#10

Data Platform Development

Docker Compose & Kafka

"Crafting a functional pipeline for realtime data streaming"

Table of Contents


Overview

Moduler:

- Docker Compose
- Kafka *(and terminologies)*
- Kafka Consumers *(Bonus)*

Utbildningsmoment

- Dataplatfformar, bakgrund och syfte
- Git och github i teamkontext ✅
- Komponenter och teknologier i en data platform ✅
- ETL vs ELT
- Utveckling av mjukvara mot databaser ✅
- Använda Python mot relationsdatabaser och andra datakällor såsom csv, http xml/json ✅
- Använda Python mot realtidsdataströmmar såsom message queues och/eller event streaming platforms ✅
- Använda Python och för att rensa, validera och transformera data
- Workflow processer ✅

02

Docker Compose


Multi-containers
Why?

Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience.

Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.

Compose works in all environments - production, staging, development, testing, as well as CI workflows. It also has commands for managing the whole lifecycle of your application:

- Start, stop, and rebuild services
- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

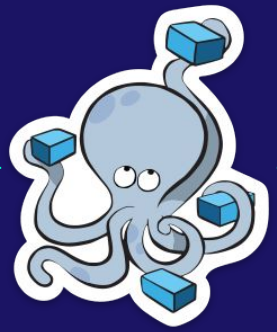
Docker Compose
(Multi-containers)

När används det?

- Development Environments *(databases, queues, caches, web service APIs, etc)*
- Automated Testing Environments *(creates & destroys containers)*
- Multiple services, one host

Läs mer: https://docs.docker.com/compose/

Creates a **private network**



Dockerfile

FastAPI app

Dockerfile

PostgreSQL

Dockerfile

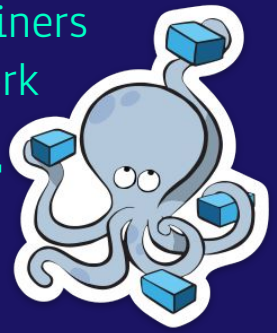
Backend

Private Network

```
DATABASE_URL = "postgresql://postgres:password@db:5432/appdb"
```

@db = hostname *(DNS)*

Attaches all service containers
to the same private network



```
services:
  api:
    build: .
  db:
    image: postgres:16
```

A service is an abstract definition of a computing resource within an application which can be scaled or replaced independently from other components. Services are backed by a set of containers, run by the platform according to replication requirements and placement constraints. As services are backed by containers, they are defined by a Docker image and set of runtime arguments. All containers within a service are identically created with these arguments.

A Compose file must declare a `services` top-level element as a map whose keys are string representations of service names, and whose values are service definitions. A service definition contains the configuration that is applied to each service container.

Läs mer https://docs.docker.com/reference/compose-file/services/

Example of what a **collection** of **services** look like:

```yaml
version: "3.9"

services:
 api:
   build: .
   ports:
     - "8000:8000"
   depends_on:
     - db
   environment:
     DATABASE_URL: postgresql://postgres:postgres@db:5432/appdb

 db:
   image: postgres:16
   environment:
     POSTGRES_USER: postgres
     POSTGRES_PASSWORD: postgres
     POSTGRES_DB: appdb
```


Download Project
Git Clone

Clone the repository

Open terminal

```
git clone <repo-url>
cd <project-folder>
```

Install dependencies

From the project root:

```
uv sync
```

This creates a virtual environment and installs all dependencies from `uv.lock`.

Create a file named `.env` in the project root : Or checkout the `.env-example` file for quicker setup

```
DB_USER=postgres
DB_PASSWORD=postgres
DB_HOST=localhost
DB_PORT=5432
DB_NAME=products
```

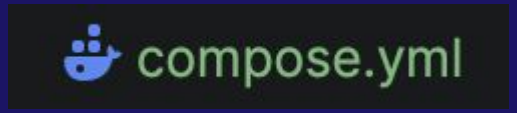
https://github.com/Krillinator/data-platform-lektion-10-kafka-docker-compose/tree/main


Docker Compose
Step by Step

Docker Compose
(FastAPI + PostgreSQL)

NOTERA: Vi behöver inte PostgreSQL installerat eller öppet här!
För att: Den kommer skapas av en ‘image’. Dockers superkraft!

Docker Compose
(File)



Kan också heta ‘docker-compose’

compose.yml

```yaml
services:

 postgres: # must match .env hostname
   image: postgres:18
   container name: postgres
   environment:
     POSTGRES USER: ${DB USER}
     POSTGRES PASSWORD: ${DB_PASSWORD}
     POSTGRES_DB: ${DB_NAME}
   volumes:
     - postgres_data:/var/lib/postgresql

 app:
   build: .
   env file:
     - .env
   depends on:
     - postgres
   ports:
     - "8000:8000"

volumes:
 postgres_data:
```

Docker Compose (Content)

Docker containers are ‘ephemeral’ vilket innebär att data tas bort.

