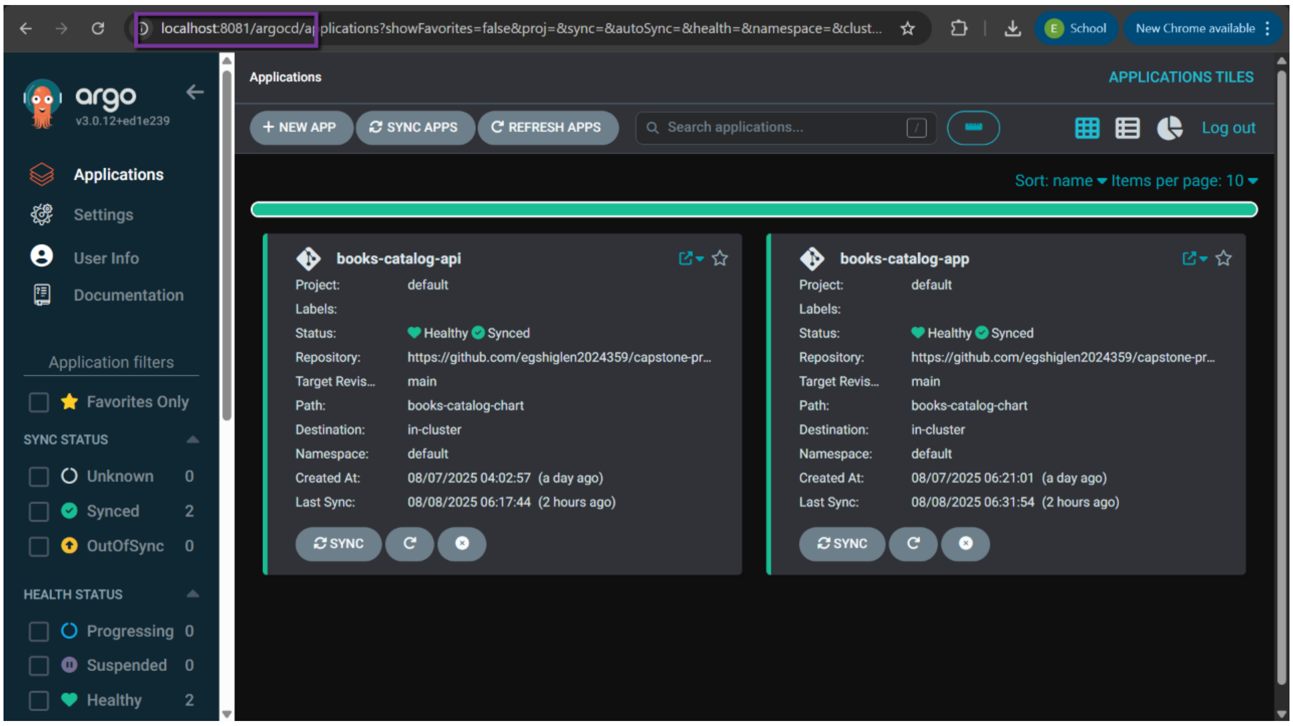
6. **Migration job couldn't pull image**
   - imagePullSecrets missing from Job spec caused ImagePullBackOff.
   - Fix: Added imagePullSecrets: ghcr-token to both Job and Deployment.
7. **Database not ready for migrations**
   - Migrations failed when DB wasn't up yet.
   - Fix: Added wait-for-DB check in entrypoint.sh using netcat.
8. **Installed database chart without values.yaml**

- Used default credentials, so Django couldn't connect.
- Fix: Installed Bitnami Postgres with custom auth.username, auth.password, auth.database values.

9. **Forgot to create ghcr-token secret**
   - Deployment and Job couldn't pull the private image.
   - Fix: Created GHCR registry secret and referenced it everywhere.

C) Ingress & Routing

10. **Argo CD at /argocd inaccessible**
    - Missing server.basehref and server.rootpath for path-prefix in values.
    - Fix: Set both to /argocd in argocd-values.yaml and configured Traefik ingress.



