Expert Architecture Training Program

Week 1 - Day 1: Kafka Lab Guide

Inspire Success, Your Goals & Opportunities

ISYGO Consulting Services

September 19, 2025

Contents

Introduction

This lab guide complements the Week 1-Day 1 Theory Guide. Participants will implement an **Event-Driven Architecture (EDA)** using **Apache Kafka**. The exercises cover **Kafka setup, topic creation, producers and consumers, fault tolerance, monitoring, logging, security, and multi-tenancy**, enabling participants to achieve all Day 1 objectives efficiently.

Astuce (Trick): Allocate at least 4GB RAM to Docker Desktop and ensure a stable internet connection to pull images quickly.

Objectives of the Lab

By the end of this lab session, participants will be able to:

1. Install and configure Kafka on Windows using Docker.
2. Create Kafka topics with multiple partitions for scalability.
3. Implement Java Kafka producers and consumers with error handling.
4. Understand Kafka consumer groups, offsets, and partition assignment.
5. Test event publishing and consumption with multiple consumers.
6. Apply monitoring and logging for Kafka events using tools like Kafdrop.
7. Demonstrate fault-tolerance scenarios with message persistence.
8. Configure Kafka security with SSL and SASL/PLAIN.
9. Implement multi-tenancy using tenant-specific topics and ACLs.

Chapter 1

Environment Setup

1.1 Prerequisites

- **Operating System**: Windows 10 or higher (build 19041 or later).
- **Java Development Kit**: JDK 17 (download from Oracle; verify with `java -version`).
- **Docker Desktop**: Version 4.10 or higher (download from Docker; enable WSL 2 backend).
- **IDE**: IntelliJ IDEA Community or Eclipse (download from JetBrains or Eclipse).
- **Command Line Tools**: PowerShell 7 or Windows Terminal (download from Microsoft).
- **Maven**: Version 3.8 or higher (download from Maven; verify with `mvn -version`).

1.1.1 Acceptance Criteria

- `java -version` confirms JDK 17.
- `docker -version` shows version 4.10 or higher.
- `mvn -version` confirms Maven 3.8 or higher.
- IDE is installed with a new Maven project template.

1.1.2 Troubleshooting

- **Issue:** Docker Desktop fails to start. **Solution:** Enable virtualization in BIOS, install WSL 2, restart Docker.
- **Issue:** JDK not recognized. **Solution:** Set `JAVA`$_{H}OMEtoJDKpathandupdate$`PATH`.
- `**Issue:**  Maven commands fail.  **Solution:**  Set M2`$_{H}OMEandaddMavenbinto$`PATH.`

Astuce (Trick): Run `docker info` to verify Docker setup; increase memory to 4GB in Docker settings for stability.

1.2 Kafka Setup Using Docker

1. Create a folder `C:YourUser>1d1-lab`.
2. Create `w1d1-lab/docker-compose.yml`:

```
version: '3.8'
services:
  zookeeper:
    image: wurstmeister/zookeeper:latest
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
  kafka:
    image: wurstmeister/kafka:latest
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_BROKER_ID: 1
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_MESSAGE_MAX_BYTES: 1000000
      KAFKA_NUM_PARTITIONS: 3
      KAFKA_MIN_INSYNC_REPLICAS: 1
      KAFKA_DEFAULT_REPLICATION_FACTOR: 1
    depends_on:
      - zookeeper
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
  kafdrop:
    image: obsidiandynamics/kafdrop:latest
    ports:
      - "9000:9000"
    environment:
      KAFKA_BROKERCONNECT: kafka:9092
    depends_on:
      - kafka
```

3. Open PowerShell in `w1d1-lab` (`cd  /Desktop/w1d1-lab`) and run:

```
docker-compose up -d
```

4. Verify containers:

```
docker ps
```

Look for `wurstmeister/kafka`, `wurstmeister/zookeeper`, and `obsidiandynamics/kafdrop`.

5. Access Kafdrop at `http://localhost:9000`.
6. (Advanced) Enable JMX for monitoring: - Add to Kafka environment: `KAFKA`$_JMX_OPTS:$ ”$-Dcom.sun.management.jmxremote-Dcom.sun.management.jmxremote.authenticate=$ $false$” $-Exposeport:$ `- "9999:9999"`.

`6. Stop environment:`

```
docker-compose down
```

1.2.1 Acceptance Criteria

- `docker-compose.yml` exists in `w1d1-lab`.
- `docker ps` lists Kafka, Zookeeper, and Kafdrop containers.
- `docker logs <kafka-container-id>` shows no errors.
- Kafdrop UI is accessible at `http://localhost:9000`.