‘Volumes’ vidbehåller data

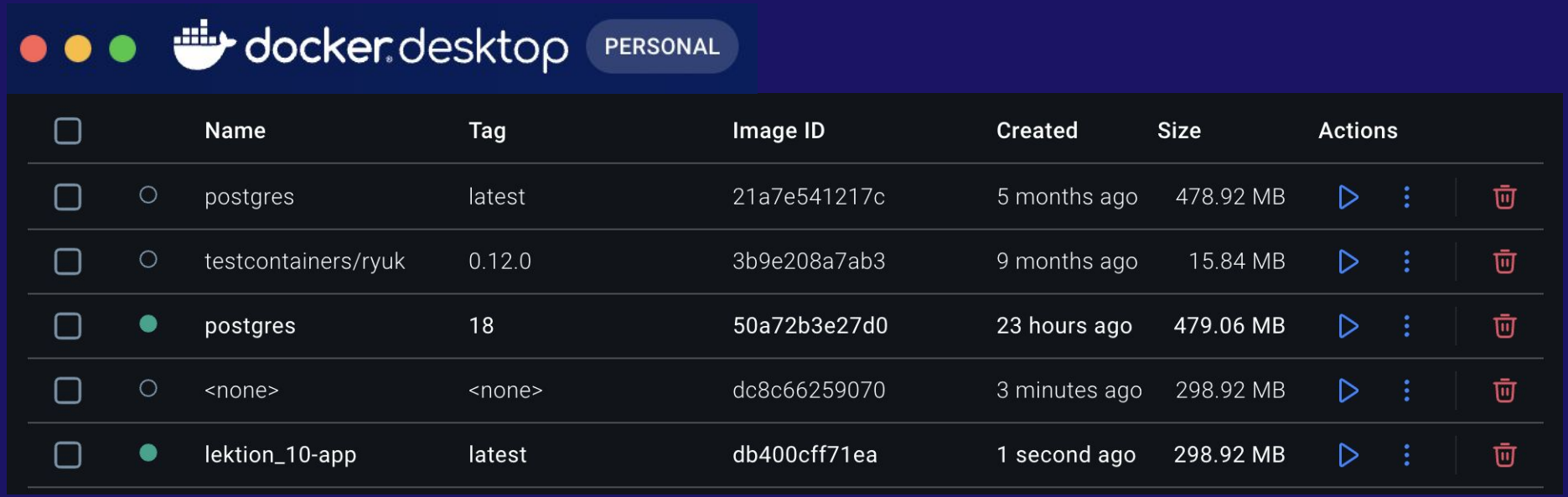
Docker Compose (Build)

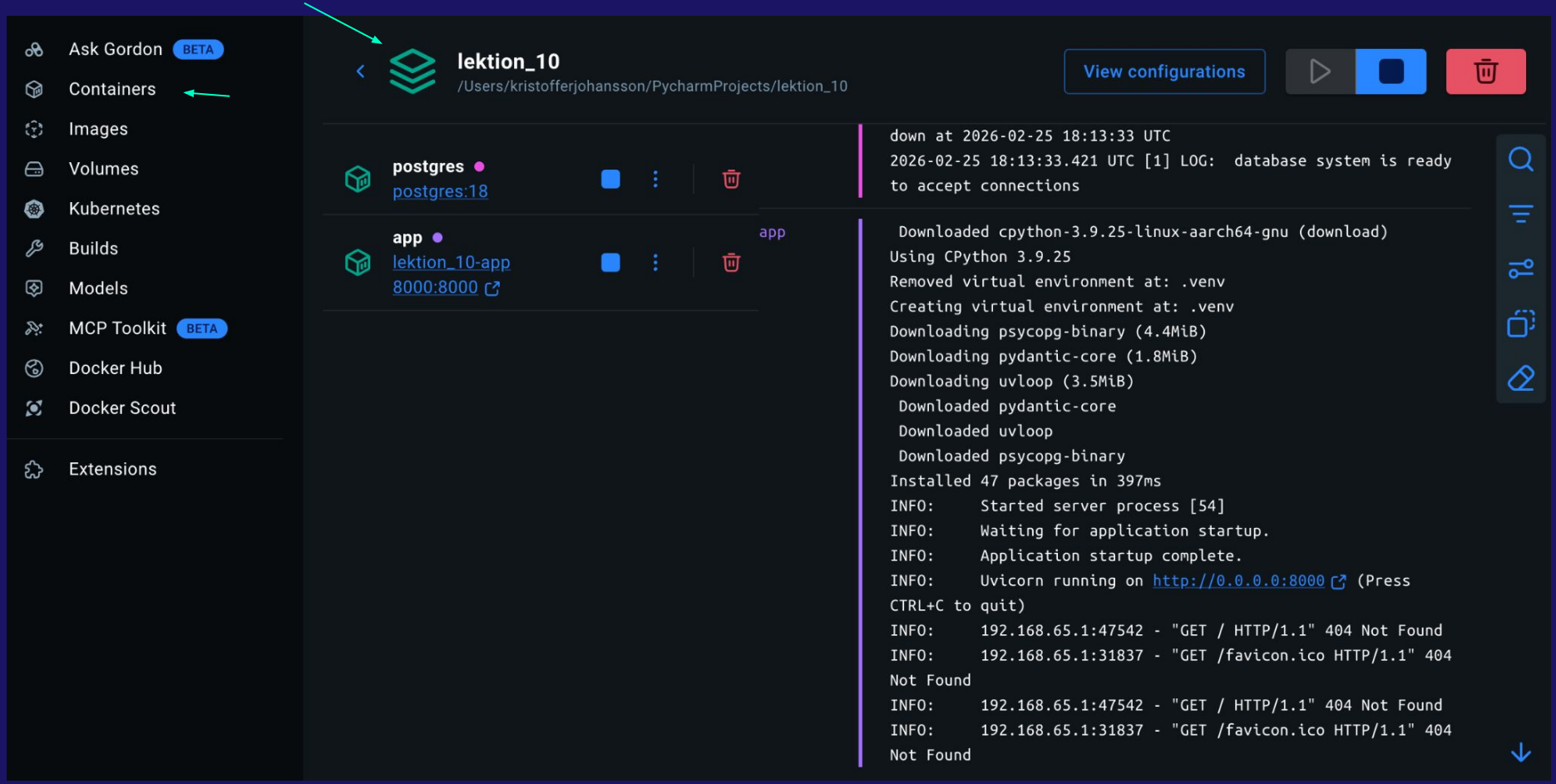
```
$ docker compose up --build
```

This will read the Dockerfile, build the image and start the containers.

```
(.venv) kristofferjohansson@Kristoffers-MacBook-Pro lektion_10 % docker compose up --build
[+] up 12/14
 ⠸ Image postgres:18 [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿⣦⣀⣿⣿⣿⣿⣿]   109MB / 160.9MB Pulling
```

Docker Compose
(up and running)



