

# Cloud Platforms Project

Leveraging 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

1. Problem Definition & Goal
2. Proposed Architecture & Methodology
3. Technology Stack
4. Hybrid Cloud Deployment (Phase A)
5. Full Orchestration & Automation (Phase B)
6. IoT Integration & Case Study
7. Conclusions & Future Work

## Problem Definition & Goal

---

- **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

---

- 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)

---

- **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.

```
frontend_1 | 2026/02/04 09:50:29 [notice] 1#1: start worker process 30
backend_1  | [dotenv@17.2.3] injecting env (0) from .env -- tip: 🔒 pre
backend_1  | ✓ MinIO Bucket 'fabric-uploads' created!
backend_1  | ✓ RabbitMQ Connected successfully!
backend_1  | 🚀 Fabric ERP is Cloud-Active on port 5000
backend_1  | Startup message sent to RabbitMQ (system_logs queue)
backend_1  | ✓ Mongodbs connected
```

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

# MIGRATION STRATEGY: DOCKER COMPOSE TO MICROK8S

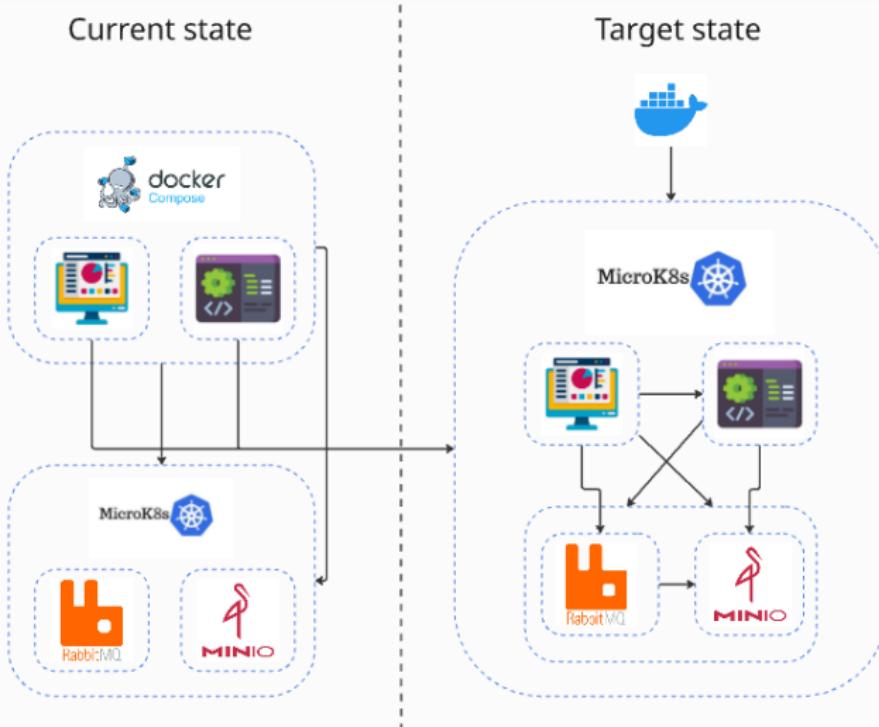


Figure 2: Current vs Target state

## Full Orchestration & Automation (Phase B)

---

- **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

```
● drosos@kats:~/cp/fabric_cloud/my_files$ ./deploy_fabric.sh
🚀 Starting Fabric Cloud Deployment...
📦 Deploying Infrastructure...
deployment.apps/rabbitmq unchanged
service/rabbitmq-service unchanged
deployment.apps/minio unchanged
service/minio-service unchanged
deployment.apps/nodered unchanged
service/nodered-service unchanged
⏳ Waiting 15s for infrastructure...
💻 Deploying Fabric App (Frontend + Backend)...
deployment.apps/fabric-backend created
service/fabric-backend-service created
deployment.apps/fabric-frontend created
service/fabric-frontend-service created
✅ Deployment finished! Checking Pods...
NAME                      READY   STATUS        RESTARTS   AGE
fabric-backend-799f6fb466-kghwx  0/1    ContainerCreating   0          19s   182 x 41
fabric-frontend-965c88b55-8lgkx  0/1    ContainerCreating   0          11s
minio-78665cbdb7-jn7zk         1/1    Running        6 (2m34s ago)  13h
nodered-f679f5cc4-d2rps       1/1    Running        1 (2m35s ago)  10h
rabbitmq-5965475b5b-dc52b     1/1    Running        10 (2m35s ago) 16h
NAME                     TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)           AGE
fabric-backend-service   NodePort    10.152.183.164  <none>        5000:30143/TCP  28s
fabric-frontend-service NodePort    10.152.183.159  <none>        80:30002/TCP   20s
kubernetes              ClusterIP   10.152.183.1    <none>        443/TCP         16h
minio-service            ClusterIP   10.152.183.62   <none>        9000/TCP,9001/TCP 15h
nodered-service          ClusterIP   10.152.183.112  <none>        1880/TCP        10h
rabbitmq-service         ClusterIP   10.152.183.207  <none>        5672/TCP,15672/TCP 16h
☁️App is ready at: http://localhost:30002
```

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

---

- **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.

