Making Computers Talk to Each Other: A Not-So-Technical Guide

An introduction to Kafka and deploying simple ML models with data streaming

Sreemaee Akshathala
Research Engineer, Software Engineering Research Centre
Chandrasekar S, Vyakhya Gupta
Research Students, Software Engineering Research Centre

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

SERC
Software Engineering Research Centre

What Problems Are We Solving Today?

Modern software needs to

- Handle large amounts of data
- Process information in real-time
- Be reliable and scalable
- Work across multiple computers
- Be easy to maintain and update

Docker

Your Software's Shipping Container

- **One unit**

  Everything your program needs is packed together

- **Consistency**

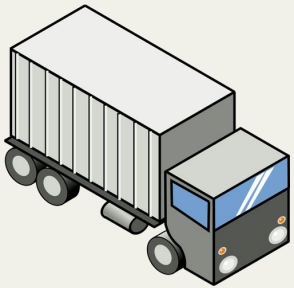
  Works the same way everywhere

- **Portability**

  Makes sharing and deploying software easier

- **Isolation**

  Like having a complete mini-computer for each part of your application



INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

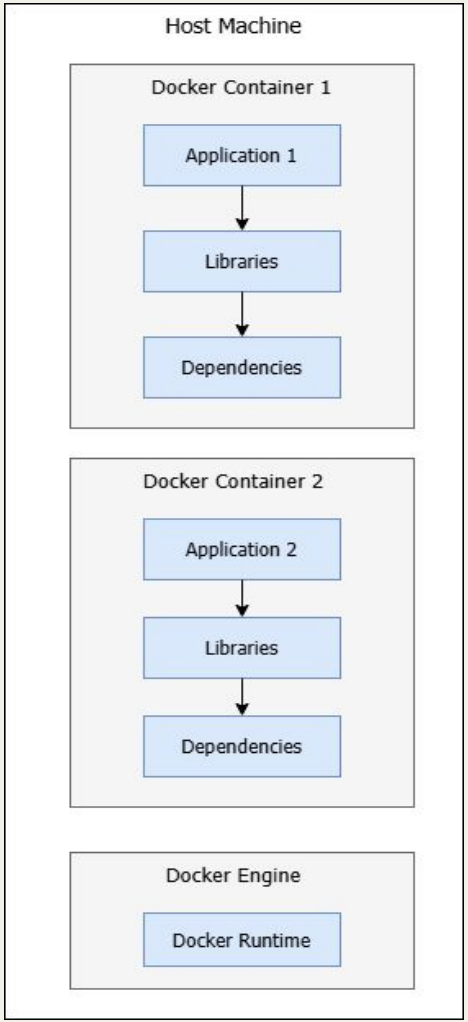
What problems does Docker solve?

- Applications often run into issues when moved between environments (e.g., "It works on my machine!").
- Differences in operating systems, dependencies, or configurations can break applications.
- Docker ensures that an application runs consistently no matter where it is deployed.

Definition:

Docker packages an application and all its dependencies into a ***container***. A container is a lightweight, standalone, and executable unit that includes everything needed to run the application (code, libraries, settings, etc.).


docker


Host Machine
Docker Container 1
Application 1
Libraries
Dependencies
Docker Container 2
Application 2
Libraries
Dependencies
Docker Engine
Docker Runtime

Navigate to the following repository

https://github.com/sa4s-serc/kafka_demo

Apache Kafka

The messenger system

The Publish-Subscribe Pattern:

- **Publishers (Producers):** Send messages to topics
- **Topics:** Categories for different types of messages
- **Subscribers (Consumers):** Receive messages from topics

