Distributed Computation Framework

MSc Computer Science

Technologies for Big Data Management

A.Y. 2024/2025

About Us

Fueled by passion, driven by technology

We are engineering tomorrow's world

Site Reliability Engineer

Christian Bieri

DevOps Engineer

Frederico Fischer

DevOps Engineer

Leandro Hoenen

Agenda

An overview of the project, its structure and outlook

1. Project Overview
2. Framework Architecture
3. Job/Container Handler & Data Generators
4. Live Demo
5. Limitations & Outlook
6. Learning Outcomes

Project Overview

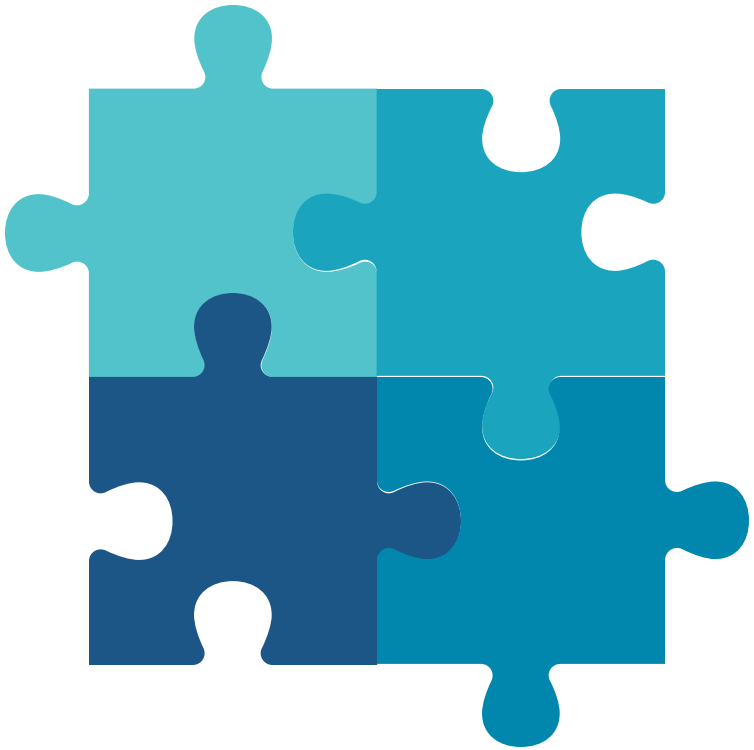
The Distributed Computation Framework

The Distributed Computation Framework (DCF) is designed to provide a user-friendly platform that manages the lifecycle of Docker containers. The objective of these containers is to simulate sample data and deliver it via distinct Kafka topics

Supplied Capabilities

Job Orchestration

A concurrent control server responsible for the efficient and performant orchestration of jobs.



Provide Flexibility

Submit jobs by providing a set of parameters (e.g., container image, Kafka topic, etc.)

Enable Scaling

Distribution of different jobs over different agents, allowing for multiple concurrent jobs, each with unique parameters

