INFO 8985 - Monitoring And Logging

Final Project Report

Submitted by

Cibi Sharan Cholarani - 9015927

Deepak Tamizhalagan - 8983627

Nikhil Shankar Chirakkal Sivasankaran - 9026254

Richard Andrey Biscazzi - 8903530

Zafar - 9027671

Table of Contents

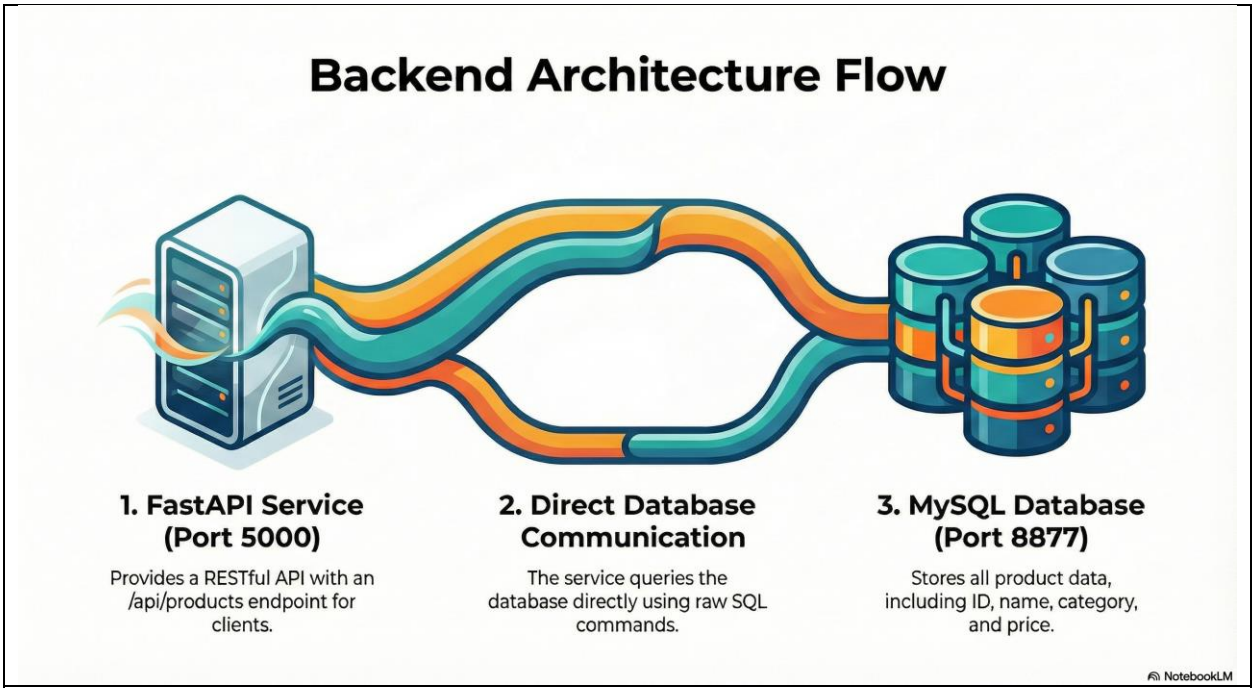
MySQL & API Integration 2
Architecture 2
Components 3
1. MySQL Database 3
2. Python FastAPI 3
3. Products Microfrontend (Angular) 3
4. Shell Microfrontend (Angular) 4
Steps to Run 4
Key Files 5
Data Flow 6
Logging with FastAPI, Prometheus, Grafana, Loki & Promtail (Deepak) 6
Frontend Testing with Jasmine and Karma 9

MySQL & API Integration

Python FastAPI service that replaces hardcoded products with MySQL database.