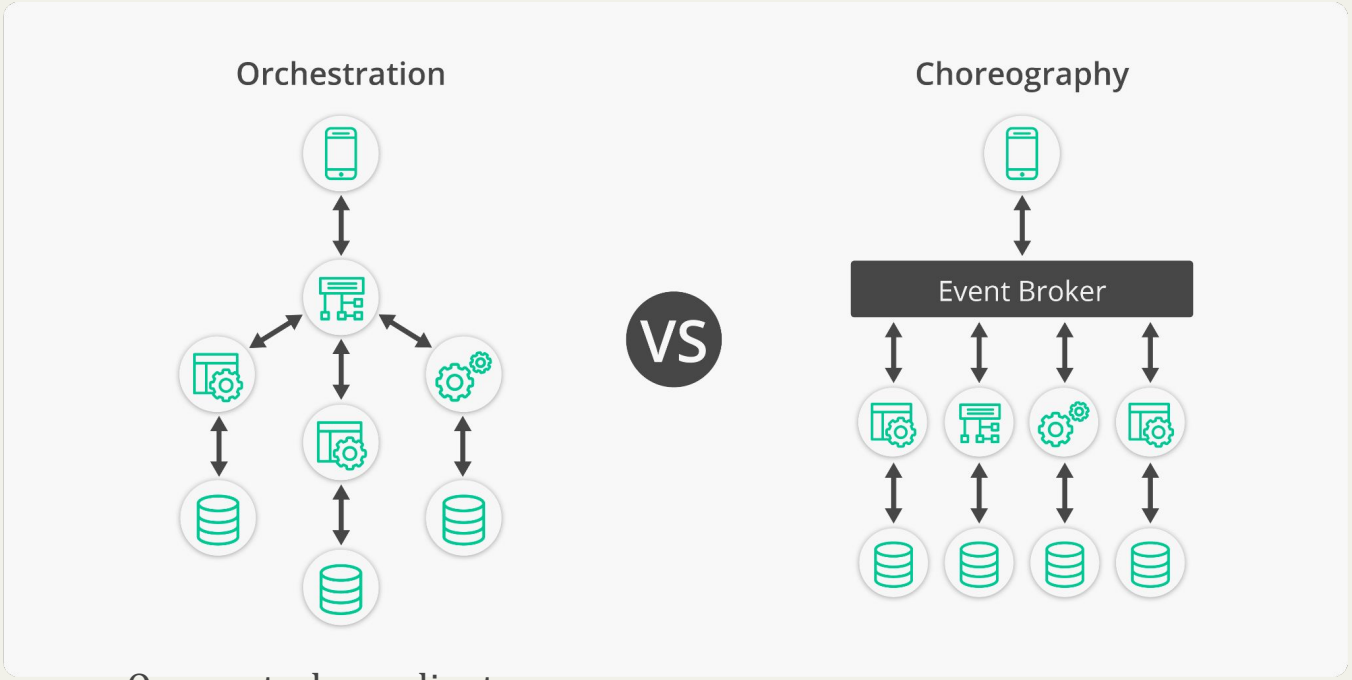
Reliable message delivery

Handles millions of messages

Keeps track of message history

Multiple consumers can receive the same message

Orchestration v/s Choreography



One central coordinator manages the flow

Everyone knows what to do

Source: solace.com

Orchestration v/s Choreography

- the orchestrator acts as a leader to send and receive info from each service before moving to the next service
- sequential process, one depends on the other

- does not have a middleman
- services can talk to each other independently

General advice: Prefer choreography over orchestration since it is more flexible, and cost of change is lesser

Key Takeaways

1. Docker makes software portable and consistent
2. Kafka enables reliable message delivery
3. Orchestration provides central control
4. Choreography allows direct communication
5. ML can make predictions and detect anomalies

What do we expect you to know?

Don’t worry, it’s simpler than it sounds!
All you need is:

A Basic Understanding of Machine Learning (ML):

- Familiarity with concepts like training and prediction.
- Ex: Knowing that ML models "learn" from data to make decisions.

A Basic Understanding of Python Programming:

- Reading and running Python scripts.
- Basic terminal commands (e.g., python script.py)
- If you don’t work with the Command prompt often, we do have the commands set, so don’t worry.

No Deep Knowledge Required. We’ll walk you through each step — from setting up tools to running the code.

➔ We will be doing the follow along tutorial on a Windows OS, but we have a TA to help you if you have a Linux or a Mac OS
➔ We expect your system has Python installed.