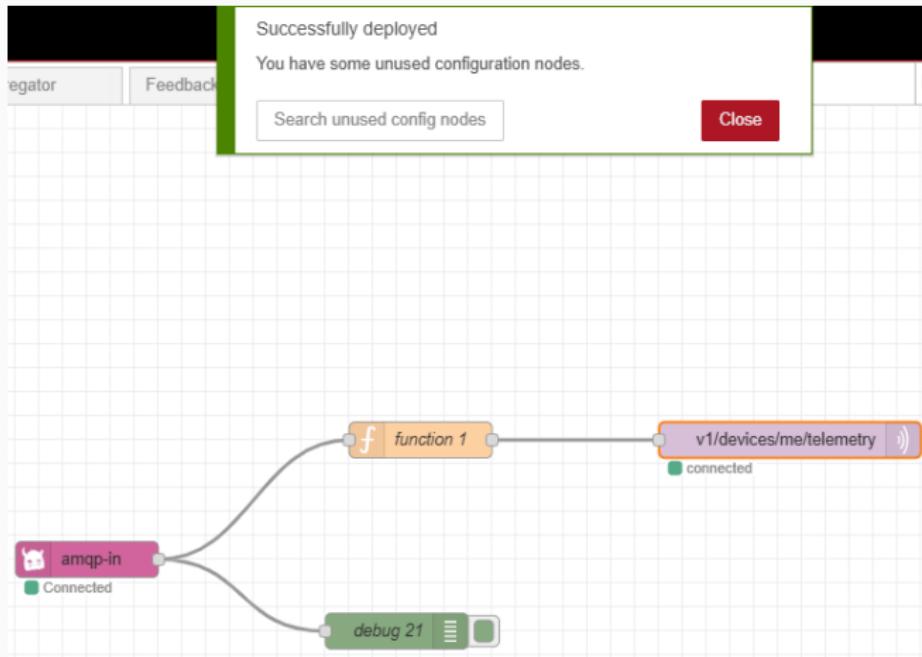


Figure 5: Node-RED flow

# Thingsboard telemetry

The screenshot shows the Thingsboard interface for a 'Smart Warehouse Sensor'. The top navigation bar includes 'Device details' and a red circle with a white pen icon, indicating an edit mode. Below the navigation is a table titled 'Telemetry' with columns for 'Last update time', 'Key ↑', and 'Value'. The table lists five data points: humidity (47.5), product (Columbia), status (active), and temperature (22.7). Each row has a checkbox in the first column and a trash can icon in the last column.

Last update time	Key ↑	Value
2026-02-05 12:17:25	humidity	47.5
2026-02-05 12:17:25	product	Columbia
2026-02-05 12:17:25	status	active
2026-02-05 12:17:25	temperature	22.7