Architecture

```
Shell Microfrontend (4200)
    └─> Products Microfrontend (4201)
            └─> Python API (5000)
                    └─> MySQL (8877)
```



Backend Architecture

Components

1. MySQL Database

- **Port:** 8877
- **Database:** products_db
- **Table:** products (id, name, category, price, stock)

- **Runs in:** Docker container

2. Python FastAPI

- **Port:** 5000
- **Endpoints:**
  - `GET /api/products` - Returns all products from MySQL
  - `GET /health` - Health check
  - `GET /metrics` - Prometheus metrics
  - `GET /docs` - API documentation
- **Tech:** FastAPI + pymysql (raw SQL, no ORM)
- **Runs in:** Docker container

3. Products Microfrontend (Angular)

- **Port:** 4201
- **What changed:**
  - Created `ProductService` with HttpClient
  - Calls `http://localhost:5000/api/products`
  - Replaced hardcoded HTML table with `*ngFor`
- **Fix applied:** Removed `providedIn: 'root'` to avoid circular dependency in Module Federation

4. Shell Microfrontend (Angular)

- **Port:** 4200
- **Role:** Host app that loads Products remote
- **No changes needed**

Steps to Run

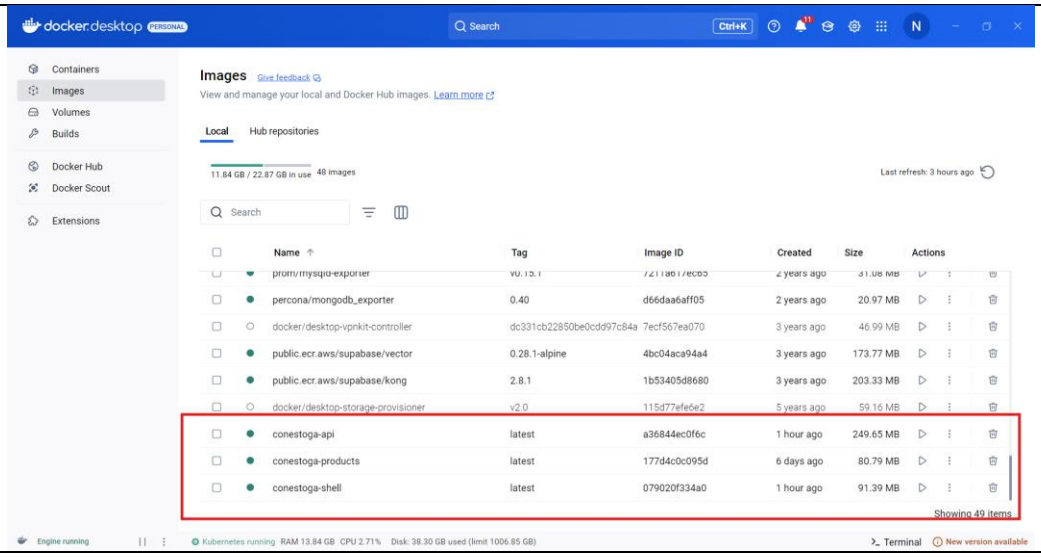
```
# 1. Start MySQL + API
docker-compose up -d

# 2. Seed database
docker exec -it products-api python seed.py

# 3. Start Products microfrontend
cd products/products
npm start

# 4. Start Shell microfrontend
cd shell/shell
npm start
```