Abstract Complexity

Provide a graphical user interface and hide the underlying complexity

Simplify Integration

Language-agnostic container integration by using standard libraries

Framework Architecture

High-Level Architecture

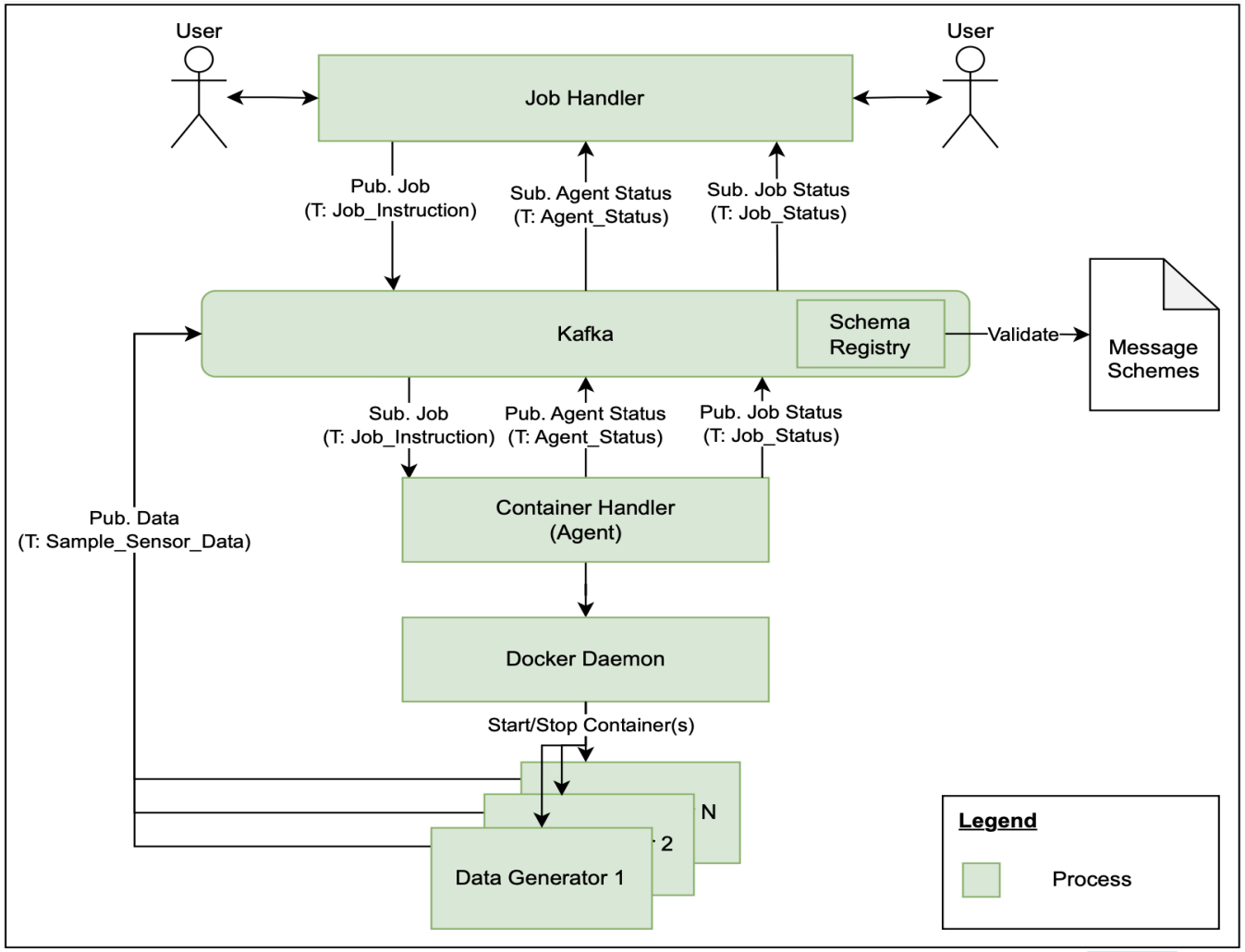
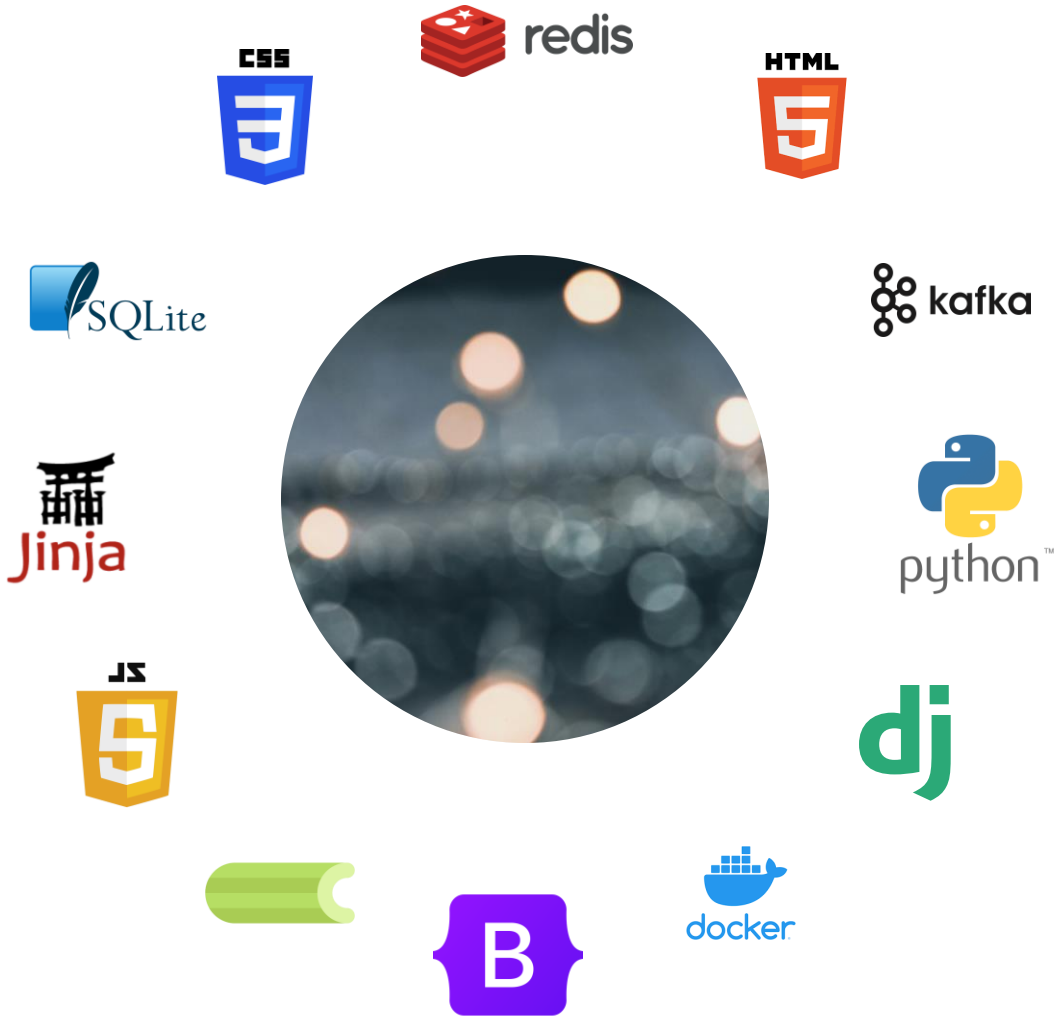