Figure 6: Thingsboard telemetry

# Thingsboard Dashboard

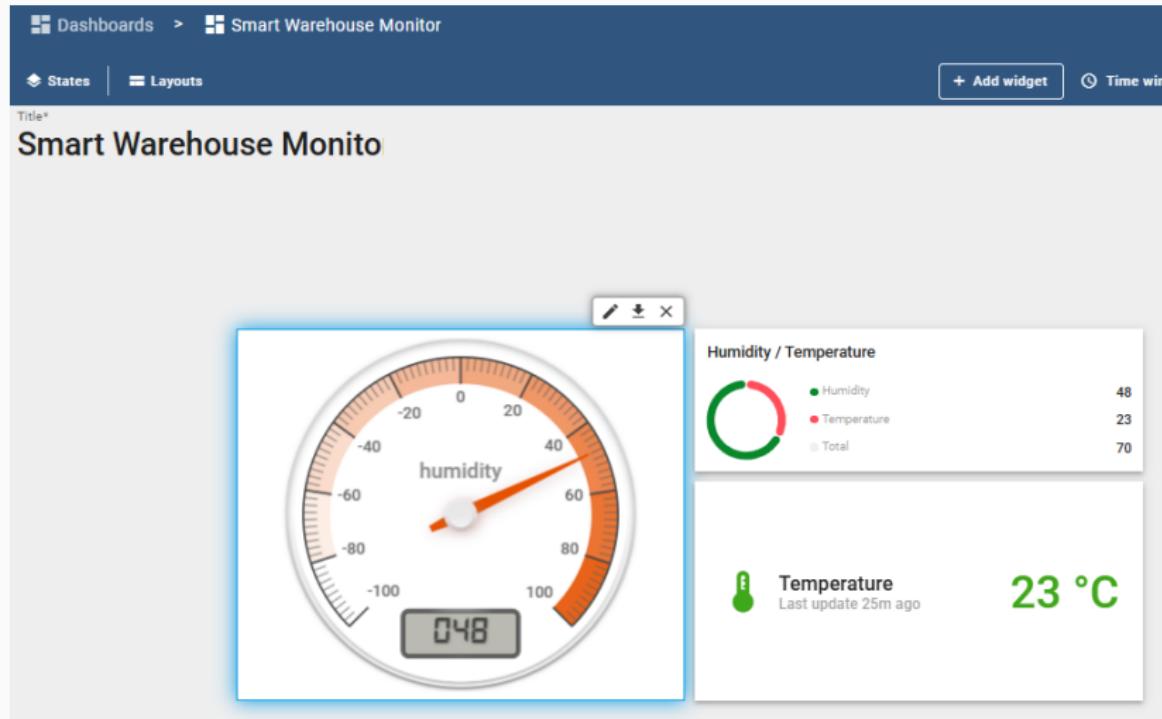


Figure 7: Thingsboard Dashboard

# Data Flow diagram

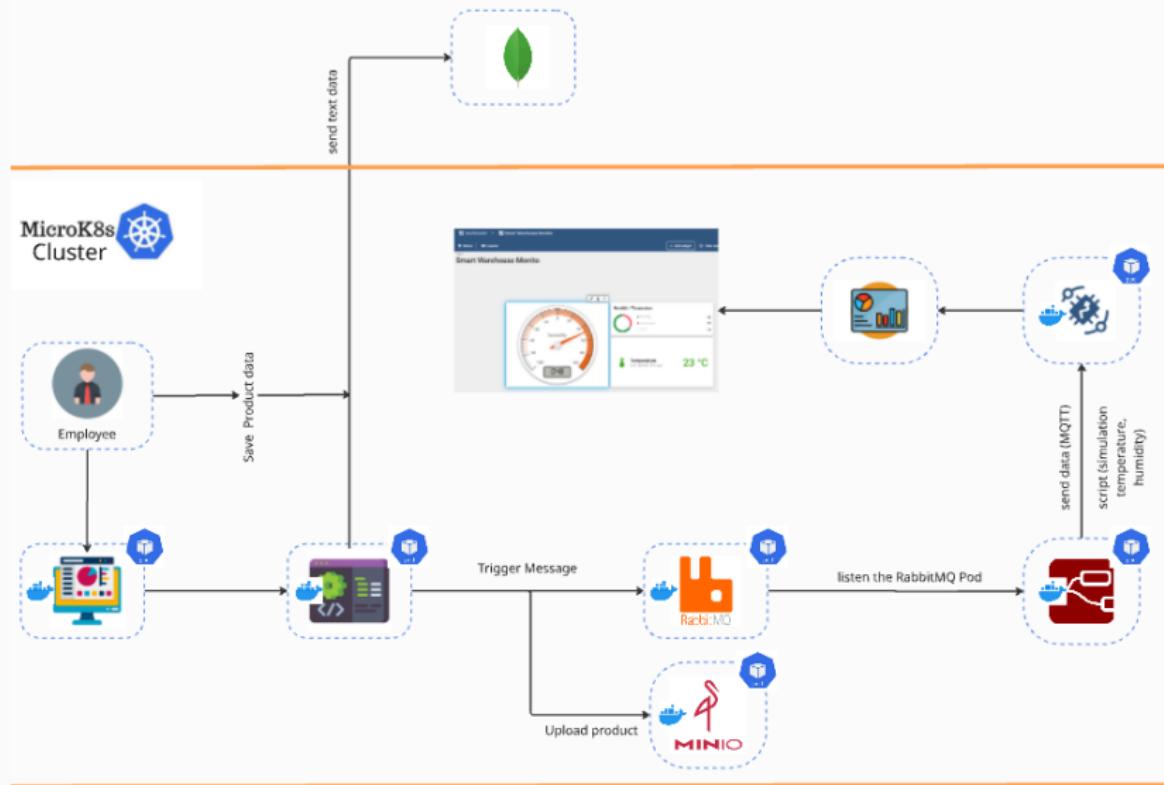


Figure 8: Data flow diagram for "Add product" scenario

## Conclusions & Future Work

---

- **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.

Get the source of the whole project from

*[github.com/drososkats/fabric\\_cloud.git](https://github.com/drososkats/fabric_cloud.git)*

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



ΧΑΡΟΚΟΠΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ  
HAROKOPIO UNIVERSITY



Thank you! / Questions? ☺