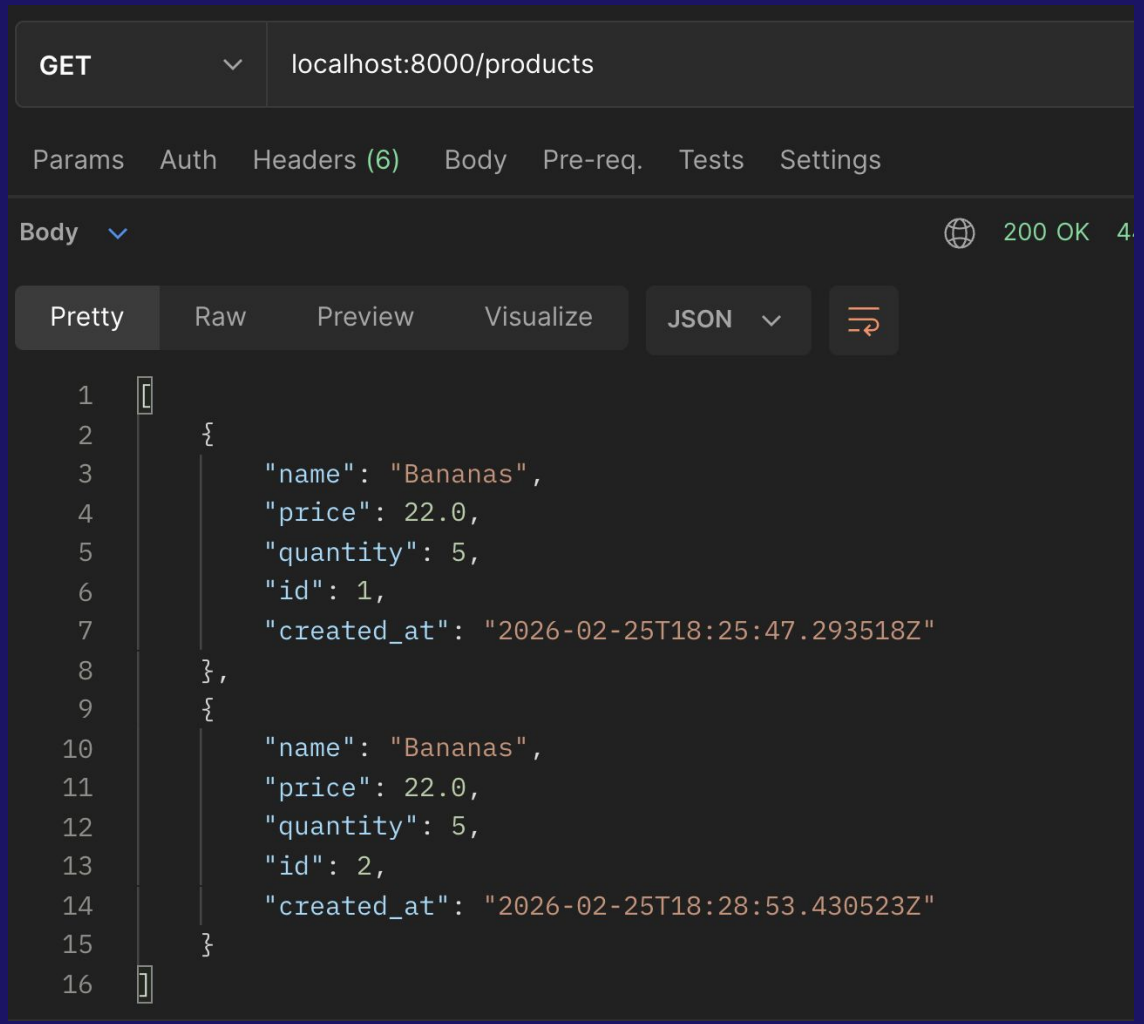

Ask Gordon BETA
Containers
Images
Volumes
Kubernetes
Builds
Models
MCP Toolkit BETA
Docker Hub
Docker Scout
Extensions
lektion_10
/Users/kristofferjohansson/PycharmProjects/lektion_10
View configurations
postgres
postgres:18
app
lektion_10-app
8000:8000
app
down at 2026-02-25 18:13:33 UTC
2026-02-25 18:13:33.421 UTC [1] LOG: database system is ready to accept connections
Downloaded cpython-3.9.25-linux-aarch64-gnu (download)
Using CPython 3.9.25
Removed virtual environment at: .venv
Creating virtual environment at: .venv
Downloading psycopg-binary (4.4MiB)
Downloading pydantic-core (1.8MiB)
Downloading uvloop (3.5MiB)
Downloaded pydantic-core
Downloaded uvloop
Downloaded psycopg-binary
Installed 47 packages in 397ms
INFO: Started server process [54]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: 192.168.65.1:47542 - "GET / HTTP/1.1" 404 Not Found
INFO: 192.168.65.1:31837 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO: 192.168.65.1:47542 - "GET / HTTP/1.1" 404 Not Found
INFO: 192.168.65.1:31837 - "GET /favicon.ico HTTP/1.1" 404 Not Found

POST http://localhost:8000/products

Params Auth Headers (8) Body Pre-req. Tests Settings

raw JSON

```
{
    "name": "Bananas",
    "price": 22,
    "quantity": 5
}
```



Detta kommer ligga kvar även under omstart

03

Kafka


Kafka
What is it?

Apache Kafka

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.

Manufacturing

10 OUT OF 10

Banks

7 OUT OF 10

Insurance

10 OUT OF 10

Telecom

8 OUT OF 10

Transportation

8 OUT OF 10

Energy and Utilities

10 OUT OF 10

Above is a snapshot of the number of top-ten largest companies using Kafka, per-industry.

Läs mer: https://kafka.apache.org/

Core capabilities

Kafka boasts core capabilities that are battle tested and ready to power businesses in the digital world.

High Throughput

Deliver messages at network limited throughput using a cluster of machines with latencies as low as 2ms.