5. Test

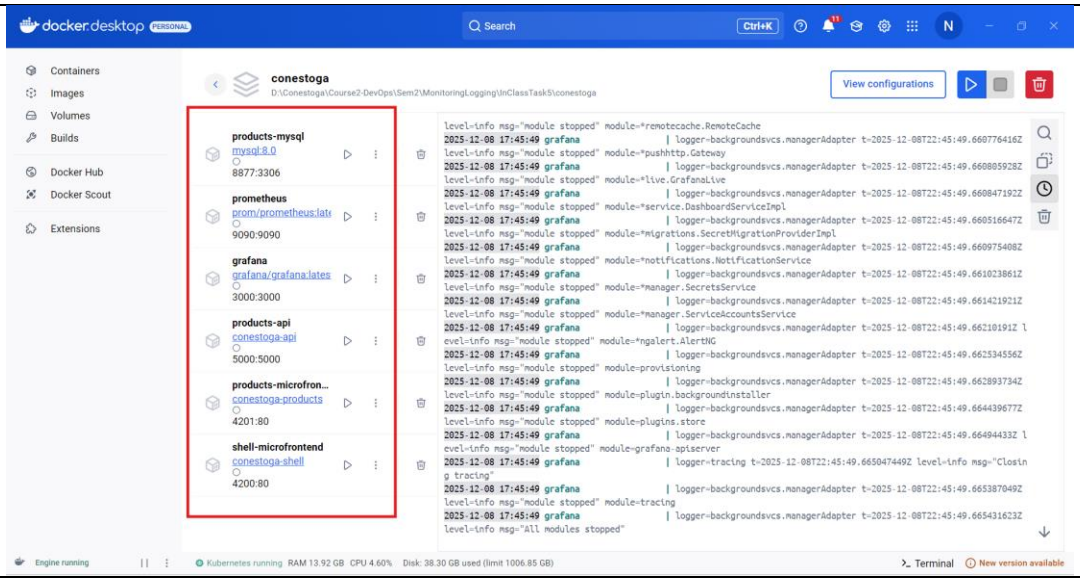
Open http://localhost:4200

Click "Go to Products"

Products load from MySQL!



Docker Images



Container Running

Key Files

```
api/
├── main.py              # FastAPI with raw SQL queries
├── seed.py              # Database seeder
```

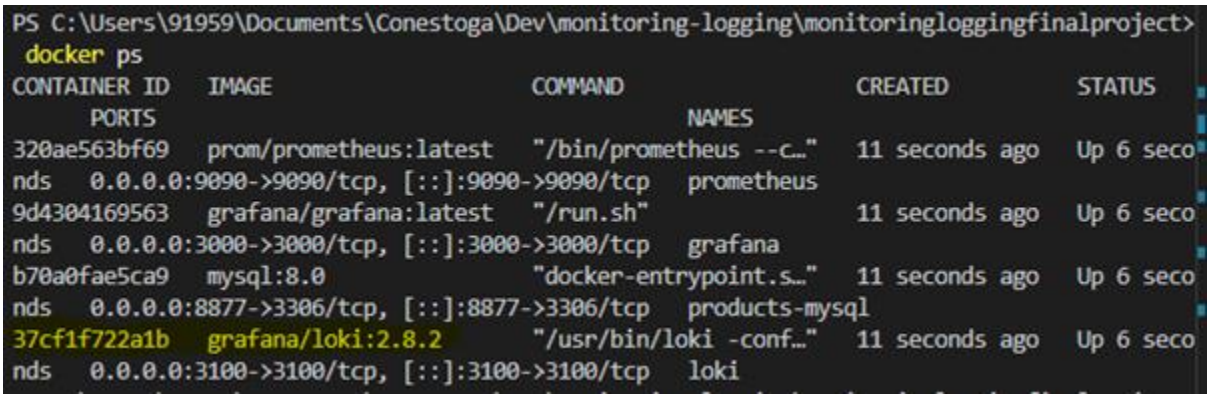
```
├── requirements.txt     # 4 dependencies only
└── Dockerfile

docker-compose.yml       # MySQL + API containers + Microfrontend

products/products/src/app/products/
├── product.model.ts     # Product interface
├── product.service.ts   # API HTTP calls
├── products.component.ts   # Uses ProductService
└── products.module.ts   # Provides ProductService
```

Data Flow

1. User opens Shell (http://localhost:4200)
2. Clicks "Go to Products"
3. Shell loads Products remote via Module Federation
4. ProductsComponent calls ProductService.getProducts()
5. ProductService makes HTTP call to http://localhost:5000/api/products
6. FastAPI queries MySQL: `SELECT * FROM products`
7. Returns JSON to Angular
8. Products display in table with *ngFor

Logging with FastAPI, Prometheus, Grafana, Loki & Promtail (Deepak)