Figure 01: Distributed Computation Framework Architecture (own illustration)



Technologies

Empowering innovation with cutting-edge technology

To build flexible, scalable, and innovative solutions, we leverage a carefully curated technology stack. From backend frameworks to containerization, each tool is handpicked to fulfil our unique requirements and deliver exceptional results

Job/Container Handler & Data Generators

Job Handler

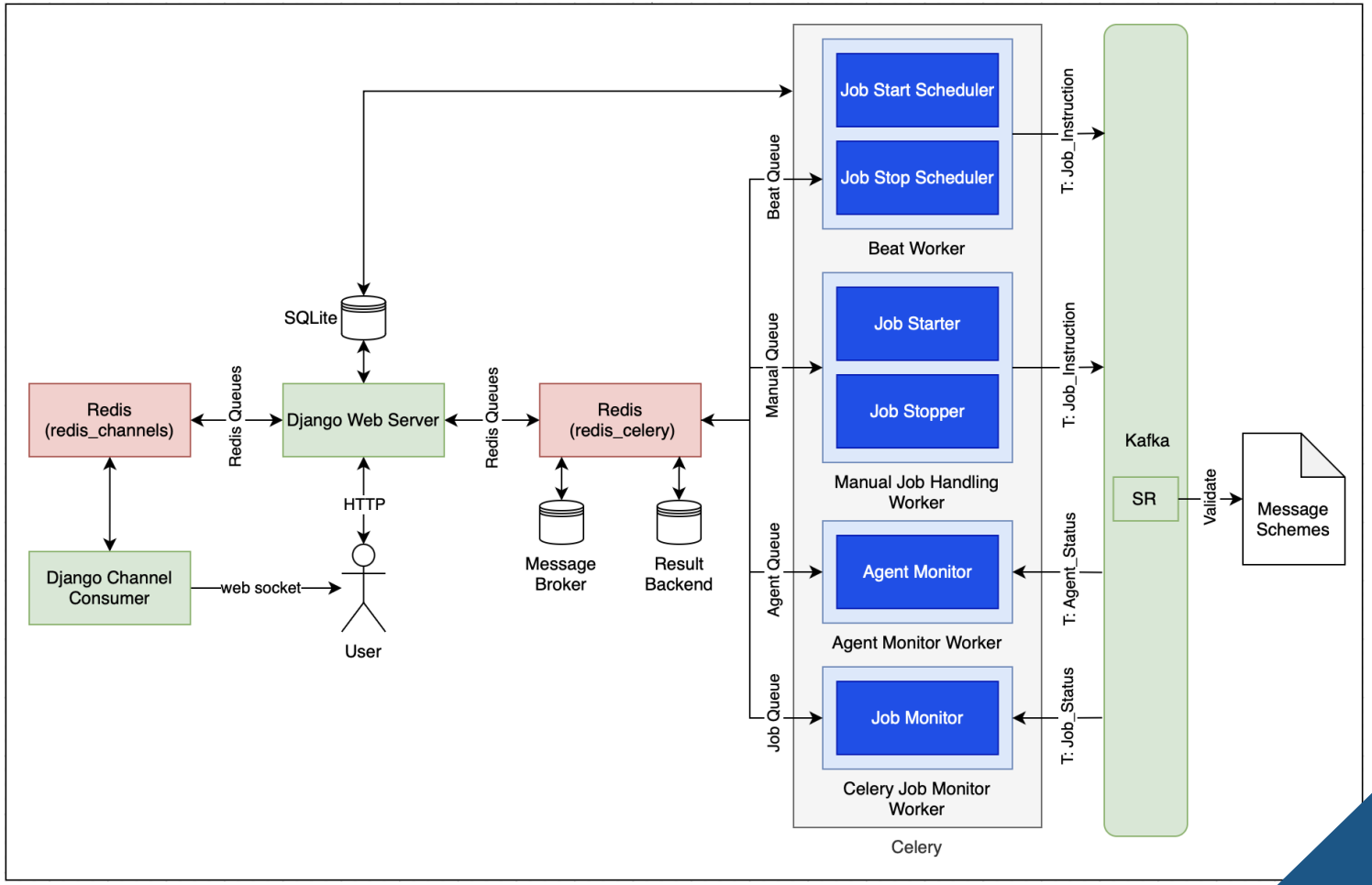


Figure 02: Job Handler Architecture (own illustration)