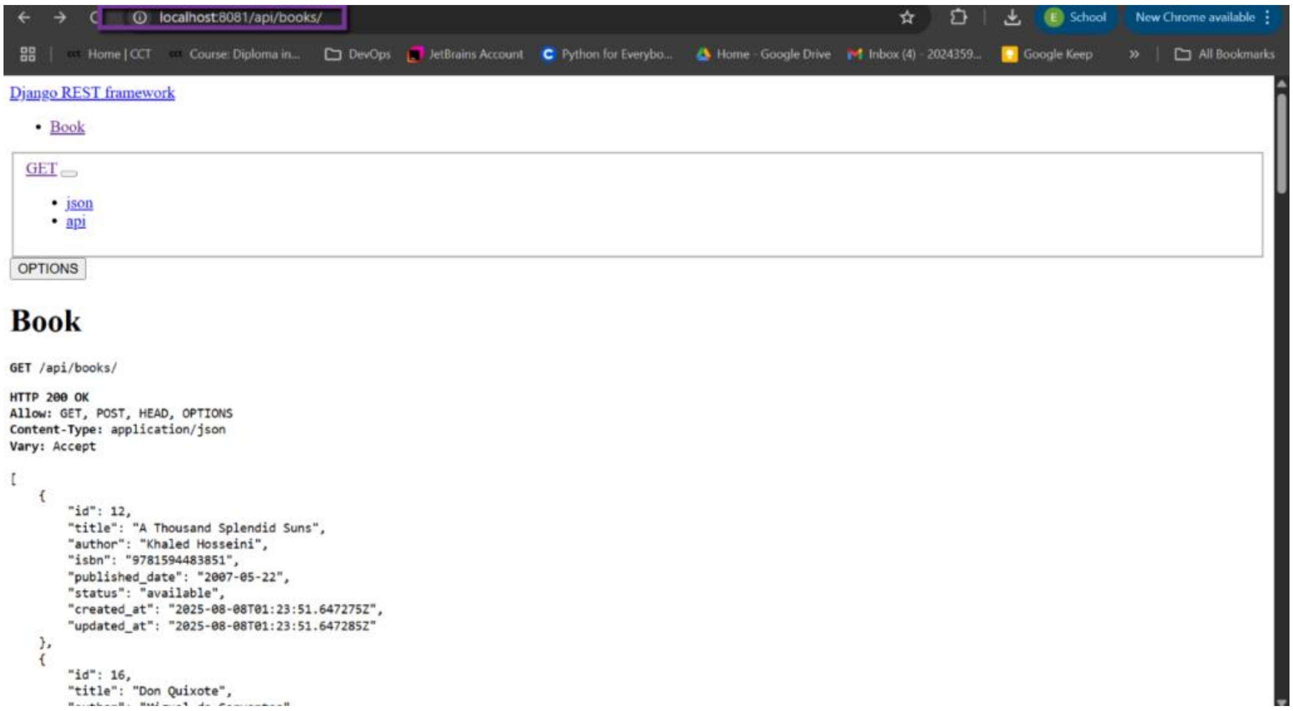
books-catalog-api

Project:	default
Labels:	
Status:	Healthy Synced
Repository:	https://github.com/egshiglen2024359/capstone-pr...
Target Revis...	main
Path:	books-catalog-chart
Destination:	in-cluster
Namespace:	default
Created At:	08/07/2025 04:02:57 (a day ago)
Last Sync:	08/08/2025 06:17:44 (2 hours ago)

books-catalog-app

Project:	default
Labels:	
Status:	Healthy Synced
Repository:	https://github.com/egshiglen2024359/capstone-pr...
Target Revis...	main
Path:	books-catalog-chart
Destination:	in-cluster
Namespace:	default
Created At:	08/07/2025 06:21:01 (a day ago)
Last Sync:	08/08/2025 06:31:54 (2 hours ago)

11. **API at /api/books inaccessible**
    - Ingress path didn't match Django URLs.
    - Fix: Updated both urls.py and ingress path to align.