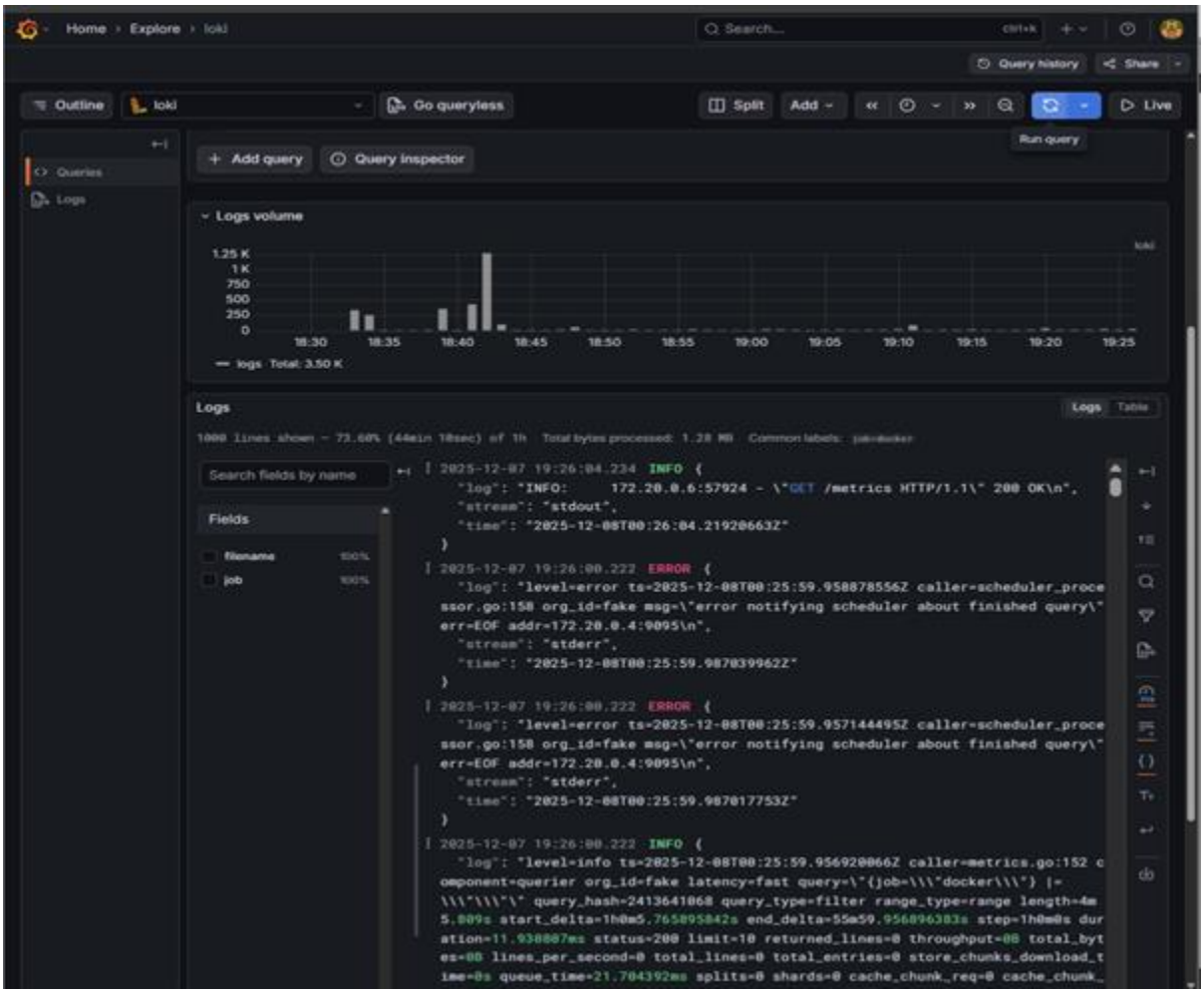
In this task, my responsibility was to integrate a complete logging and monitoring solution for our FastAPI application. The purpose of this implementation was to make our application observable during run-time and help us analyse performance, behaviour, errors, and operational events. For this, I used Prometheus and Grafana for monitoring, and Loki + Promtail for centralized logging. My work continues from Zafar's monitoring setup and extends it into full logging visibility.



