HAROKOPIO UNIVERSITY
DEPARTMENT OF INFORMATICS AND TELEMATICS

Cloud Platforms Project

Laveraging Cloud Platforms for IoT Applications: A MicroK8s Implementation.

Katsimpras Drosos - ais25123

Harokopio University of Athens – Department of Informatics and Telematics
MSc in Advances in Computer Science and Informatics Systems
Professor: Kousiouris Georgios | Tsadimas Anargyros
Academic Year: 2025-2026

Table of Contents

Problem Definition & Goal

Defining the Challenge: From Monolith to Cloud-Native

- **The Challenge:** Transitioning a legacy monolithic Inventory ERP to a modern, scalable environment.
- **Current Limitations:**
  - **Scalability Issues:** Hard to scale individual components without affecting the whole system.
  - **Single Point of Failure:** Tightly coupled services increase the risk of complete system downtime.
  - **Deployment Complexity:** Environment dependencies make migration to production error-prone.
- **The Goal:** To design a distributed, Cloud-Native Smart Warehouse system using Microservices and IoT integration.

Proposed Architecture & Methodology

Strategic Implementation: From Containerization to Hybrid Orchestration

- Architectural Pillars:
  - **Containerization:** Packaging services into Docker images for environment-agnostic execution.
  - **Full Orchestration:** Transitioning from simple runtime to MicroK8s for automated lifecycle management.
- The Hybrid Deployment Strategy (Consistent in both phases):
  - **Data Persistence (Public Cloud):** Leveraging MongoDB Atlas (DBaaS) to ensure data availability and durability outside the local cluster.
  - **Application & Logic (Private Cloud):** Running Backend, Frontend, and services (rabbitmq, minIO, node-RED, Thingsboard) within the local MicroK8s environment.
- **Automation:** Using YAML manifests to eliminate manual configuration.

Technology Stack

Infrastructure and Technologies: A Unified Cloud-Native Stack

- **Integration Maturity:** Selection of production-grade tools that support a full application lifecycle.
- **Hybrid Interoperability:** Seamless communication between on-premise MicroK8s resources and public DBaaS (MongoDB Atlas).
- **Event-Driven Foundation:** Leveraging RabbitMQ and Node-RED for real-time asynchronous data flows.

Service	Technology	Role in Infrastructure
Orchestration	MicroK8s	Container management and orchestration
Containerization	Docker & Hub	Image registry for distribution
Messaging	RabbitMQ	Asynchronous communication (Event Bus)
Database	MongoDB Atlas	Managed NoSQL database service
Storage	MinIO	Object Storage for files and images
IoT Logic	Node-RED	Flow-based data processing
Visualization	Thingsboard	IoT Dashboards & Real-time monitoring

Table 1: Infrastructure Technologies and Services

Hybrid Cloud Deployment (Phase A)

Initial Implementation: Integrating Local Containers with Managed DBaaS

- **Custom Docker Images:**: We created **Dockerfiles** for the Frontend and Backend to build our own custom images.
- **Docker-Compose Setup:**: We used a ***docker-compose*** file to run the Frontend and Backend containers together easily.
- **Kubernetes Infrastructure:** We created YAML manifests to set up RabbitMQ and MinIO as services inside the**MicroK8s cluster**.
- **Hybrid Database Connection:**: The Backend connects directly to MongoDB Atlas in the Public Cloud using a simple connection string in the **.env** file.
- **Proof of Concept**: System logs confirm that all parts (Docker, MicroK8s, and Atlas) communicate correctly.