Orchestrator Demo

Cricket Score Predictor

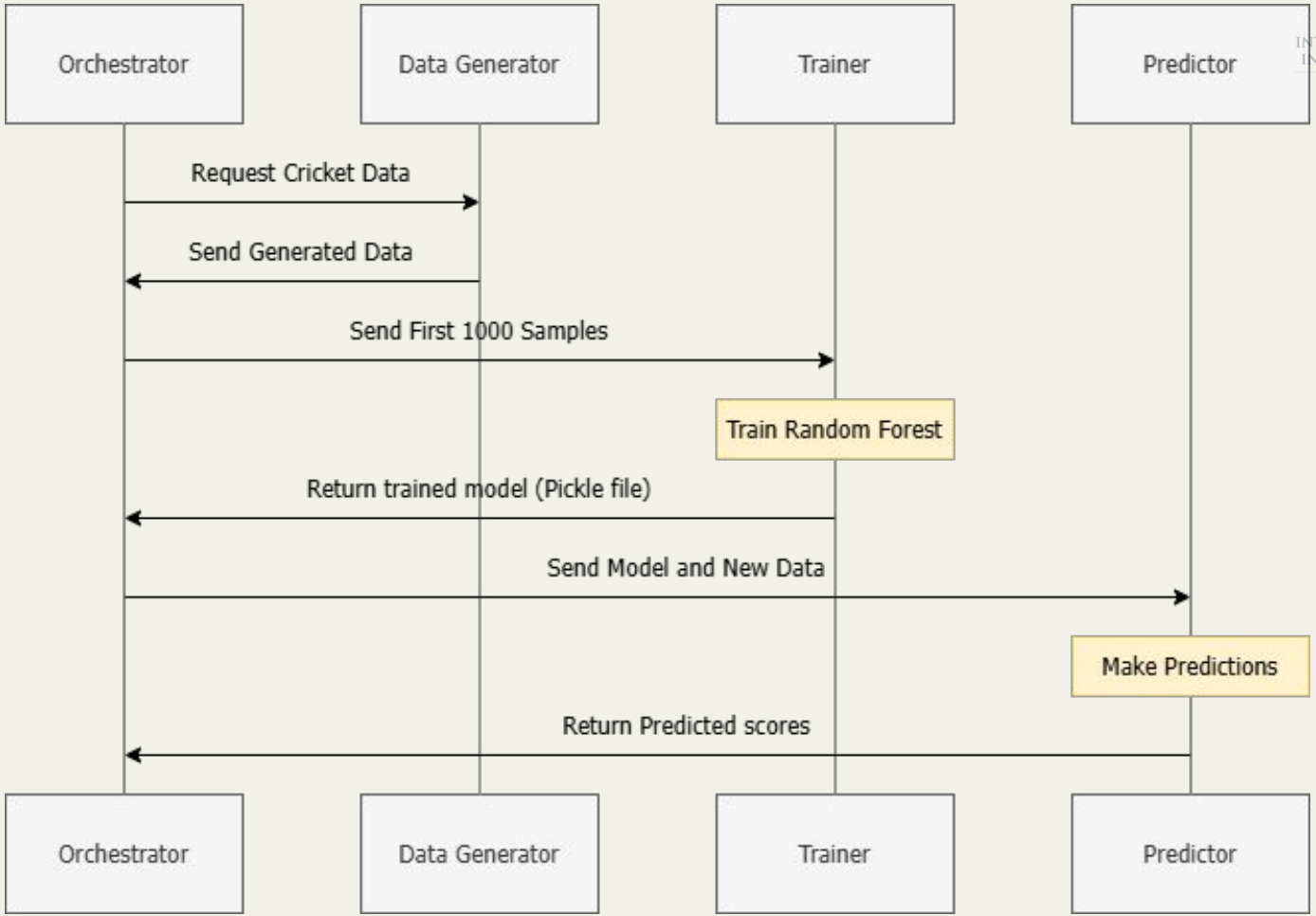
What will we see:

- How data flows through the system
- How the ML model learns
- How predictions are made
- How the orchestrator coordinates everything

Orchestrator Demo

Cricket Score Predictor

Kafka	Orchestrator	Data Generator	Trainer	Predictor
Broker	Committee Leader	Real-Time Data	Group of experts	Final Vote
Is the medium via which all the communication and data streaming takes place	Coordinates the entire process	Generates ball-by-ball cricket data	Learns patterns using Random Forest	Makes future predictions


Orchestrator
Data Generator
Trainer
Predictor
Request Cricket Data
Send Generated Data
Send First 1000 Samples
Train Random Forest
Return trained model (Pickle file)
Send Model and New Data
Make Predictions
Return Predicted scores
Orchestrator
Data Generator
Trainer
Predictor

Live Demo Instructions

It's not rocket science, we promise

Random Forest explained

Think of it as:

- A group of experts making decisions together
- Each expert looks at different aspects of the game
- They vote to make the final prediction
- More accurate than a single expert
- Unlike real committees, they decide quickly!

Data Generation and training


```python
def generate_cricket_data(num_samples=1000):
    data = []
    for _ in range(num_samples):
        current_over = random.uniform(5, 20)
        current_score = current_over * random.uniform(5, 10)  # avg 5-10 runs per over
        wickets = random.randint(0, 9)
        run_rate = current_score / current_over

        # Features that affect final score
        batting_strength = random.uniform(0.6, 1.0)
        pitch_condition = random.uniform(0.7, 1.0)

        # Calculate final score (target variable)
        remaining_overs = 20 - current_over
        wicket_factor = (10 - wickets) / 10
        final_score = current_score + (remaining_overs * run_rate * wicket_factor *
                                       batting_strength * pitch_condition)
        final_score = max(final_score * random.uniform(0.9, 1.1), current_score)  # Add noise
```

```python
def train_model(self):
    self.logger.info("Starting model training...")
    training_data = []

    # Collect training data
    for message in self.consumer:
        print("recieved: ", message.value)
        training_data.append(message.value)
        if len(training_data) >= 1000:  # Collect 1000 samples
            break

    if not training_data:
        self.logger.error("No training data received!")
        return

    self.logger.info(f"Collected {len(training_data)} training samples")
```

```python
# Save model and scaler
with open('cricket_model.pkl', 'wb') as f:
    pickle.dump((model, scaler), f)
```

Data Orchestration and Prediction

```python
class MatchOrchestrator:
    def __init__(self, bootstrap_servers):
        self.logger = setup_logger('MatchOrchestrator')
        self.producer = KafkaProducer(
            bootstrap_servers=bootstrap_servers,
            value_serializer=lambda v: json.dumps(v).encode('utf-8')
        )

    def start_training(self):
        self.logger.info("Generating and sending training data...")
        training_data = generate_cricket_data(1000)

        for data_point in training_data:
            print("Sending: ", data_point)
            self.producer.send('training-data', data_point)
        self.producer.flush()

        self.logger.info("Training data sent. Initiating model training...")

    def simulate_match(self, batting_strength=0.8, pitch_condition=0.85):
        self.logger.info("Starting match simulation...")
        current_score = 0
        wickets = 0
```

```python
def predict(self, match_state):
    features = np.array([[
        match_state['current_over'],
        match_state['current_score'],
        match_state['wickets'],
        match_state['run_rate'],
        match_state['batting_strength'],
        match_state['pitch_condition']
    ]])

    features_scaled = self.scaler.transform(features)
    prediction = self.model.predict(features_scaled)[0]

    return round(prediction)

def load_model(self):
    # Load model
    try:
        with open('cricket_model.pkl', 'rb') as f:
            self.model, self.scaler = pickle.load(f)
        self.logger.info("Model loaded successfully")
```

```python
def start(self):
    self.logger.info("Score predictor started. Waiting for match events...")
    for message in self.consumer:
        if not self.model_loaded:
            self.load_model()
        match_state = message.value
        predicted_score = self.predict(match_state)
```

```yaml
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    image: wurstmeister/kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_CREATE_TOPICS: "training-data:1:1,match-events:1:1,predictions:1:1,model-status:1:1"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - zookeeper
```

Partitions (1 per topic):

- Kafka will create **1 partition** for each topic.
- **Impact:** No parallelism within the topic, meaning all data resides in one partition, handled by a single broker. For simple setups.