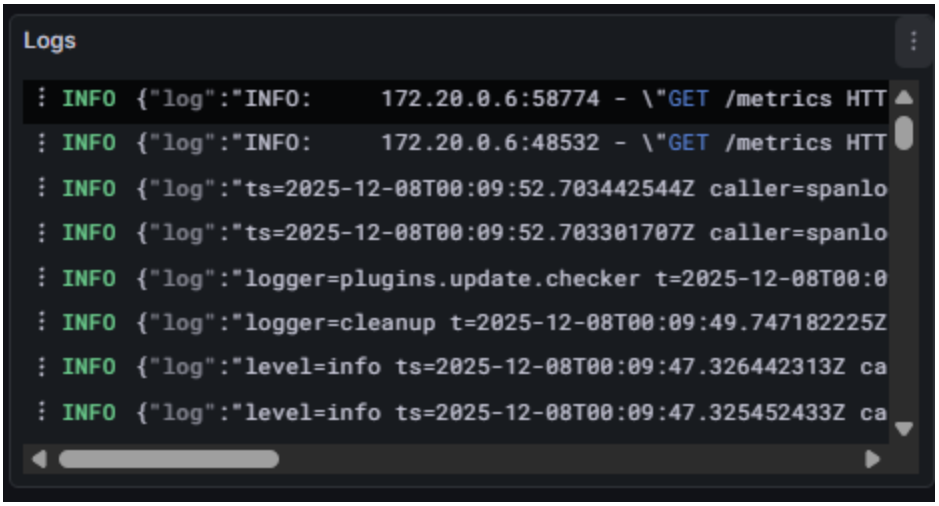
The first step was enabling Prometheus metrics and exposing them from FastAPI. Our application automatically exposes a /metrics endpoint which Prometheus scrapes at regular intervals, and Grafana visualizes these metrics on dashboards. I also tested this by calling our API endpoints multiple times and confirmed that metrics were received in Prometheus and displayed in Grafana.

Secondly, I configured Loki and Promtail as additional services inside Docker Compose. Promtail collects logs from all application containers, and sends them into Loki, which is connected to Grafana as a data source. After running the system, I checked Grafana → Explore and confirmed that the application logs were successfully visible. I also created a

test endpoint /force-error to intentionally generate error logs and verified that these errors were captured inside Loki.



Finally, I enhanced FastAPI to produce structured JSON logs, including useful contextual information such as HTTP method, endpoint, client address, and status. This helps us filter, search, and analyse different types of events. The logs are formatted in a clean JSON structure, which makes them suitable for production observability. Overall, this improves debugging, error tracking, and gives full transparency into application behaviour.



This completes my part of the project successfully. The system now supports end-to-end monitoring and centralized logging, which can be further extended using alerting rules, anomaly detection, or distributed tracing in future scope.

Frontend Testing with Jasmine and Karma

In our project, we implemented automated testing using Jasmine and Karma. Jasmine was used to define unit and integration test cases using describe, it, and expect functions. Karma acted as the test runner that launched Chrome and executed our tests in a real browser environment.

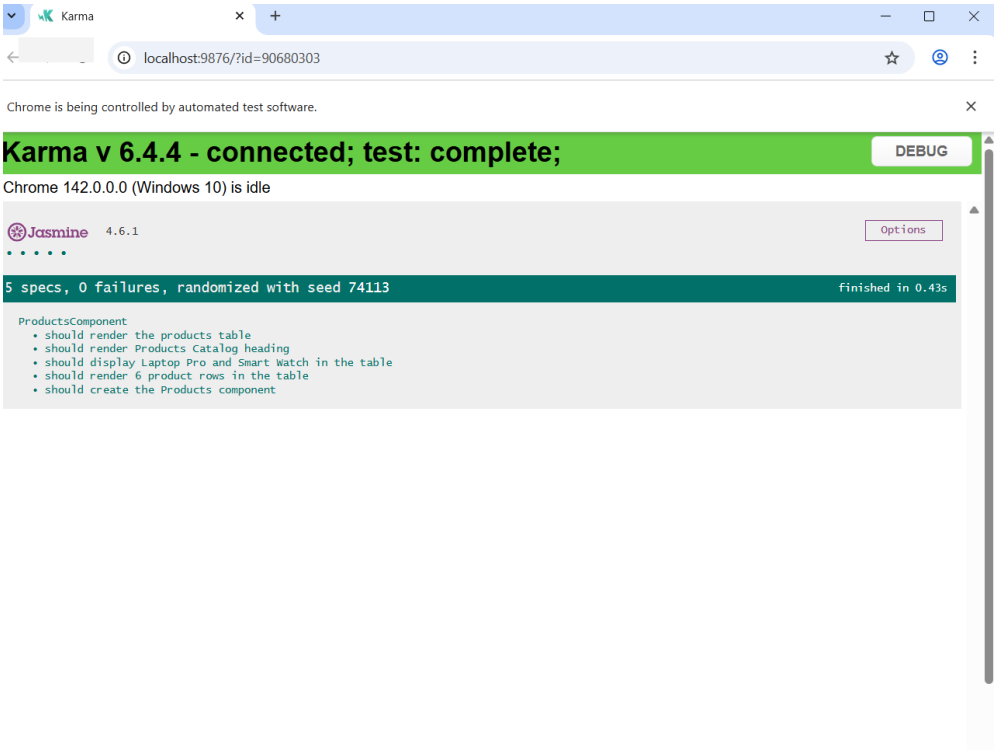
We wrote unit tests for the Shell app to verify component creation, UI headings, and navigation routing. We also wrote a unit test for the Products microfrontend to verify that business data such as Laptop Pro and Smart Watch are correctly displayed.