Container Handler (Agent)

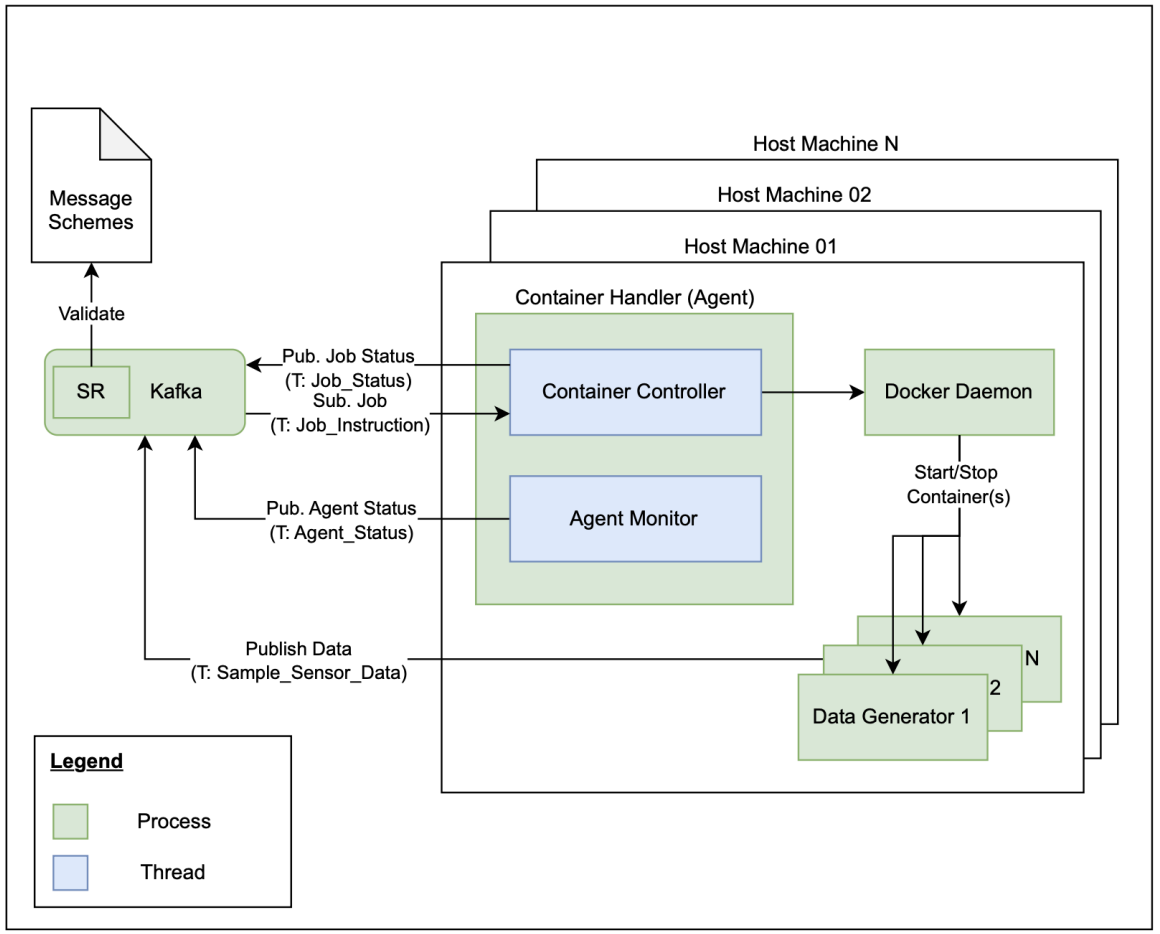
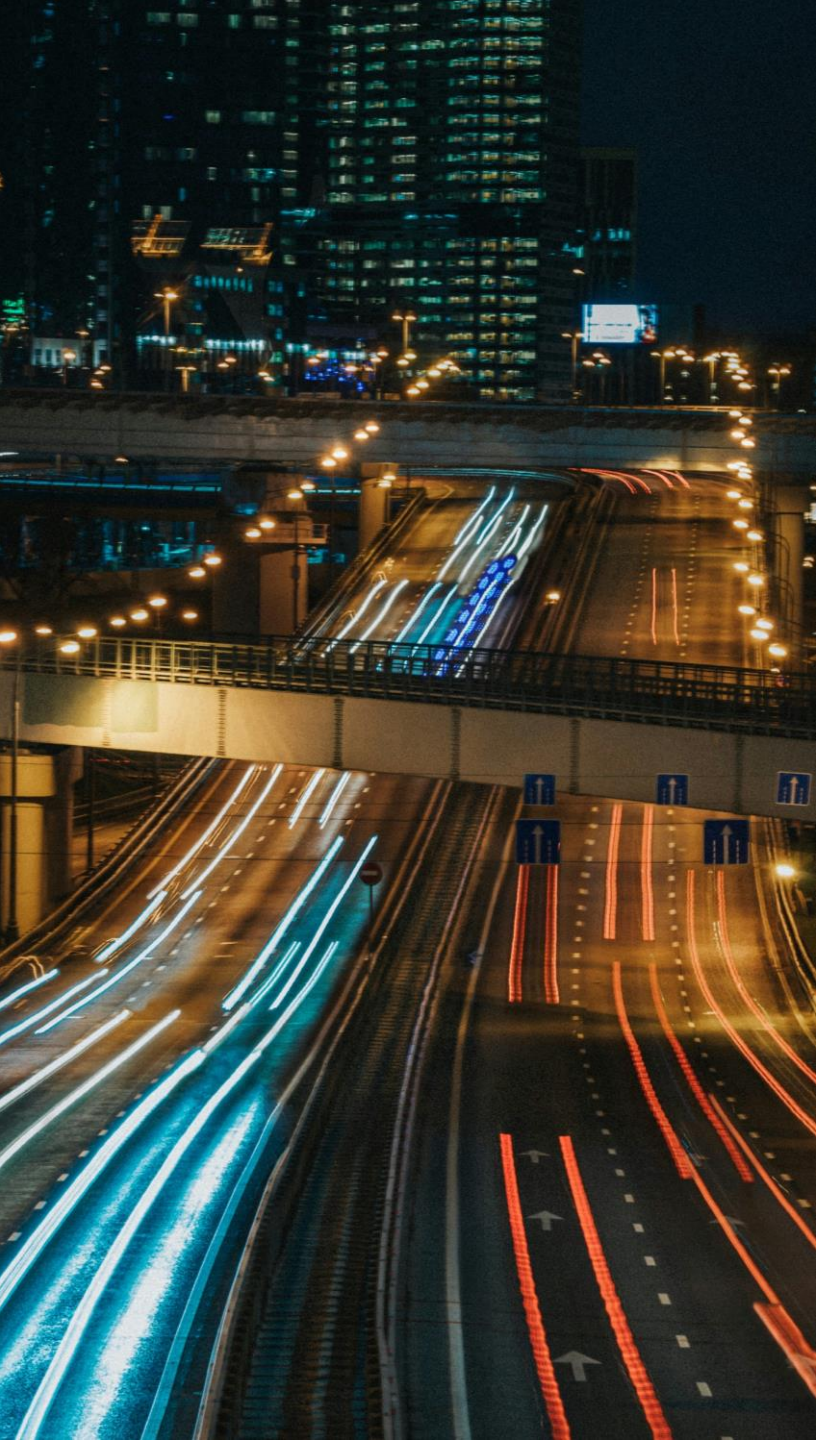