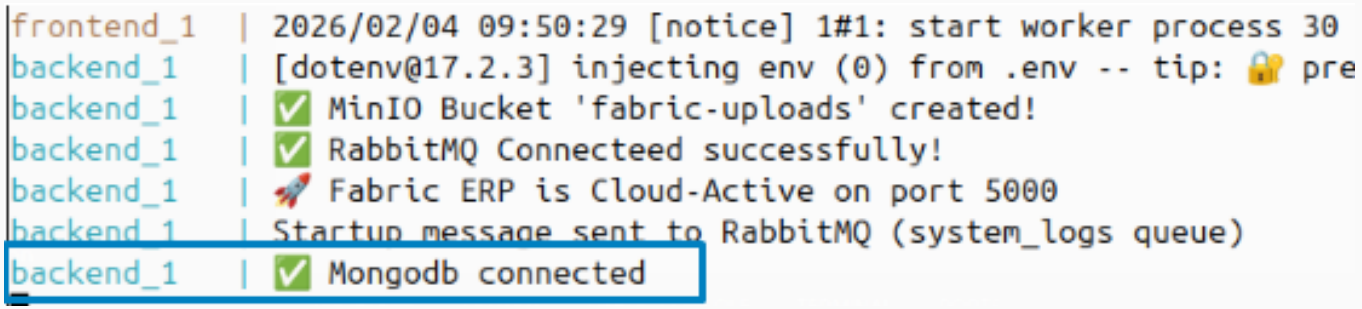

Figure 1: Backend logs confirming connection to MinIO, RabbitMQ, and MongoDB Atlas.