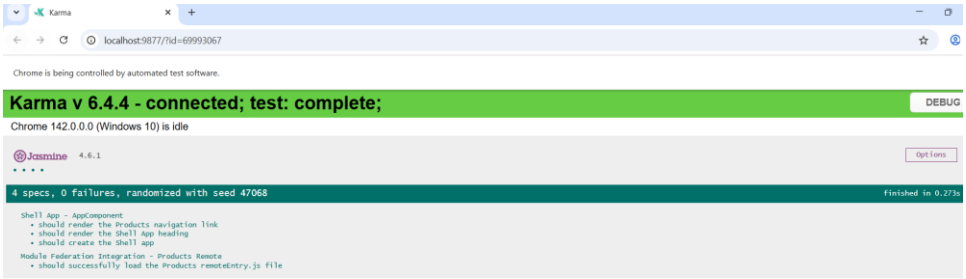
Finally, we implemented a Module Federation integration test that dynamically fetched the Products remoteEntry.js file from localhost, validating real runtime microfrontend communication. All tests passed successfully, confirming that our microfrontend architecture is stable and production-ready.

Unit tests for Shell Component:
- To check if the Shell Application component is successfully created by Angular.
- Should render the Shell app contents (here, its the heading we have given)
- Should render the products navigation link

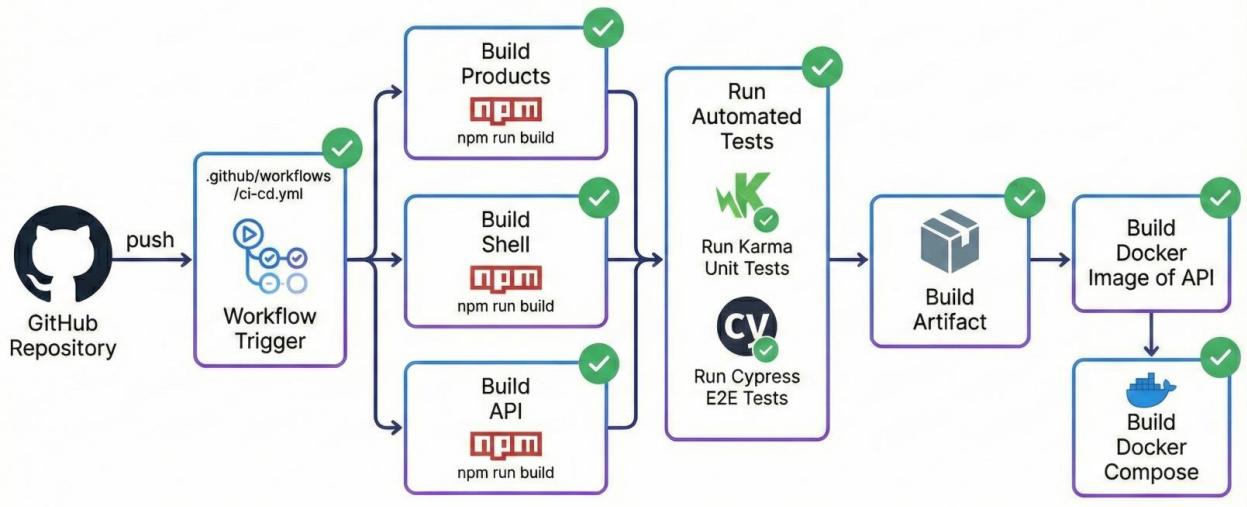
Unit tests for Products Component:
- To ensure that the Products table renders correct product data.
- The UI table properly renders the table data.

Module federation Integration tests:
- Ensure that shell application can fetch the products microfrontend over the network using Webpack module federation.
- It sends a network request to ` http://localhost:4201/remoteEntry.js` and confirms that the file exists and the response is successful.