Scalable

Scale production clusters up to a thousand brokers, trillions of messages per day, petabytes of data, hundreds of thousands of partitions. Elastically expand and contract storage and processing.

Permanent Storage

Store streams of data safely in a distributed, durable, fault-tolerant cluster.

High Availability

Stretch clusters efficiently over availability zones or connect separate clusters across geographic regions.

Built-in Stream Processing

Process streams of events with joins, aggregations, filters, transformations, and more, using event-time and exactly-once processing.

Connect to almost anything

Kafka's out-of-the-box Connect interface integrates with hundreds of event sources and event sinks including Postgres, JMS, Elasticsearch, AWS S3, and more.

History (bonus)
(Zookeeper vs KRaft)

ZooKeeper is a **standalone distributed system** used for coordination.

Separate installation that was ran as its own cluster.
Other systems (like Kafka) connected to it. *(Detta innebär att det inte var integrerat)*

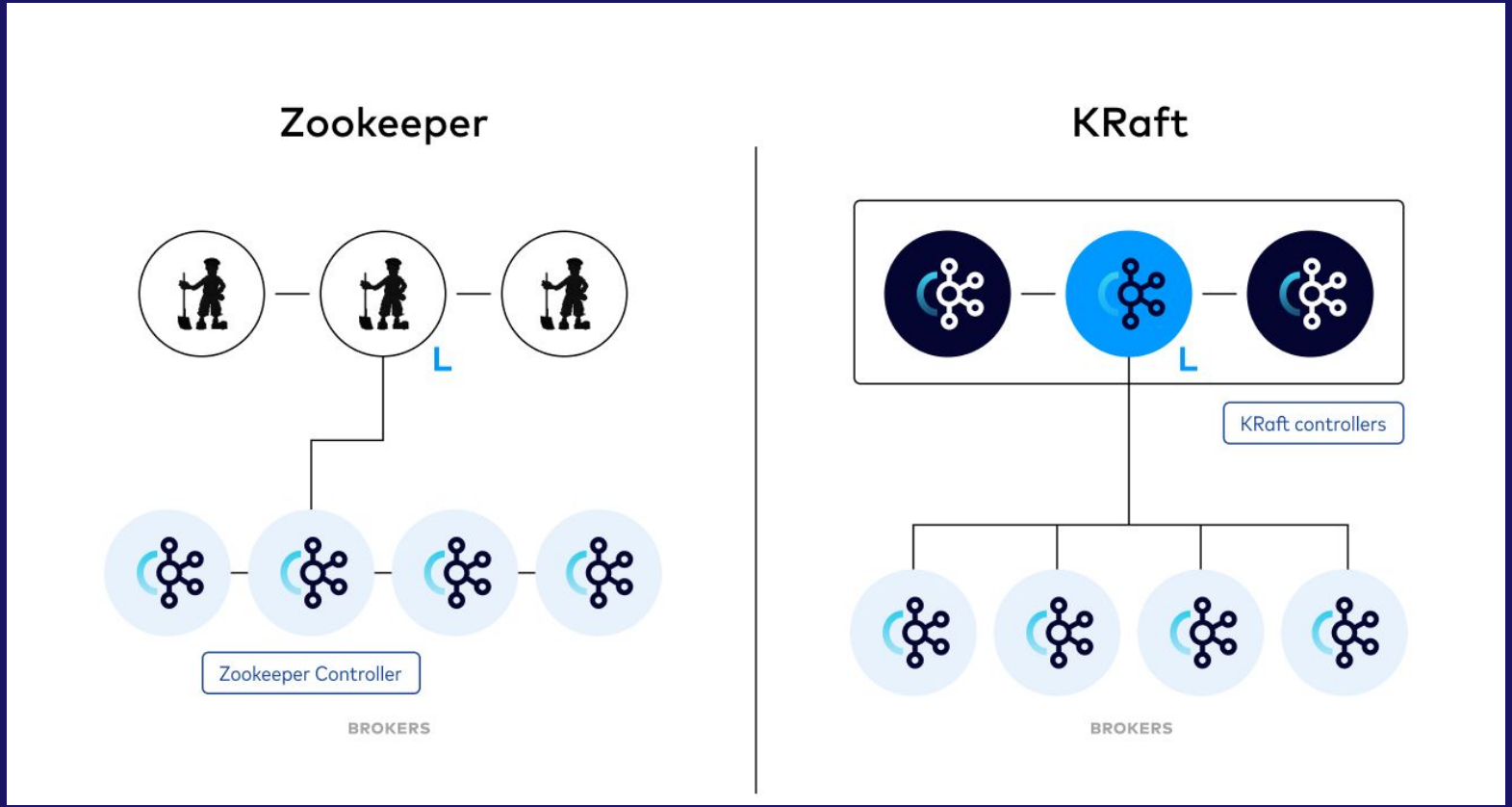
History (bonus) #2
(Zookeeper vs KRaft)

It's a **coordination service**.

It stores small pieces of shared metadata like:

- Who is leader?
- Which nodes are alive?
- What configuration exists?
- What topics exist (in Kafka's case)?

ZooKeeper:
Ett separat distribuerat koordineringssystem



Läs mer https://kafka.apache.org/42/operations/kraft/

Kafka & Coordination
(Why is it needed?)

För att Kafka är ett **distribuerat system**.

- Distribuerat = flera servrar samtidigt.