Book

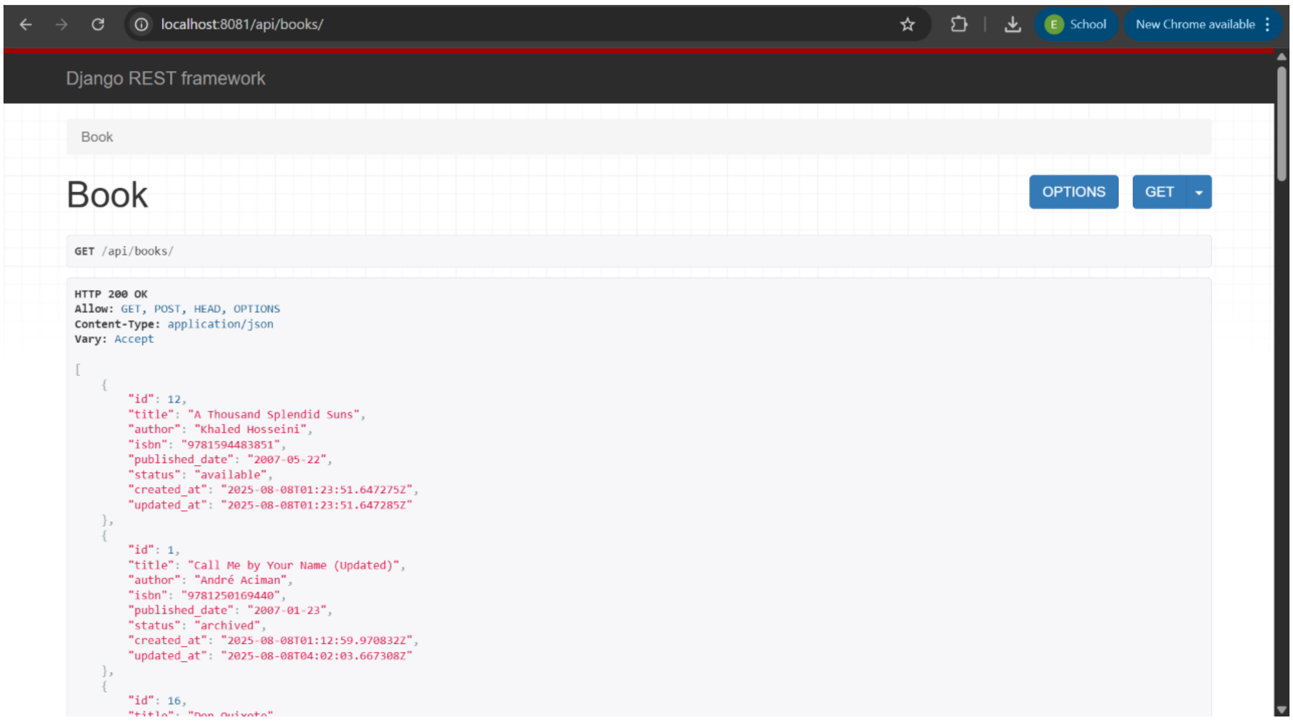
```
GET /api/books/

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 12,
        "title": "A Thousand Splendid Suns",
        "author": "Khaled Hosseini",
        "isbn": "9781594483851",
        "published_date": "2007-05-22",
        "status": "available",
        "created_at": "2025-08-08T01:23:51.647275Z",
        "updated_at": "2025-08-08T01:23:51.647285Z"
    },
    {
        "id": 16,
        "title": "Don Quixote",
```

12. **DRF UI missing CSS**

    - Static/media files weren't routed via ingress.
    - Fix: Added /static and /media paths in ingress rules.