Migration Strategy: Docker Compose to MicroK8s

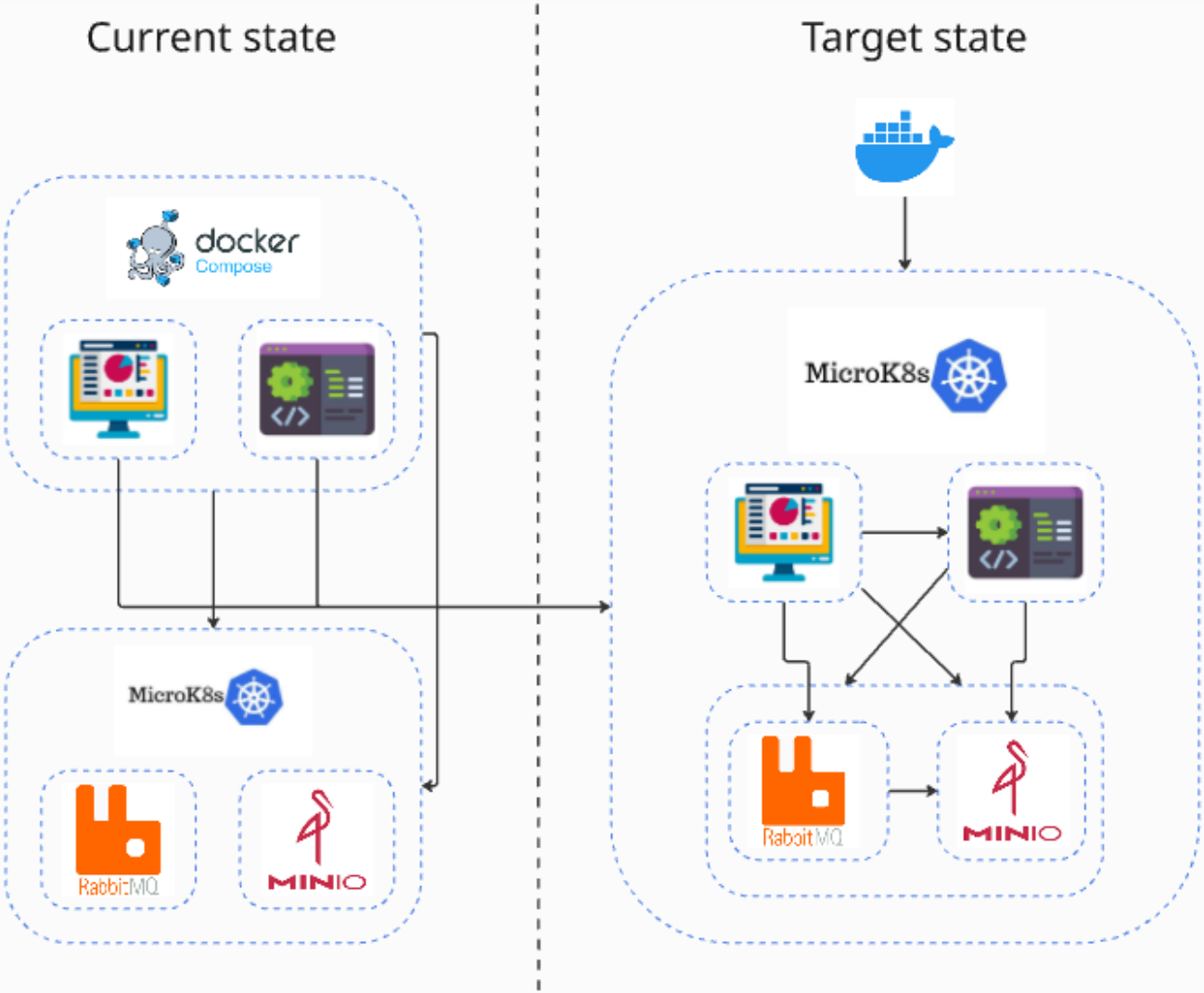

Figure 2: Current vs Target state

Full Orchestration & Automation (Phase B)

Automated Deployment

- **Image Preparation**: Created final *Docker* images for Frontend and Backend and pushed them to *Docker Hub*.
- **Kubernetes Configuration**: Developed *YAML* manifests for every service, including the application and infrastructure (*MinIO*/*RabbitMQ*).
- **IoT Integration**: Added *Node-RED* and *ThingsBoard* into the cluster using dedicated *YAML* files to complete the ecosystem.
- **Final Deployment**: Executed the *deploy_fabric.sh* script to pull images and start all services in the correct order.