Så fort du har flera servrar uppstår problem

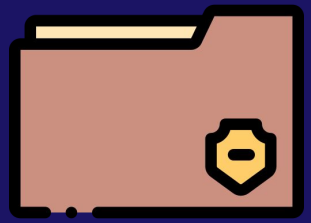

Kafka
Terminologies

Kafka Terms
(Record)

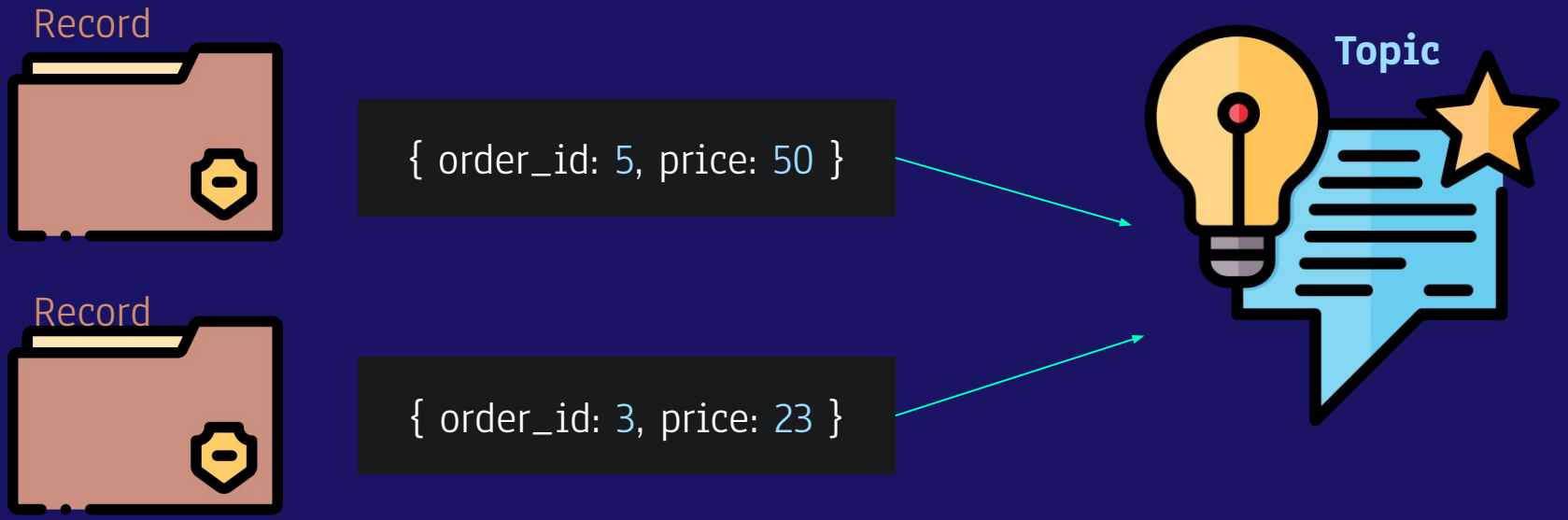
En **record** är: Ett enskilt meddelande

Exempelvis:

```
</> JSON

{ "order_id": 123, "amount": 500 }
```



Kafka Terms
(Records are sent to Topics)



Kafka Terms
(Partition)

```
Topic: orders
   ├── Partition 0
   ├── Partition 1
   └── Partition 2
```