Book

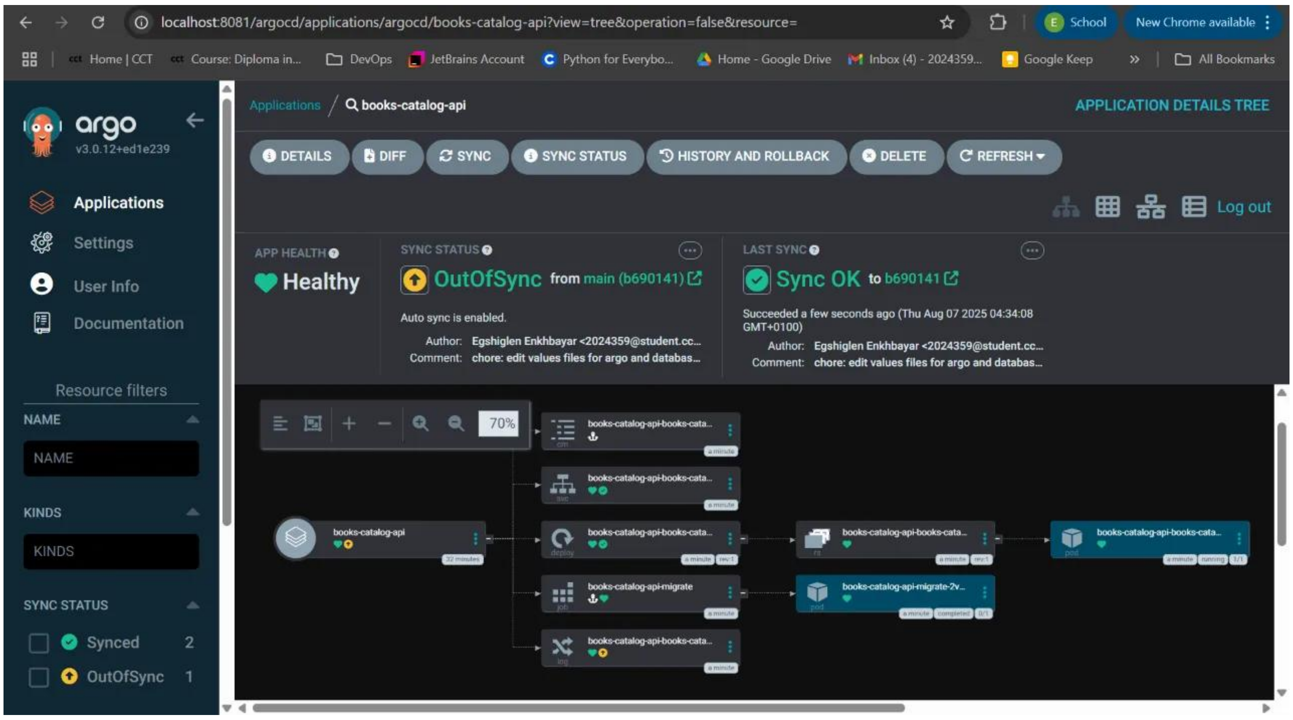
```
GET /api/books/

HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 12,
        "title": "A Thousand Splendid Suns",
        "author": "Khaled Hosseini",
        "isbn": "9781594483851",
        "published_date": "2007-05-22",
        "status": "available",
        "created_at": "2025-08-08T01:23:51.647275Z",
        "updated_at": "2025-08-08T01:23:51.647285Z"
    },
    {
        "id": 1,
        "title": "Call Me by Your Name (Updated)",
        "author": "André Aciman",
        "isbn": "9781250169440",
        "published_date": "2007-01-23",
        "status": "archived",
        "created_at": "2025-08-08T01:12:59.970832Z",
        "updated_at": "2025-08-08T04:02:03.667308Z"
    },
    {
        "id": 16,
        "title": "Don Quixote",
```

13. **Ingress OutOfSync in Argo CD**

    - Missing host field and wrong ingress rule structure caused drift detection.