Deploy result

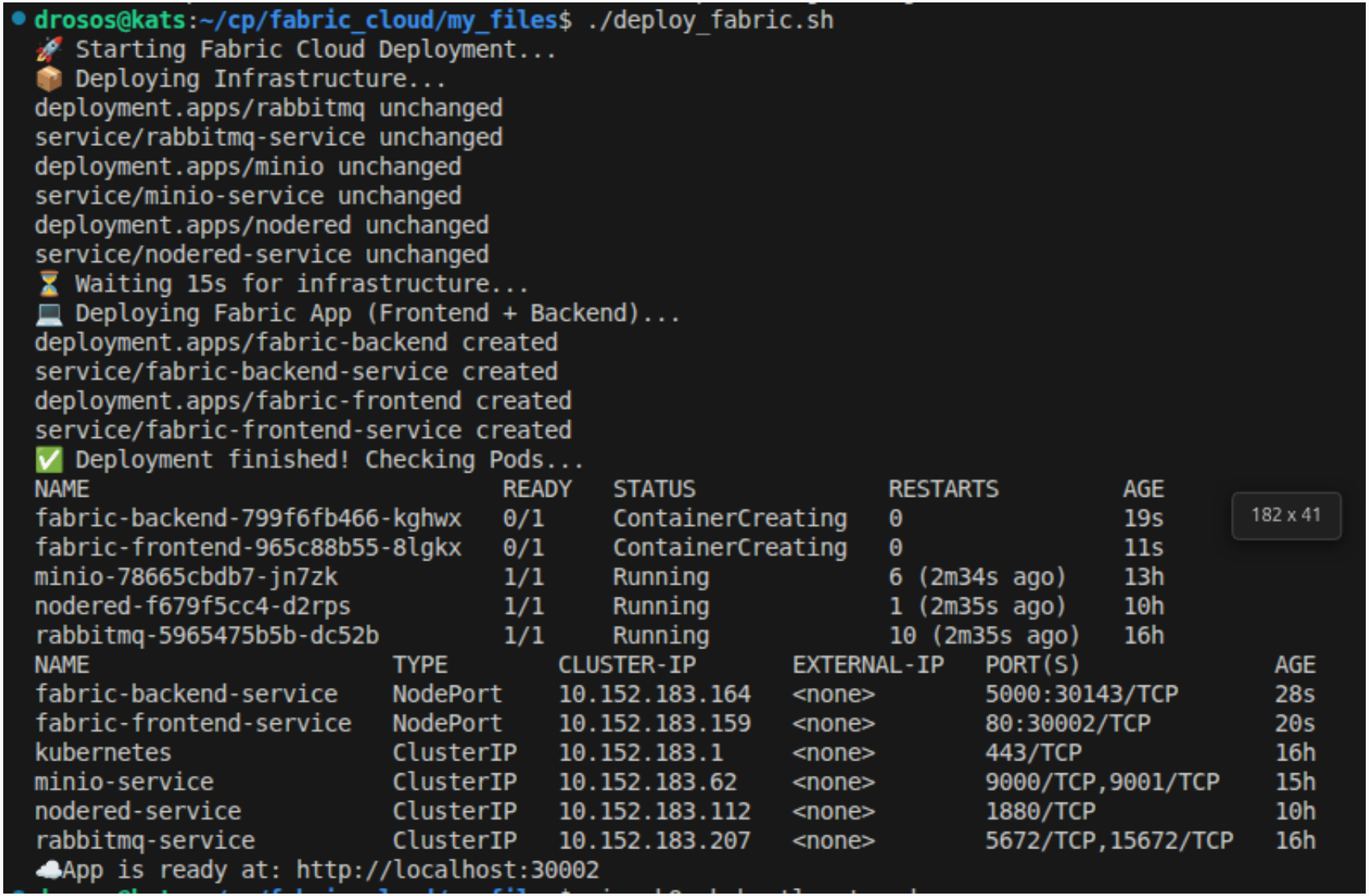

Figure 3: Final Deployment using the script

Operational Status: All Services Running in MicroK8s

```
drosos@kats:~/cp/fabric_cloud/my_files$ microk8s kubectl get pods
fabric-backend-6b8f8f4c68-b26vk   1/1     Running             1 (47m ago)    8h
fabric-frontend-965c88b55-8lgkx   1/1     Running             1 (47m ago)    9h
minio-78665cbdb7-jn7zk            1/1     Running             7 (47m ago)    23h
nodered-f679f5cc4-d2rps           1/1     Running             2 (47m ago)    20h
rabbitmq-5965475b5b-dc52b         1/1     Running             11 (47m ago)   25h
thingsboard-74689b6c5d-txskc      0/1     ContainerCreating   0              112s
drosos@kats:~/cp/fabric_cloud/my_files$ microk8s kubectl get pods
NAME                              READY   STATUS    RESTARTS       AGE
fabric-backend-6b8f8f4c68-b26vk   1/1     Running   1 (49m ago)    8h
fabric-frontend-965c88b55-8lgkx   1/1     Running   1 (49m ago)    9h
minio-78665cbdb7-jn7zk            1/1     Running   7 (49m ago)    23h
nodered-f679f5cc4-d2rps           1/1     Running   2 (49m ago)    20h
rabbitmq-5965475b5b-dc52b         1/1     Running   11 (49m ago)   25h
thingsboard-74689b6c5d-txskc      1/1     Running   0              3m35s
```

Figure 4: microk8s kubectl get pods

IoT Integration & Case Study

IoT Integration

- **The Trigger**: Adding a new product in the *ERP* app starts a real-time data sequence.
- **Media Storage**: Product images are automatically uploaded to the *MinIO* bucket.
- **Event Messaging**: *RabbitMQ* alerts the system that a new entry is ready for processing.
- **Logic Engine**: *Node-RED* consumes the message and simulates warehouse conditions (Temp/Humidity) using a custom script.
- **Live Dashboard**: All data is pushed to *ThingsBoard* for real-time monitoring and history tracking.