Figure 03: Container Handler (Agent) Architecture (own illustration)



Data Generators (Integration)

Language-agnostic container integration
by using standard libraries

Install and load dependencies

```
python3.13 -m pip install confluent-kafka
from confluent_kafka import Producer import os
```

Load environment variables and initialize Kafka producer

```
kafka_bootstrap_servers_docker = os.getenv("KAFKA_BOOTSTRAP_SERVERS_DOCKER")
kafka_topic = os.getenv("KAFKA_TOPIC")

if kafka_bootstrap_servers and kafka_topic:
        kafka_producer = Producer({"bootstrap.servers": kafka_bootstrap_servers_docker})
```

Produce data

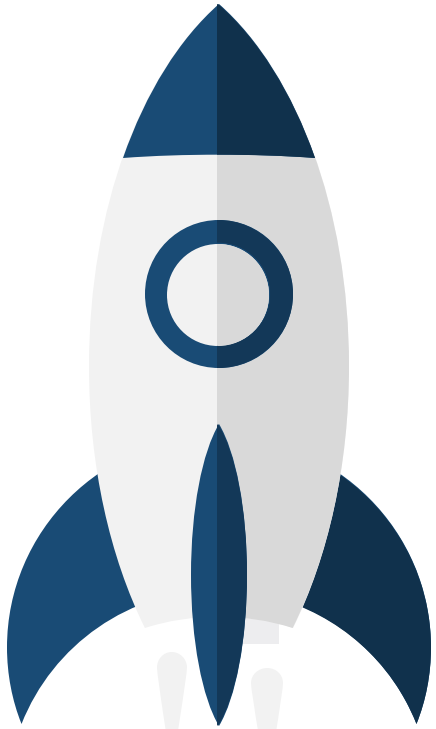
```
kafka_producer.produce(topic=kafka_topic, value=json.dumps(output_data).encode('utf-8'))
```

Example: https://github.com/massimocallisto/unicam-simulator/tree/feature/kafka-integration