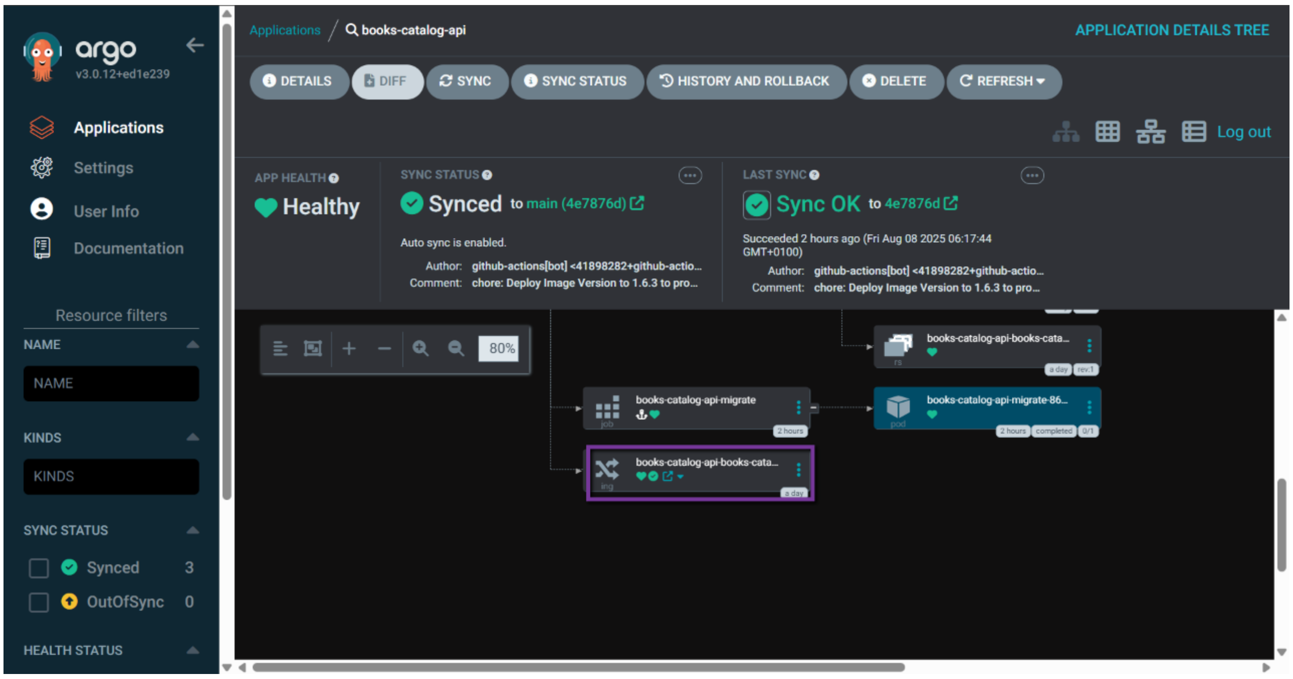


APP HEALTH: Healthy

SYNC STATUS: OutOfSync from main (b690141)
Auto sync is enabled.
Author: Egshiglen Enkhbayar <2024359@student.cc...
Comment: chore: edit values files for argo and databas...

LAST SYNC: Sync OK to b690141
Succeeded a few seconds ago (Thu Aug 07 2025 04:34:08 GMT+0100)
Author: Egshiglen Enkhbayar <2024359@student.cc...
Comment: chore: edit values files for argo and databas...

- Fix: Added proper host and pathType configuration.



APP HEALTH: Healthy

SYNC STATUS: Synced to main (4e7876d)
Auto sync is enabled.
Author: github-actions[bot] <41898282+github-actio...
Comment: chore: Deploy Image Version to 1.6.3 to pro...

LAST SYNC: Sync OK to 4e7876d
Succeeded 2 hours ago (Fri Aug 08 2025 06:17:44 GMT+0100)
Author: github-actions[bot] <41898282+github-actio...
Comment: chore: Deploy Image Version to 1.6.3 to pro...

D) Application & Data

14. **Serializer validation gaps**
    - API accepted future published dates and malformed ISBNs.
    - Fix: Added field validators and tests.
15. **Local vs cluster DB environment mismatch**
    - Missing DB host/user/password in Kubernetes environment.
    - Fix: Added them to Helm values → ConfigMap/Secret → Deployment.

Overall Lesson Learned

This project pushed me harder than any coding assignment I've done before. I didn't just learn Docker, Kubernetes, Helm, Argo CD, and CI/CD — I learned how to stay calm when everything fails at once. I saw how one tiny misconfiguration can take down the whole pipeline, and how fixing it means carefully checking every link in the chain. The biggest lesson was that DevOps is as much about patience, persistence, and problem-solving as it is about technical skills.

Main Challenge

The hardest part was dealing with repeated deployment failures. There were days when I fixed one problem only for two more to appear, and I honestly felt like breaking down because I didn't know where to even start. I even lost my entire project at one point because of a Git command mistake, and had to rebuild it from scratch multiple times. But step by step — reading logs, testing locally, tweaking YAML, re-deploying — I eventually got it all working. That struggle taught me that persistence really is the most important DevOps skill.

7. Conclusion

This was my very first time working on a project that involved Docker, Kubernetes, Helm, Argo CD, and a full CI/CD pipeline. I started with almost no experience, so the biggest achievement for me was just getting everything to build, deploy, and sync automatically. I learned a lot about how all the parts connect, and I now have a much clearer picture of what DevOps actually looks like in real life.

If I had more time and experience, there are things I would like to improve:

1. Monitoring

Why needed: Right now, if something breaks, I only know by manually checking. It would be nice to have something that tells me automatically when the app or database is down.

Why not done now: I didn't know how to set that up yet, and my focus was just getting the app running successfully.

2. HTTPS

Why needed: At the moment the app runs on HTTP, which isn't secure if it was on the internet. HTTPS would make it safer.

Why not done now: I didn't have a domain name or know how to configure certificates yet, so I kept it simple.

3. More Testing

Why needed: My tests check the basics, but I know more tests would help catch problems earlier.

Why not done now: I was still learning how to write tests for Django and didn't want to get stuck there before finishing the main project.

4. Better Secrets Management

Why needed: Right now, secrets like database passwords are in Kubernetes Secrets, but I've read there are safer ways to handle them.

Why not done now: This was my first time dealing with Kubernetes secrets and I just wanted them to work first.

5. Documentation for Others

Why needed: If someone else wanted to run this, they might still get stuck without me explaining it.

Why not done now: I wrote a README for the project, but making step-by-step docs for beginners would take extra time.