Replication Factor (1 per topic):

- Each topic has a replication factor of **1**.
- **Impact:** No data redundancy; if the Kafka broker fails, data in these topics may be lost.

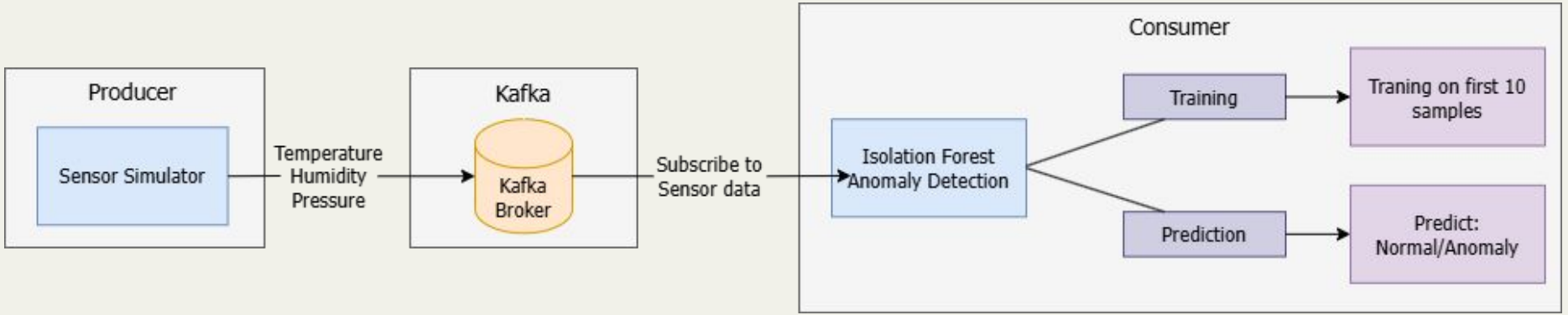
https://github.com/wurstmeister/kafka-docker - To explore more about Partitions

Choreographer Demo

Weather Monitor

Kafka	Producer	Consumer
Broker	**Sensor Simulator**	**Detector**
Is the medium via which all the communication and data streaming takes place	Produces weather data: Temperature, humidity, pressure	• Processes data using Isolation Forest • Learns normal patterns • Detects anomalies