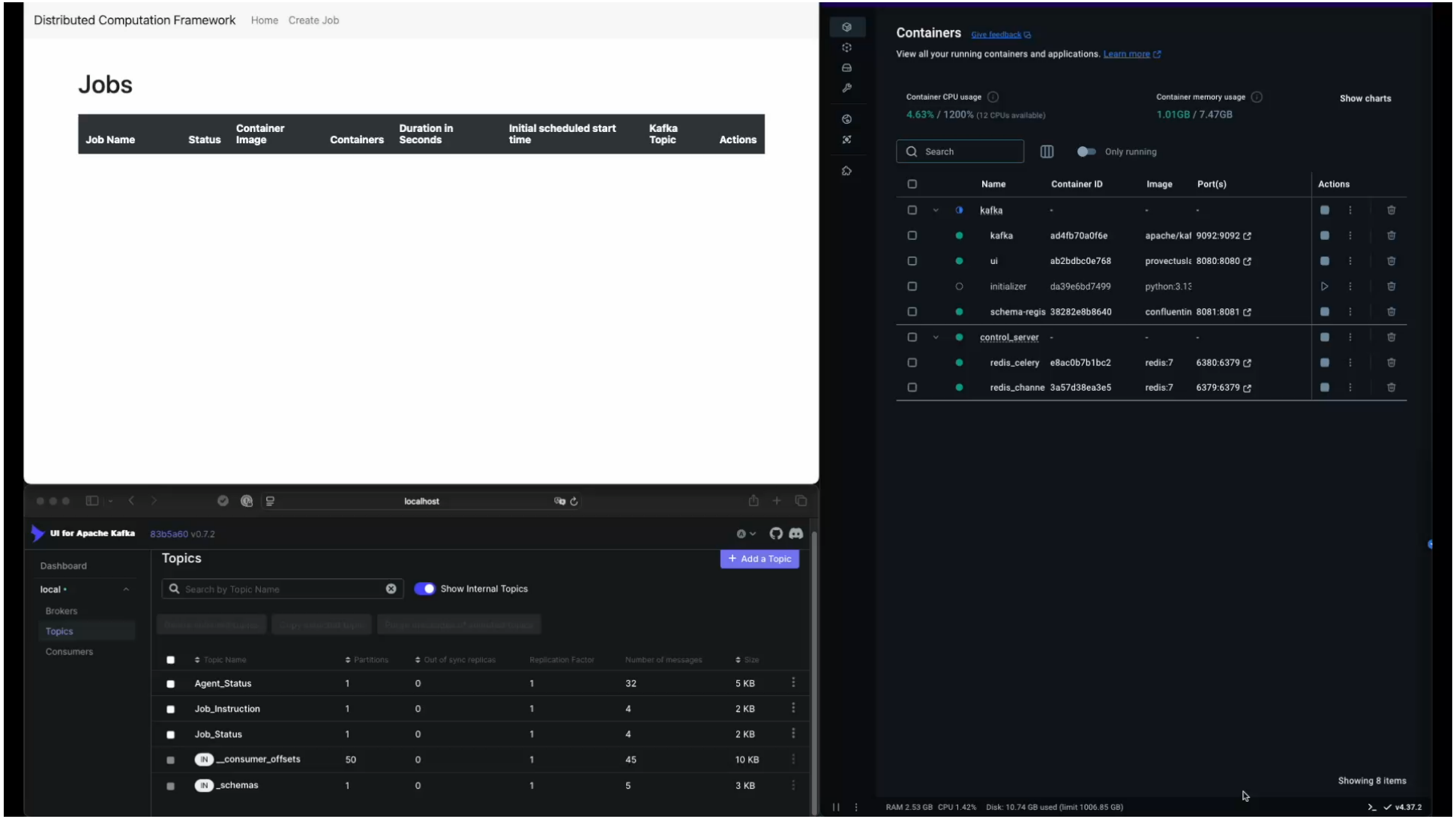
Live Demo

Live Demo