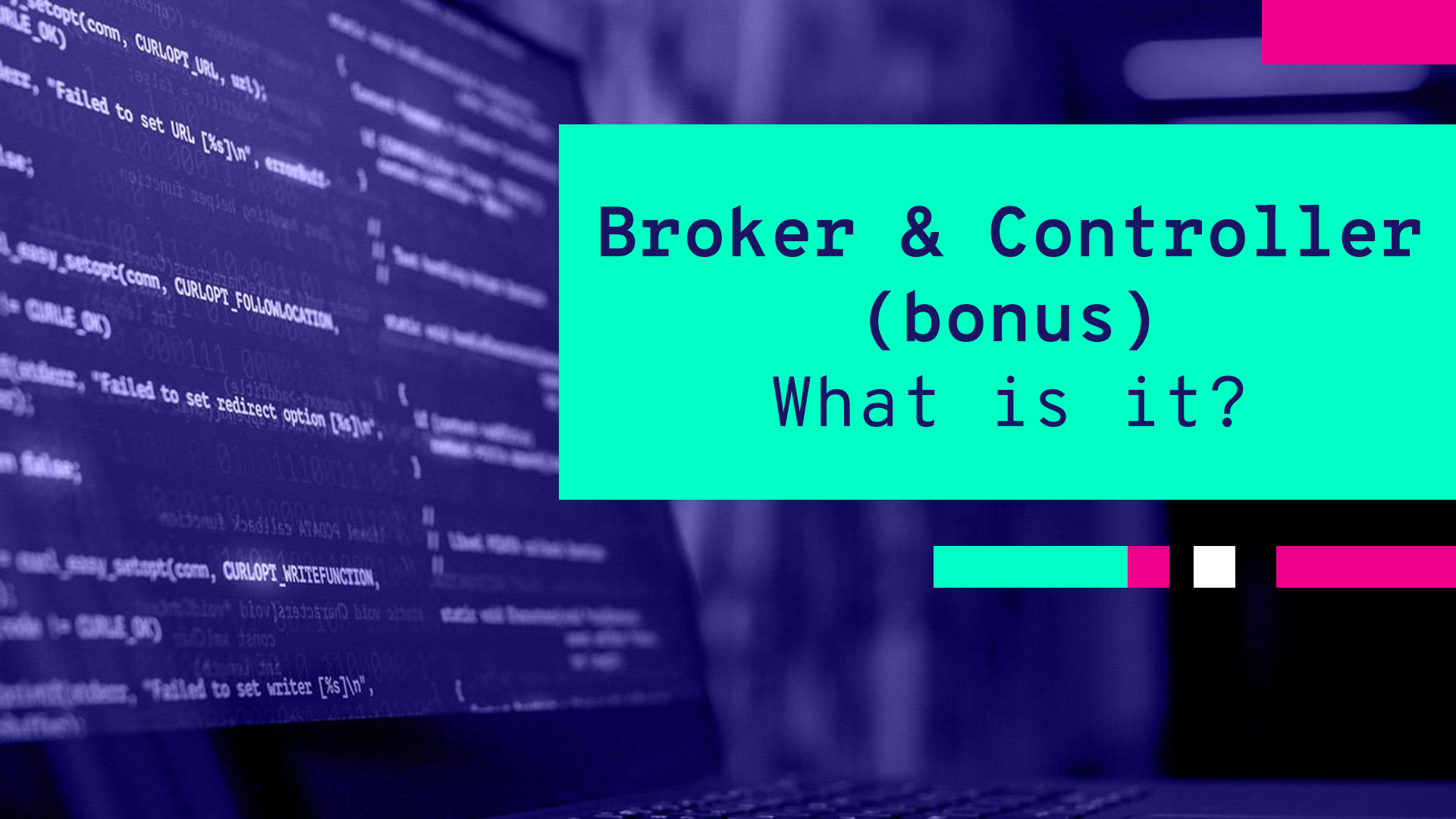
En partition innehåller flera Records

Partition är en strikt ‘append only’ log

```
Partition 0
Offset 0 → {order A}
Offset 1 → {order B}
Offset 2 → {order C}
```

```
Topic: orders
  Partition 0
    offset 0 → {order 123}
    offset 1 → {order 124}

  Partition 1
    offset 0 → {order 200}
    offset 1 → {order 201}
```

Sammanfatta (Termer)


Broker & Controller
(bonus)
What is it?

Kafka Broker
(What is it?)

A server process that stores topic data, handles client requests (producers/consumers), and replicates partitions.

Kafka brokers:

- Accepts messages from producers *(producers is a client app that sends a message (topic))*
- Stores them on disk
- Serves messages to consumers
- Replicates data to other brokers
- Hosts topic partitions
- May be leader or follower for partitions

Kafka Controller
(What is it?)

Controllern styr **metadata**, inte data. Den bestämmer:

- Vem ska vara leader för partition 0?
- Vad händer om Broker 1 dör?
- Hur ska partitions fördelas?
- När skapas topics?

Den hanterar **klusterlogik**, inte records.

Configuration

Process Roles

In KRaft mode each Kafka server can be configured as a controller, a broker, or both using the `process.roles` property. This property can have the following values:

- If `process.roles` is set to `broker`, the server acts as a broker.
- If `process.roles` is set to `controller`, the server acts as a controller.
- If `process.roles` is set to `broker,controller`, the server acts as both a broker and a controller.

Kafka servers that act as both brokers and controllers are referred to as “combined” servers. Combined servers are simpler to operate for small use cases like a development environment. The key disadvantage is that the controller will be less isolated from the rest of the system. For example, it is not possible to roll or scale the controllers separately from the brokers in combined mode. Combined mode is not recommended in critical deployment environments.


Kafka
Docker Compose

Image: https://hub.docker.com/r/bitnami/kafka

compose.yml

```yaml
kafka:
 image: apache/kafka:4.1.1
 container_name: kafka
 ports:
   # host -> kafka (for tools running on your laptop)
   - "29092:29092"
 volumes:
   - kafka data:/bitnami
 environment:
   KAFKA NODE ID: "1"
   KAFKA PROCESS ROLES: "broker,controller"
   KAFKA CONTROLLER QUORUM VOTERS: "1@kafka:9093"
   KAFKA LISTENERS: "PLAINTEXT://:9092,CONTROLLER://:9093"
   KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka:9092"
   KAFKA LISTENER SECURITY PROTOCOL MAP: "PLAINTEXT:PLAINTEXT,CONTROLLER:PLAINTEXT"
   KAFKA CONTROLLER LISTENER NAMES: "CONTROLLER"
   KAFKA INTER BROKER LISTENER NAME: "PLAINTEXT"
   KAFKA OFFSETS TOPIC REPLICATION FACTOR: "1"
   KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: "1"
   KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: "1"
```

Docs: https://hub.docker.com/r/apache/kafka

compose.yml

```yaml
volumes:
 postgres_data:
 kafka_data:
```

Uppdatera ‘volumes’ & depends ‘on’

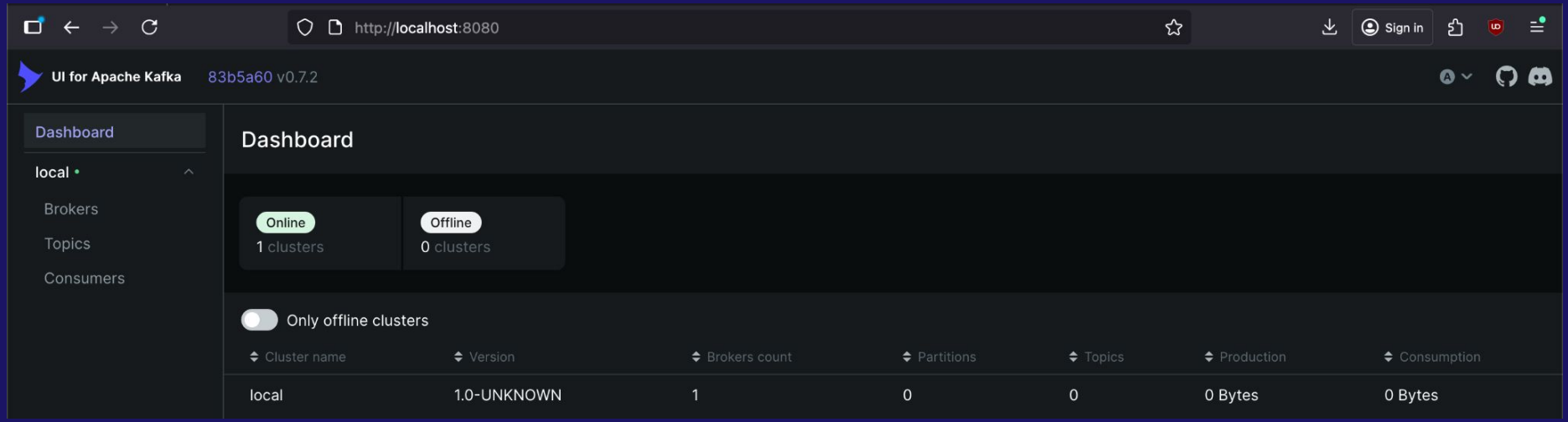
```yaml
app:
  build: .
  env_file:
    - .env
  depends_on:
    postgres:
      condition: service_started
    kafka:
      condition: service_started

  ports:
    - "8000:8000"
```