CI CD Pipeline

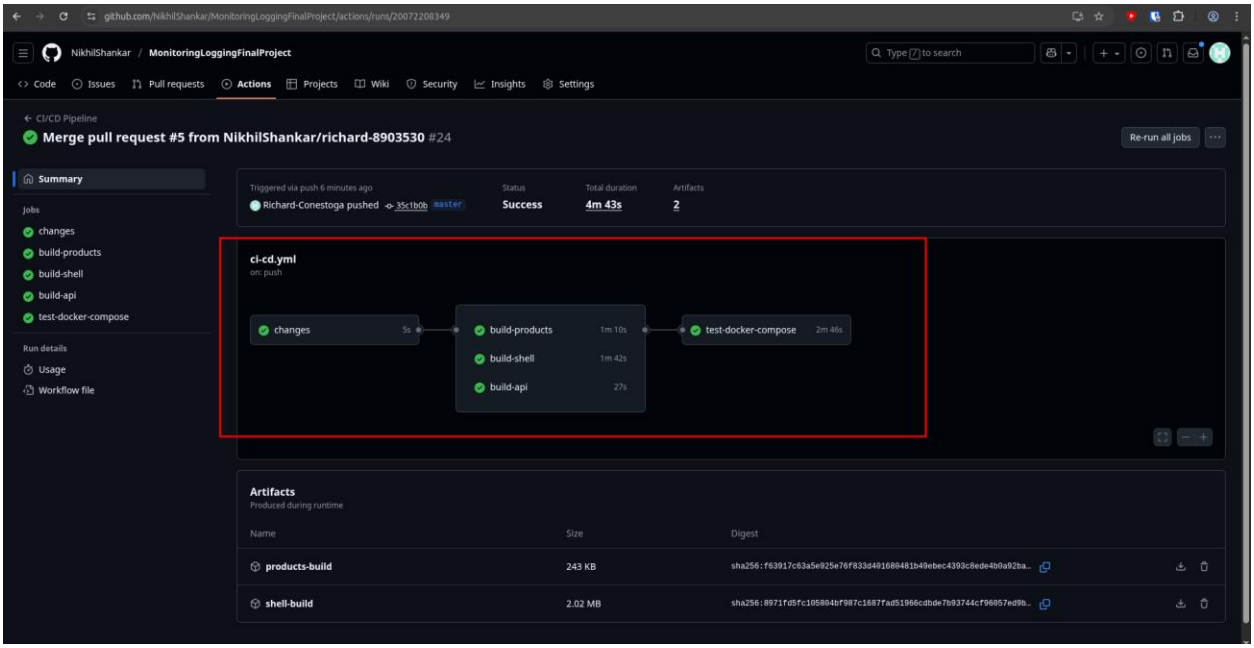


CI/CD Pipeline with GitHub Actions

- Automated workflow to build **Products** and **Shell** Angular applications
- Configured parallel job execution for faster build times
- Set up automated testing on push and pull request events
- Added Docker image building for Python API service

Key Outcomes

- Automated build verification run on code change
- Early detection of build failures and integration issues
- Consistent build environment across development and CI
- Ready for Docker image publishing to container registry
- Foundation laid for automated deployment to production


github.com/NikhilShankar/MonitoringLoggingFinalProject/actions/runs/20072208349
NikhilShankar / MonitoringLoggingFinalProject
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
CI/CD Pipeline
Merge pull request #5 from NikhilShankar/richard-8903530 #24
Re-run all jobs
Summary
Jobs
changes
build-products
build-shell
build-api
test-docker-compose
Run details
Usage
Workflow file
Triggered via push 6 minutes ago
Richard-Conestoga pushed 35c1b0b master
Status
Success
Total duration
4m 43s
Artifacts
2
ci-cd.yml
on: push
changes 5s
build-products 1m 10s
build-shell 1m 42s
build-api 27s
test-docker-compose 2m 46s
Artifacts
Produced during runtime
Name
Size
Digest
products-build
243 KB
sha256:f63917c63a5e925e76f833d401680481b49ebec4393c8ede4b0a92ba...
shell-build
2.02 MB
sha256:8971fd5fc105804bf987c1687fad51966cdbde7b93744cf96057ed9b...