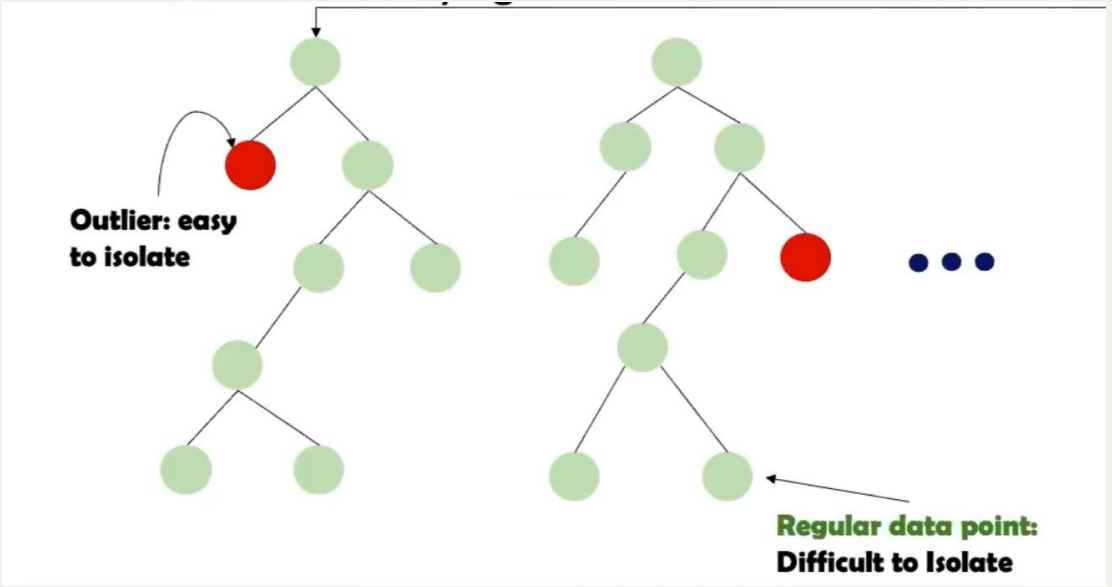
Weather Monitor - Workflow



Isolation Forest explained

How does it essentially work?

- Learns what "normal" looks like
- Spots anything unusual
- Like quality control in a factory



Source: https://www.youtube.com/watch?v=kN--TRv1UDY&t=360s&ab_channel=SixSigmaProSMART

Producer

```
def generate_sensor_data():
    return {
        'temperature': random.uniform(20, 30),
        'humidity': random.uniform(40, 80),
        'pressure': random.uniform(980, 1020)
    }

def main():
    print("Starting to send data")
    producer = KafkaProducer(
```

Consumer

```
for message in consumer:
    data = message.value
    print(data)
    buffer.append(data)

    if len(buffer) >= 10 and not detector.is_trained:
        detector.train(buffer)
        print("Model trained!")

    if detector.is_trained:
        prediction = detector.predict(data)
        status = "NORMAL" if prediction == 1 else "ANOMALY"
        print(f"Consumed: {data} - Status: {status}")
    else:
        print(f"Consumed: {data} - Collecting training data...")
```

Model

```
class AnomalyDetector:
    def __init__(self):
        self.model = IsolationForest(
            contamination=0.2,
            random_state=42
        )
        self.is_trained = False
        self.training_data = []

    def train(self, data):
        if len(data) >= 10:  # Train after collecting enough samples
            self.model.fit([[d['temperature'], d['humidity'], d['pressure']] for d in data])
            self.is_trained = True
```

```yaml
version: '3'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181

  kafka:
    image: wurstmeister/kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_CREATE_TOPICS: "sensor_data:1:1"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - zookeeper

  # producer:
  #   build: ./producer
  #   environment:
  #     KAFKA_BROKER: kafka:9092
  #   depends_on:
  #     - kafka

  # consumer:
  #   build: ./consumer
  #   environment:
  #     KAFKA_BROKER: kafka:9092
  #   depends_on:
  #     - kafka
```

This environment variable defines topics Kafka should create at startup. The syntax is:

```
KAFKA_CREATE_TOPICS: "sensor_data:1:1"
```

<topic_name>:<partitions>:<replication_factor>

- **sensor_data**: Name of the Kafka topic.
- **1**: Number of partitions for the topic.
- **1**: Replication factor (number of Kafka brokers storing copies of the data).

Partition Details

- The topic sensor_data has 1 partition.
- Since there's only one partition, all the data for this topic will reside in a single partition, and there will be no parallelism or distribution across multiple partitions.

https://github.com/wurstmeister/kafka-docker

To explore more about Partitions

Live Demo Instructions

Its not rocket science, we promise

Docker file code explanation

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY consumer.py .
COPY model.py .
CMD ["python", "consumer.py"]
```

Starts with a lightweight Python 3.9 environment.

Sets the working directory to /app inside the container.

Copies the dependencies list into the container.

Installs the required Python libraries.

Adds your application code (consumer logic and ML model).

Specifies the command to run your application when the container starts.

Prepares a containerized Python environment to run consumer.py with all its dependencies.