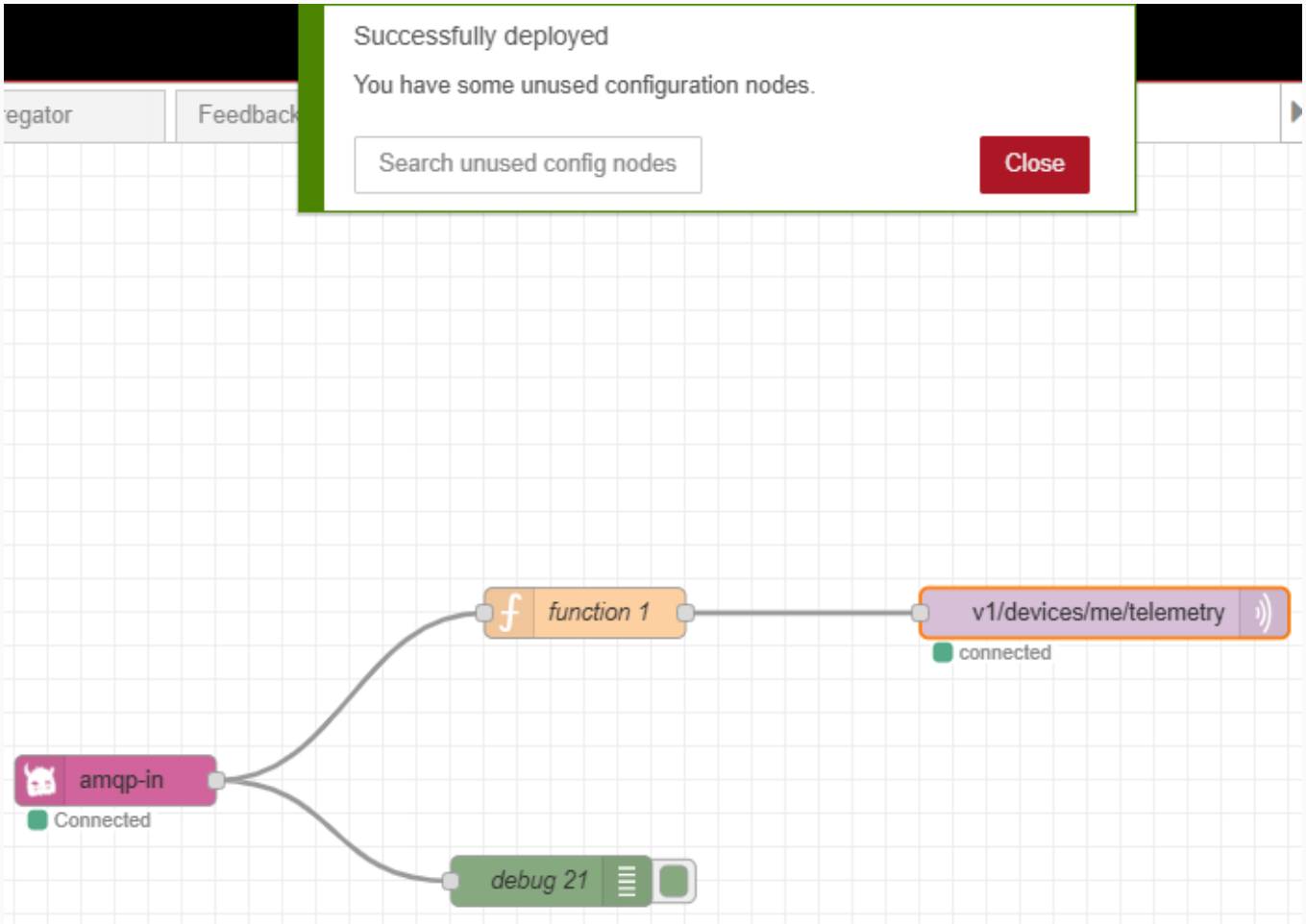

Figure 5: Node-RED flow

Thingsboard telemetry

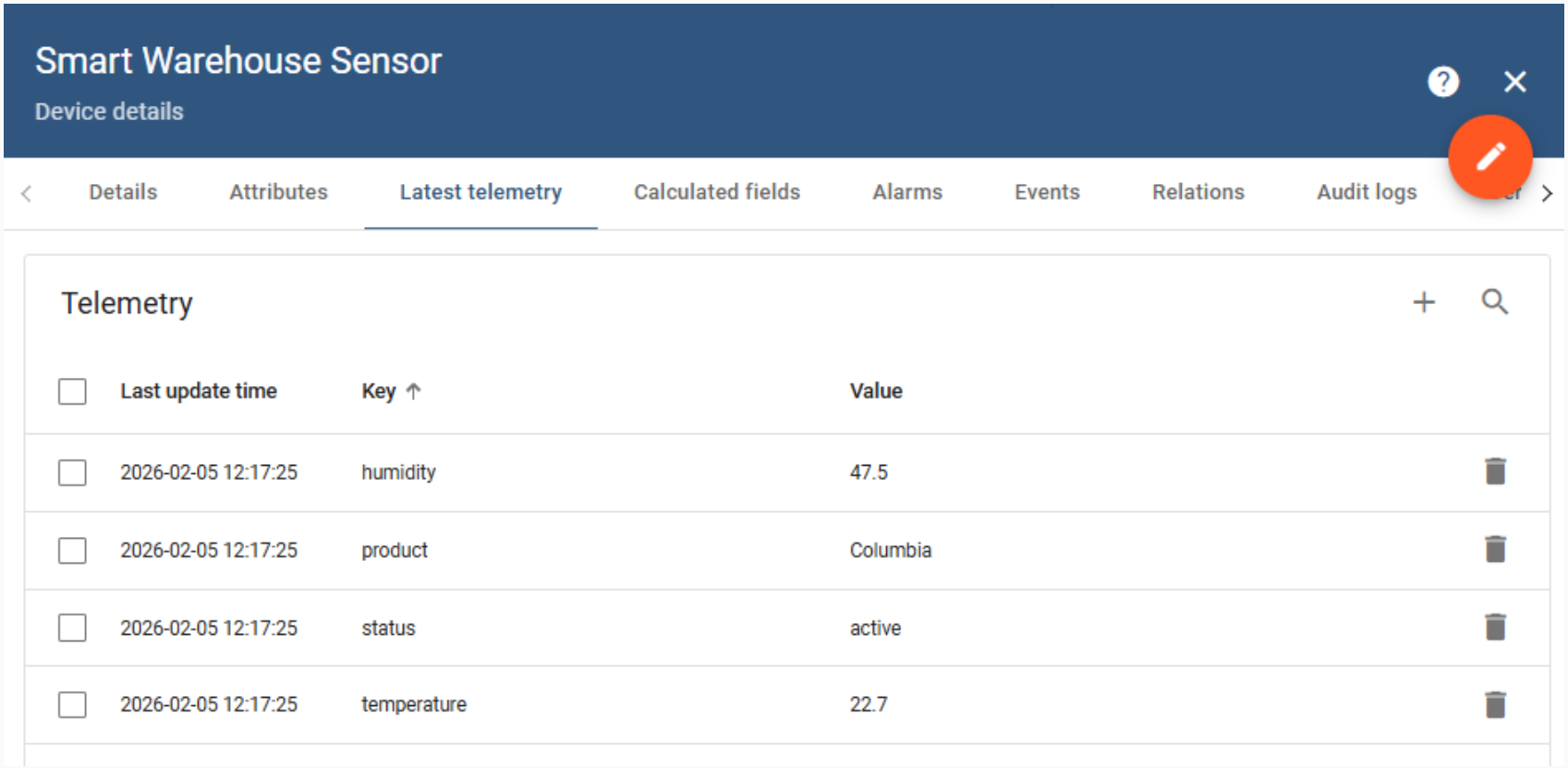

Figure 6: Thingsboard telemetry

Thingsboard Dashboard

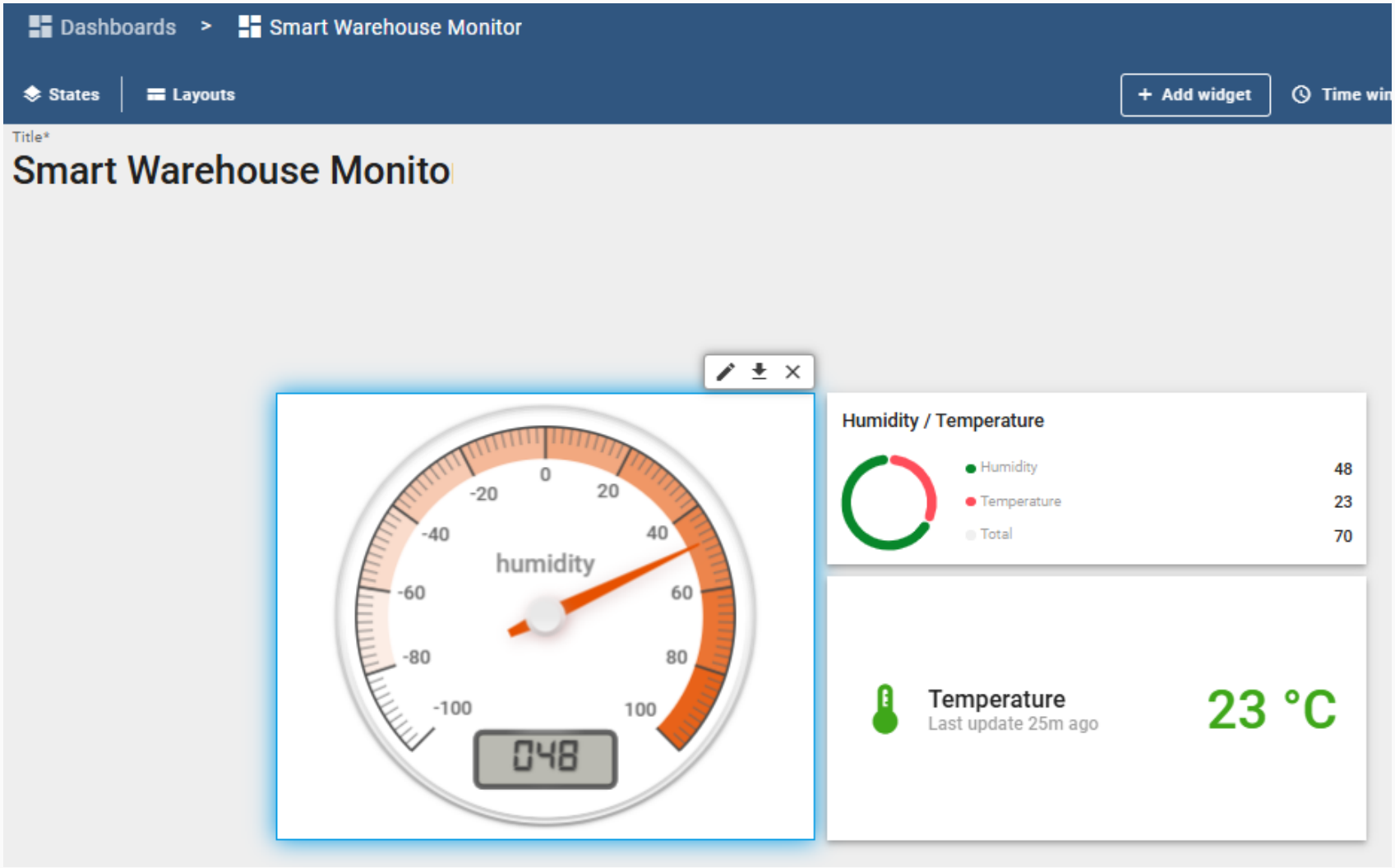

Figure 7: Thingsboard Dashboard

Data Flow diagram

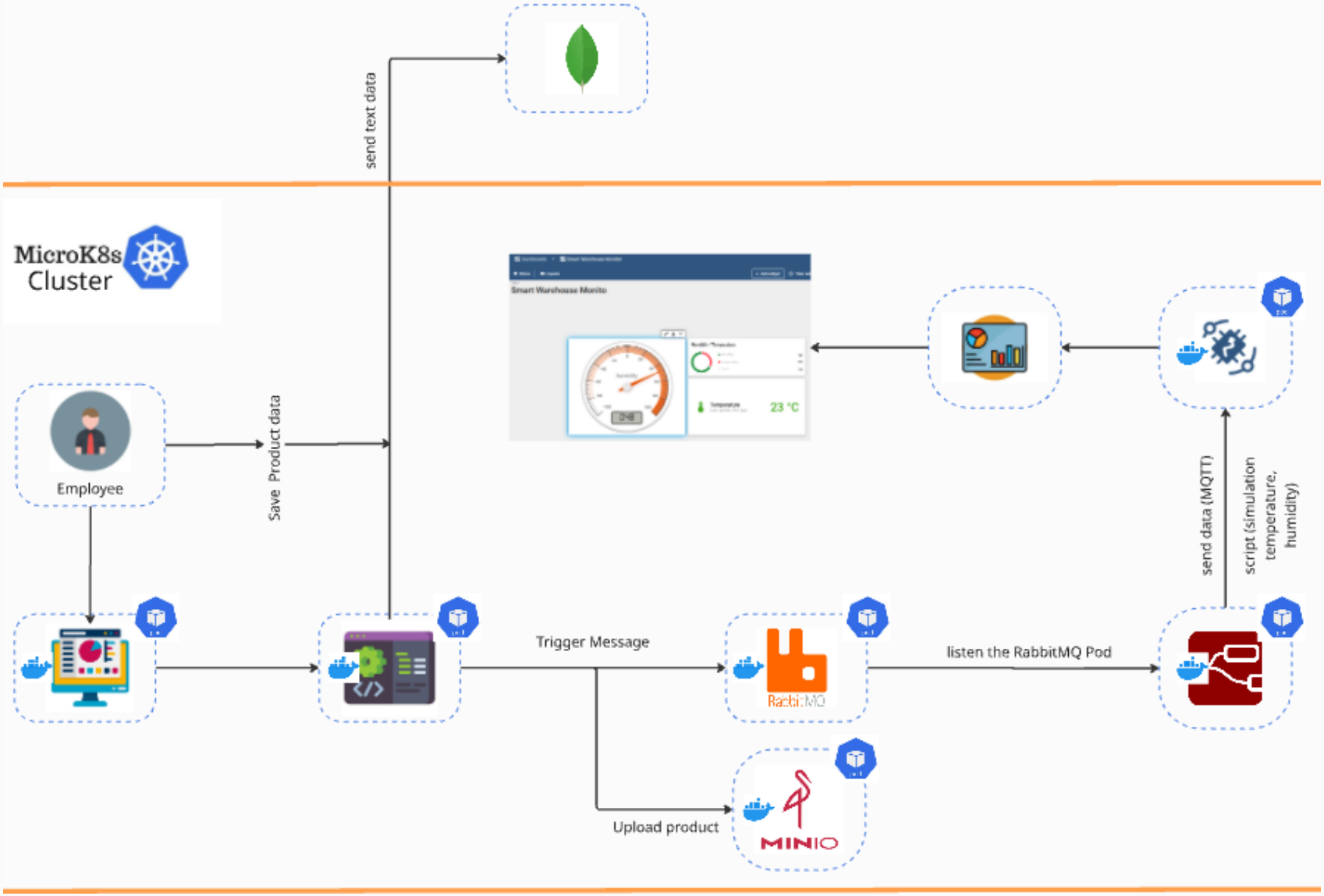

Figure 8: Data flow diagram for ”Add product” scenario

Conclusions & Future Work

Conclusions & Future Work

- **Automation Success**: Fully automated the deployment of 7+ services using *Bash* scripts and *Kubernetes YAML* files.
- **Hybrid Efficiency**: Successfully connected local *MicroK8s* services with the *MongoDB Atlas* public cloud.
- **Identity Management (Future)**: Integrating *Keycloak* to provide secure, centralized authentication (*SSO*) for all services.
- **Data Persistence (Future)**: Using Persistent Volumes (*PV*) to ensure data is safe if a *Pod* restarts.
- **Traffic Control (Future)**: Implementing an Ingress Controller for professional *URL* management and improved security.

GitHub repository

Get the source of the whole project from

github.com/drososkats/fabric_cloud.git

The presentation *itself* is licensed under Katsimpras Drosos (ais25123)

ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ
HAROKOPIO UNIVERSITY

Thank you! / Questions? ☺