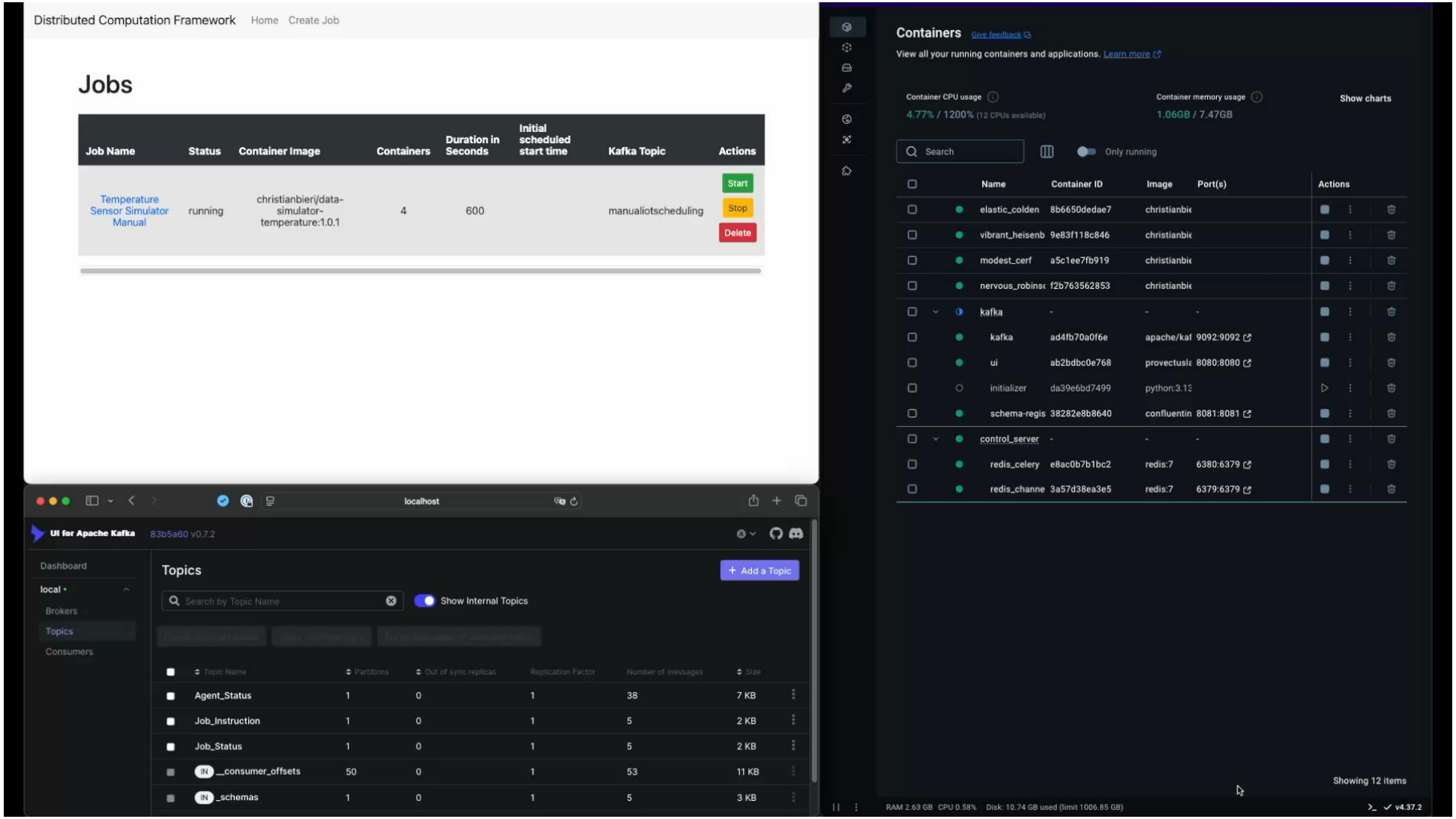
Convince yourself of our solution!