```
kafka-1      | [2024-12-17 11:41:27,867] INFO [GroupCoordinator 1001]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
kafka-1      | [2024-12-17 11:41:27,946] INFO [ProducerId Manager 1001]: Acquired new producerId block (brokerId:1001,blockStartProducerId:0,blockEndProducerId:999) by writing to Zk with path version 1 (kafka.coordinator.transaction.ProducerIdManager)
kafka-1      | [2024-12-17 11:41:27,946] INFO Updated cache from existing <empty> to latest FinalizedFeaturesAndEpoch(features=Features{}, epoch=0). (kafka.server.FinalizedFeatureCache)
kafka-1      | [2024-12-17 11:41:27,951] INFO [TransactionCoordinator id=1001] Starting up. (kafka.coordinator.transaction.TransactionCoordinator)
kafka-1      | [2024-12-17 11:41:27,961] INFO [TransactionCoordinator id=1001] Startup complete. (kafka.coordinator.transaction.TransactionCoordinator)
kafka-1      | [2024-12-17 11:41:27,965] INFO [Transaction Marker Channel Manager 1001]: Starting (kafka.coordinator.transaction.TransactionMarkerChannelManager)
kafka-1      | [2024-12-17 11:41:28,064] INFO [ExpirationReaper-1001-AlterAcls]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOperationReaper)
kafka-1      | [2024-12-17 11:41:28,163] INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotificationListener$ChangeEventProcessThread)
kafka-1      | [2024-12-17 11:41:28,191] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1001] Starting socket server acceptors and processors (kafka.network.SocketServer)
kafka-1      | [2024-12-17 11:41:28,214] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1001] Started data-plane acceptor and processor(s) for endpoint : ListenerName(PLAINTEXT) (kafka.network.SocketServer)
kafka-1      | [2024-12-17 11:41:28,218] INFO [SocketServer listenerType=ZK_BROKER, nodeId=1001] Started socket server acceptors and processors (kafka.network.SocketServer)
kafka-1      | [2024-12-17 11:41:28,232] INFO Kafka version: 2.8.1 (org.apache.kafka.common.utils.AppInfoParser)
kafka-1      | [2024-12-17 11:41:28,233] INFO Kafka commitId: 839b886f9b732b15 (org.apache.kafka.common.utils.AppInfoParser)
kafka-1      | [2024-12-17 11:41:28,233] INFO Kafka startTimeMs: 1734435688218 (org.apache.kafka.common.utils.AppInfoParser)
kafka-1      | [2024-12-17 11:41:28,243] INFO [KafkaServer id=1001] started (kafka.server.KafkaServer)
kafka-1      | [2024-12-17 11:41:28,524] INFO [broker-1001-to-controller-send-thread]: Recorded new controller, from now on will use broker localhost:9092 (id: 1001 rack: null) (kafka.server.BrokerToControllerRequestThread)
kafka-1      | creating topics: sensor_data:1:1
kafka-1      | WARNING: Due to limitations in metric names, topics with a period ('.') or underscore ('_') could collide. To avoid issues it is best to use either, but not both.
kafka-1      | Created topic sensor_data.
kafka-1      | [2024-12-17 11:41:36,180] INFO [ReplicaFetcherManager on broker 1001] Removed fetcher for partitions Set(sensor_data-0) (kafka.server.ReplicaFetcherManager)
kafka-1      | [2024-12-17 11:41:36,365] INFO [Log partition=sensor_data-0, dir=/kafka/kafka-logs-9573d2e1cc60] Loading producer state till offset 0 with message format version 2 (kafka.log.Log)
kafka-1      | [2024-12-17 11:41:36,388] INFO Created log for partition sensor_data-0 in /kafka/kafka-logs-9573d2e1cc60/sensor_data-0 with properties {} (kafka.log.LogManager)
kafka-1      | [2024-12-17 11:41:36,393] INFO [Partition sensor_data-0 broker=1001] No checkpointed highwatermark is found for partition sensor_data-0 (kafka.cluster.Partition)
kafka-1      | [2024-12-17 11:41:36,394] INFO [Partition sensor_data-0 broker=1001] Log loaded for partition sensor_data-0 with initial high watermark 0 (kafka.cluster.Partition)
```

Thank You!