Kafka UI
Bonus

compose.yml

```yaml
kafka-ui:
 image: provectuslabs/kafka-ui:latest
 container_name: kafka-ui
 ports:
   - "8080:8080"
 environment:
   KAFKA_CLUSTERS_0_NAME: local
   KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka:9092
 depends_on:
   - kafka
```

Inkluderar ett användargränssnitt



Bra för debugging och visuell feedback

Kafka cluster = en/flera Kafka brokers som arbetar tillsammans


Kafka Streams
With Products

Kafka Python (synchronous)
(install)

```
$ uv add kafka-python
```

Injicera .env filen!

.env
(Adding topics & servers)

.env

```
KAFKA_BOOTSTRAP_SERVERS=kafka:9092
PRODUCTS_TOPIC=products.created
```

Main.py
(Loading Kafka .env variables)

main.py

```
# Kafka Setup
KAFKA_BOOTSTRAP_SERVERS = os.getenv("KAFKA_BOOTSTRAP_SERVERS",
"kafka:9092")
PRODUCTS_TOPIC = os.getenv("PRODUCTS_TOPIC", "products.created")
```

Main.py
(Setup Producer)

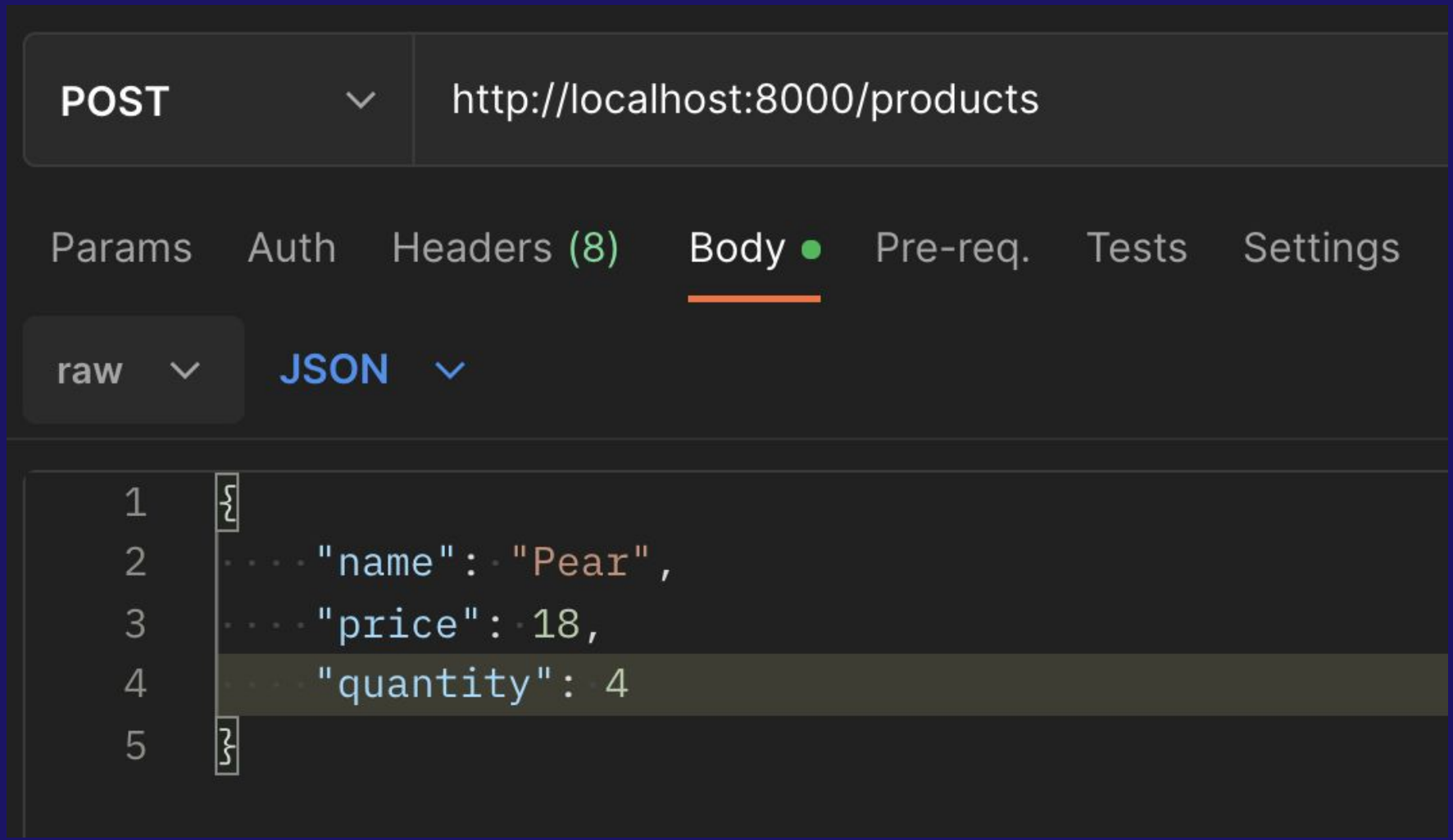
main.py

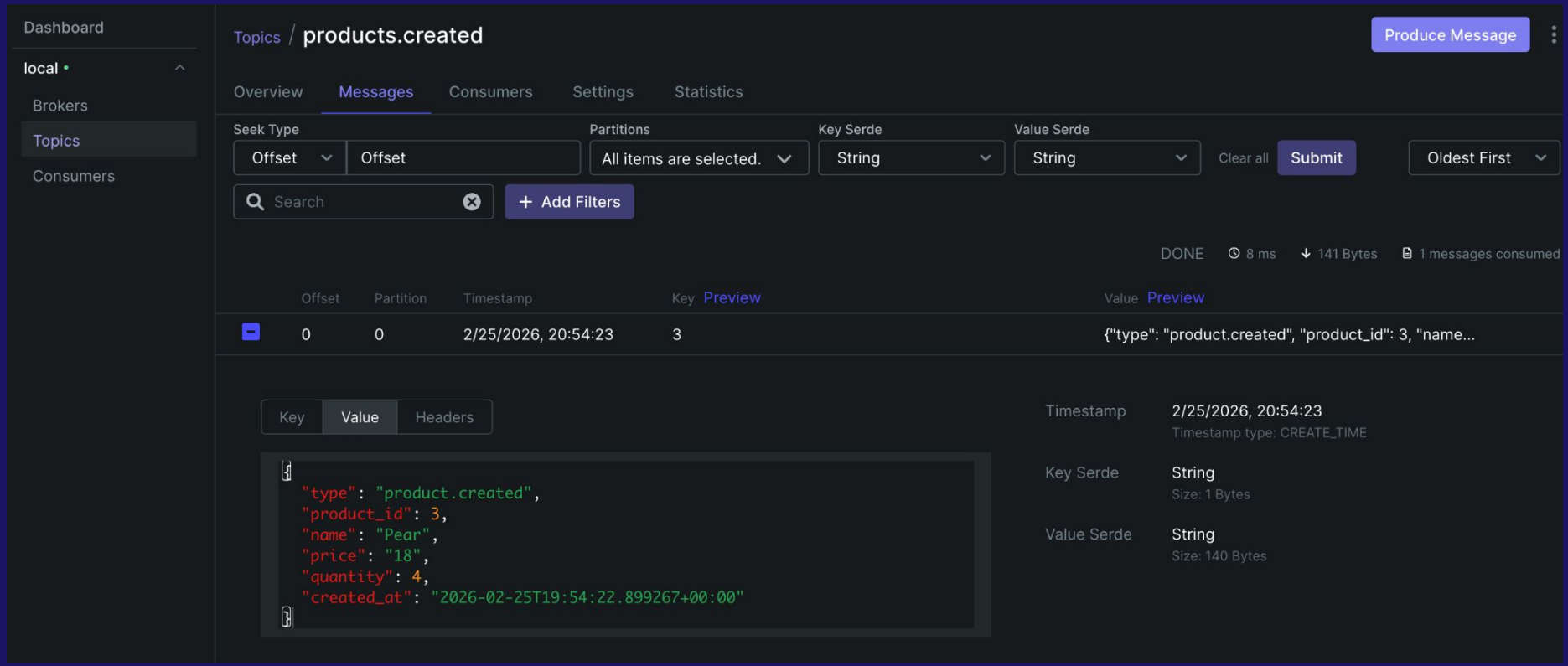
```python
# Kafka producer (sync)
app.state.kafka_producer = KafkaProducer(
    bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS,
    value_serializer=lambda v: json.dumps(v).encode("utf-8"),
    key_serializer=lambda k: k.encode("utf-8") if k else None,
)

yield

# Shutdown
try:
    app.state.kafka_producer.close()
except Exception:
    pass

app.state.pool.close()
```

Main.py
(Setup Producer)

main.py

```python
# Publish "product.created" event to Kafka
event = {
    "type": "product.created",
    "product_id": row["id"],
    "name": row["name"],
    "price": str(row["price"]),
    "quantity": row["quantity"],
    "created_at": row["created_at"].isoformat(),
}

app.state.kafka_producer.send(
    PRODUCTS_TOPIC,
    key=str(row["id"]),
    value=event,
)
```


POST
http://localhost:8000/products
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
raw
JSON