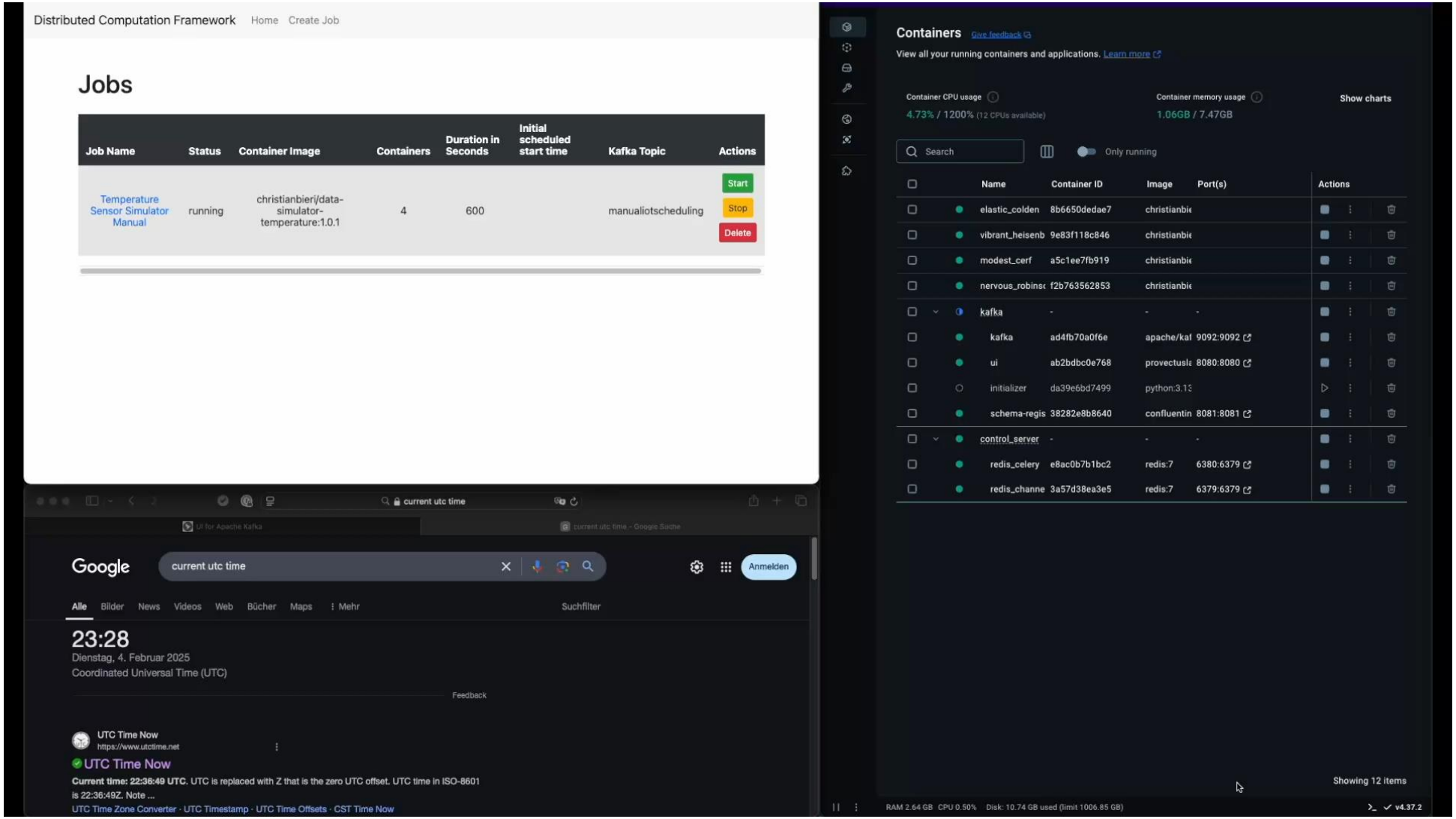
Manual Job Handling



Error Tolerance



Automated Job Scheduling



Limitations & Outlook

Limitations & Outlook

Single (Local) Host Setup

While appropriate for Kafka (Schema Registry) and the Job Handler, for the deployment in a production environment, unsuitable for Container Handler (Agent) and Data Generators

Limited Scalability

While working with a single partition is not an issue for the Job_Status and Agent_Status topic, the Job_Handling topic becomes faces limited scalability in such a setting

DockerHub Only

As of now, images of Data Generators can only be fetched from DockerHub

Basic Agent Monitoring

Currently, the Agent is subject to only one active check: The Docker daemon checker

Development-Only DB

As it stands, a SQLite DB is used. While sufficient for development purposes, it is not recommended for production environments

Containerization!

Containerize the Container Handler for the DCF to be truly distributed. Hence, running the Agent on multiple host machines is straightforward

Partitioning/Agent Adjustments!

Split Job_Handling into two topics (e.g., Job_Creation & Job_Deletion) and adjust Container Controller thread accordingly

Access to Multiple Container Registries!

Adjust the job handling Avro schema which not only involves the addition of information regarding the container registry, but also relevant authentication details

Additional Monitoring!

Incorporate supplementary checks such as those pertaining to resource availability In order to enhance reliability of the Agent

Production-Grade DB!

Migrate the DB to a more robust solution such as PostgreSQL

Learning Outcomes

Skills Acquired

Kafka, Schema Registry &
Avro Schemes

(Python) Docker API

Django, Django Channels, Redis,
Celery & multithreading

Distribution, Scalability &
Fault Tolerance

Collaboration &
Agile Development

Thank You!