{
"name": "Pear",
"price": 18,
"quantity": 4
}


Dashboard
local
Brokers
Topics
Consumers
Topics / products.created
Produce Message
Overview
Messages
Consumers
Settings
Statistics
Seek Type
Offset
Offset
Partitions
All items are selected.
Key Serde
String
Value Serde
String
Clear all
Submit
Oldest First
Search
Add Filters
DONE
8 ms
141 Bytes
1 messages consumed
Offset
Partition
Timestamp
Key Preview
Value Preview
0
0
2/25/2026, 20:54:23
3
{"type": "product.created", "product_id": 3, "name...
Key
Value
Headers
{
"type": "product.created",
"product_id": 3,
"name": "Pear",
"price": "18",
"quantity": 4,
"created_at": "2026-02-25T19:54:22.899267+00:00"
}
Timestamp
2/25/2026, 20:54:23
Timestamp type: CREATE_TIME
Key Serde
String
Size: 1 Bytes
Value Serde
String
Size: 140 Bytes


Listen for change
With Products

compose.yml

```yaml
consumer:
 build: .
 env_file:
   - .env
 depends_on:
   - kafka
 command: [ "uv", "run", "python", "-u", "app/consumer/print.py" ]
```

Docs: https://kafka-python.readthedocs.io/en/master/

```python
import os
import json
from kafka import KafkaConsumer

KAFKA_BOOTSTRAP_SERVERS = os.getenv("KAFKA_BOOTSTRAP_SERVERS", "kafka:9092")
TOPIC = os.getenv("PRODUCTS_TOPIC", "products.created")
GROUP_ID = "print-consumer-v2"

def main():
    consumer = KafkaConsumer(
        TOPIC,
        bootstrap_servers=KAFKA_BOOTSTRAP_SERVERS,
        group_id=GROUP_ID,
        auto_offset_reset="latest",  # only new messages from now on
        value_deserializer=lambda b: json.loads(b.decode("utf-8")),
        key_deserializer=lambda b: b.decode("utf-8") if b else None,
    )

    print(f"Listening on topic: {TOPIC}")
    for msg in consumer:
        print("got event, we could mail people on each purchase :) ", msg.value)

if __name__ == "__main__":
    main()
```

THANKS !

Do you have any questions?
kristoffer.johansson@sti.se