1.2.2 Troubleshooting

- **Issue:** Kafka container exits. **Solution:** Check logs (`docker logs <kafka-container-id>`); free port 9092 (`netstat -ano | findstr 9092`).
- **Issue:** Zookeeper connection fails. **Solution:** Verify port 2181 (`telnet localhost 2181`).
- **Issue:** Kafdrop inaccessible. **Solution:** Ensure port 9000 is free; check `KAFKA`$_BROKERCON$

Chapter 2

Kafka Topics and Partitions

2.1 Creating Topics

1. Get Kafka container ID:

   ```
   docker ps
   ```

2. Access container shell:

   ```
   docker exec -it <kafka-container-id> bash
   ```

3. Create topic `orders`:

   ```
   bin/kafka-topics.sh --create --topic orders --bootstrap-server localhost:9092
       --partitions 3 --replication-factor 1
   ```

4. Verify:

   ```
   bin/kafka-topics.sh --list --bootstrap-server localhost:9092
   ```

5. (Advanced) Inspect topic:

   ```
   bin/kafka-topics.sh --describe --topic orders --bootstrap-server localhost:9092
   ```

6. (Advanced) Set retention:

   ```
   bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type topics
       --entity-name orders --alter --add-config retention.ms=86400000
   ```

2.1.1 Acceptance Criteria

- `orders` appears in `-list` output.
- `-describe` confirms 3 partitions, replication factor 1.
- Retention is 86400000 ms if advanced step applied.

2.1.2 Troubleshooting

- **Issue:** Topic creation fails. **Solution:** Check Kafka running; test `localhost:9092` (`telnet localhost 9092`).
- **Issue:** Topic not listed. **Solution:** Re-run creation; check syntax.
- **Issue:** Retention config error. **Solution:** Verify topic exists; correct command syntax.

Astuce (Trick): Increase partitions to 6 for high-throughput scenarios; use `-partitions` based on consumer count.

2.2 Understanding Partitions

Partitions enable **parallel processing** and **scalability**. Each partition is **ordered**, with **offsets** tracking consumption.

- **Use Case**: Consumer groups process partitions concurrently.
- **Best Practice**: Set partitions with $\lceil$events/sec/consumer capacity$\rceil$.
- **Advanced**: Use `compression.type=snappy` to reduce network load.

Chapter 3

Java Kafka Producer

3.1 Project Setup

1. Create Maven project in IDE: - IntelliJ: File > New > Project > Maven. - Eclipse: File > New > Maven Project.

2. Create `pom.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
            http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.isygo</groupId>
    <artifactId>kafka-lab</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>3.5.0</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>2.0.7</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.10.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

3. Build project:

```
mvn clean install
```

4. (Advanced) Add JUnit for testing:

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.9.2</version>
    <scope>test</scope>
</dependency>
```

3.1.1 Acceptance Criteria

- `mvn clean install` returns "BUILD SUCCESS".
- Kafka and SLF4J dependencies are resolved.
- IDE imports Kafka classes.

3.1.2 Troubleshooting

- **Issue:** Dependency download fails. **Solution:** Check internet; run `mvn -U clean install`.
- **Issue:** IDE cannot resolve classes. **Solution:** Refresh project (IntelliJ: Maven > Reload; Eclipse: Maven > Update).
- **Issue:** JDK version mismatch. **Solution:** Set JDK 17 in IDE and `pom.xml`.

Astuce (Trick): Use `mvn dependency:tree` to detect and resolve dependency conflicts.

3.2 Implementing a Simple Producer

1. Create `src/main/java/com/isygo/KafkaProducerExample.java`:

```
package com.isygo;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.Properties;

public class KafkaProducerExample {
   private static final Logger logger =
       LoggerFactory.getLogger(KafkaProducerExample.class);

   public static void main(String[] args) {
       Properties props = new Properties();
       props.put("bootstrap.servers", "localhost:9092");
       props.put("key.serializer",
           "org.apache.kafka.common.serialization.StringSerializer");
       props.put("value.serializer",
           "org.apache.kafka.common.serialization.StringSerializer");
       props.put("acks", "all");
       props.put("retries", 3);
       props.put("compression.type", "snappy");
       props.put("linger.ms", 5);

       try (KafkaProducer<String, String> producer = new
           KafkaProducer<>(props)) {
           for (int i = 1; i <= 10; i++) {
               ProducerRecord<String, String> record = new
                   ProducerRecord<>("orders", "order" + i, "Order data " + i);
               producer.send(record, (metadata, exception) -> {
                  if (exception != null) {
                      logger.error("Error sending message: {}",
                          exception.getMessage());
                  } else {
                      logger.info("Sent: key=order{}, partition={}, offset={}",
                          i, metadata.partition(), metadata.offset());
                  }
               });
           }
           producer.flush();
       } catch (Exception e) {
           logger.error("Producer error: {}", e.getMessage());
       }
   }
}
```

2. Run:

```
mvn exec:java -Dexec.mainClass="com.isygo.KafkaProducerExample"
```

3. (Advanced) Add partition key for consistent routing:

```
ProducerRecord<String, String> record = new ProducerRecord<>("orders",
    "customer1", "Order data " + i);
```

3.2.1 Acceptance Criteria

- Producer logs 10 messages sent.
- Messages appear in `orders` via Kafdrop.
- Logs show partition and offset details.

3.2.2 Troubleshooting

- **Issue:** Producer connection fails. **Solution:** Verify `localhost:9092`; check Kafka status.
- **Issue:** Messages not in topic. **Solution:** Confirm topic exists; use `bin/kafka-console-consum`
- **Issue:** No logs. **Solution:** Ensure SLF4J dependency; check console.

Astuce (Trick): Set `batch.size=16384` to optimize high-volume message batching.

3.3 Explanation

- `bootstrap.servers`: Broker address.
- `key.serializer/value.serializer`: Serialize data.
- `send()`: Publishes messages with callbacks.

Chapter 4

Java Kafka Consumer

4.1 Implementing a Simple Consumer

1. Create `src/main/java/com/isygo/KafkaConsumerExample.java`:

```
package com.isygo;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class KafkaConsumerExample {
    private static final Logger logger =
        LoggerFactory.getLogger(KafkaConsumerExample.class);

    public static void main(String[] args) {
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9092");
        props.put("group.id", "order-consumer-group");
        props.put("key.deserializer",
            "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
            "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("auto.offset.reset", "earliest");
        props.put("enable.auto.commit", "true");
        props.put("max.poll.records", 100);

        try (KafkaConsumer<String, String> consumer = new
            KafkaConsumer<>(props)) {
            consumer.subscribe(Collections.singletonList("orders"));
            while (true) {
                ConsumerRecords<String, String> records =
                    consumer.poll(Duration.ofMillis(100));
                for (ConsumerRecord<String, String> record : records) {
                    logger.info("Received: key={}, value={}, partition={},
                        offset={}",
                            record.key(), record.value(), record.partition(),
                            record.offset());
                }
            }
        } catch (Exception e) {
            logger.error("Consumer error: {}", e.getMessage());
        }
    }
}
```

2. Run:

```
mvn exec:java -Dexec.mainClass="com.isygo.KafkaConsumerExample"
```

3. (Advanced) Manual offset commit:

```
props.put("enable.auto.commit", "false");
consumer.commitSync();
```

4.1.1 Acceptance Criteria

- Consumer logs messages from `orders`.
- Console shows key, value, partition, offset.
- All 10 produced messages are processed.

4.1.2 Troubleshooting

- **Issue:** No messages received. **Solution:** Check topic messages; verify `auto.offset.reset`.
- **Issue:** Poll timeout. **Solution:** Increase `max.poll.interval.ms`; check broker.
- **Issue:** Duplicates. **Solution:** Enable `enable.auto.commit` or use manual commits.

Astuce (Trick): Set `max.poll.records=50` for memory efficiency in high-volume scenarios.

4.2 Explanation

- `group.id`: Enables load balancing in consumer group.
- `poll()`: Fetches messages with timeout.
- `auto.offset.reset`: Sets read start point.

Chapter 5

Advanced Lab Exercises

5.1 Multiple Consumers

1. Run two consumers:

   ```
   mvn exec:java -Dexec.mainClass="com.isygo.KafkaConsumerExample"
   ```

   Use separate terminals.

2. Produce messages using `KafkaProducerExample`.

3. Observe distribution in Kafdrop.

4. (Advanced) Check group status:

   ```
   bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe
       --group order-consumer-group
   ```

5.1.1 Acceptance Criteria

- Two consumers run with same `group.id`.
- Messages distributed across partitions.
- `-describe` shows partition assignments.

5.1.2 Troubleshooting

- **Issue:** Uneven distribution. **Solution:** Increase partitions; check group coordinator.
- **Issue:** Consumers not joining group. **Solution:** Verify `group.id`; check broker.

Astuce (Trick): Use `-reset-offsets` to restart consumer group for testing.

5.2 Fault-Tolerance Testing

1. Stop Kafka during producer run:
```
docker-compose stop kafka
```
2. Produce messages.
3. Restart Kafka:
```
docker-compose start kafka
```
4. Verify message persistence in Kafdrop.
5. Stop consumer, produce messages, restart, verify replay.
6. (Advanced) Increase replication:
```
bin/kafka-topics.sh --alter --topic orders --replication-factor 1
```

5.2.1 Acceptance Criteria

- Messages persist after restart.
- Consumer resumes from last offset.

5.2.2 Troubleshooting

- **Issue:** Messages lost. **Solution:** Check `acks=all`; verify retention.
- **Issue:** Consumer skips messages. **Solution:** Set `auto.offset.reset=earliest`.

Astuce (Trick): Increase `retention.bytes` for large datasets in fault tests.

5.3 Monitoring Kafka

1. Ensure Kafdrop in `docker-compose.yml`.
2. Restart Docker:
```
docker-compose down && docker-compose up -d
```
3. Access `http://localhost:9000`.
4. Monitor rates, lag, offsets.
5. (Advanced) Add Prometheus/Grafana: - Pull images: `prom/prometheus`, `grafana/grafana`. - Configure Kafka exporter.

5.3.1 Acceptance Criteria

- Kafdrop shows `orders` and messages.
- Lag is minimal (<10 messages).
- Prometheus/Grafana shows metrics (if advanced).

5.3.2 Troubleshooting

- **Issue:** Kafdrop inaccessible. **Solution:** Free port 9000; check `KAFKA`$_B ROKERCONNECT$.
- `**Issue:** High lag. **Solution:** Scale consumers; add partitions.`

Astuce (Trick): Use Grafana dashboards for real-time Kafka metric visualization.

Chapter 6

Kafka Security Configuration

6.1 Enabling SSL for Encryption

1. Create `w1d1-lab/certs`:

   ```
   mkdir -p w1d1-lab/certs
   ```

2. Generate certificates:

   ```
   openssl req -new -x509 -keyout w1d1-lab/certs/kafka.key -out
       w1d1-lab/certs/kafka.crt -days 365 -nodes -subj "/CN=localhost"
   ```

3. Convert to JKS:

   ```
   keytool -importcert -file w1d1-lab/certs/kafka.crt -keystore
       w1d1-lab/certs/kafka.truststore.jks -storepass changeit -noprompt
   openssl pkcs12 -export -in w1d1-lab/certs/kafka.crt -inkey
       w1d1-lab/certs/kafka.key -out w1d1-lab/certs/kafka.p12 -name kafka -passout
       pass:changeit
   keytool -importkeystore -srckeystore w1d1-lab/certs/kafka.p12 -srcstoretype
       PKCS12 -destkeystore w1d1-lab/certs/kafka.keystore.jks -deststorepass
       changeit -srcstorepass changeit
   ```

4. Update `docker-compose.yml`:

   ```
   version: '3.8'
   services:
     zookeeper:
       image: wurstmeister/zookeeper:latest
       ports:
         - "2181:2181"
     kafka:
       image: wurstmeister/kafka:latest
       ports:
         - "9092:9092"
         - "9093:9093"
       environment:
         KAFKA_ADVERTISED_HOST_NAME: localhost
         KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
         KAFKA_LISTENERS: PLAINTEXT://:9092,SSL://:9093
         KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092,SSL://localhost:9093
   ```

```
      KAFKA_SSL_KEYSTORE_LOCATION: /etc/kafka/certs/kafka.keystore.jks
      KAFKA_SSL_KEYSTORE_PASSWORD: changeit
      KAFKA_SSL_KEY_PASSWORD: changeit
      KAFKA_SSL_TRUSTSTORE_LOCATION: /etc/kafka/certs/kafka.truststore.jks
      KAFKA_SSL_TRUSTSTORE_PASSWORD: changeit
    volumes:
      - ./certs:/etc/kafka/certs
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - zookeeper
  kafdrop:
    image: obsidiandynamics/kafdrop:latest
    ports:
      - "9000:9000"
    environment:
      KAFKA_BROKERCONNECT: kafka:9092
    depends_on:
      - kafka
```

5. Update `KafkaProducerExample.java` and `KafkaConsumerExample.java`:

```
props.put("bootstrap.servers", "localhost:9093");
props.put("security.protocol", "SSL");
props.put("ssl.keystore.location",
    "C:/Users/<YourUser>/Desktop/w1d1-lab/certs/kafka.keystore.jks");
props.put("ssl.keystore.password", "changeit");
props.put("ssl.key.password", "changeit");
props.put("ssl.truststore.location",
    "C:/Users/<YourUser>/Desktop/w1d1-lab/certs/kafka.truststore.jks");
props.put("ssl.truststore.password", "changeit");
```

6. Restart Docker:

```
docker-compose down && docker-compose up -d
```

6.1.1 Acceptance Criteria

- Certificates in `w1d1-lab/certs`.
- Kafka listens on 9093 with SSL.
- Producer/consumer connect via SSL.

6.1.2 Troubleshooting

- **Issue:** SSL handshake fails. **Solution:** Check certificate paths; ensure `CN=localhost`.
- **Issue:** Keytool errors. **Solution:** Verify JDK bin in `PATH`.

6.2 Enabling SASL/PLAIN Authentication

1. Create `w1d1-lab/kafka-jaas.conf`:

```
KafkaServer {
   org.apache.kafka.common.security.plain.PlainLoginModule required
   username="admin"
   password="admin-secret"
   user_admin="admin-secret"
   user_client="client-secret"
   user_tenant1="tenant1-secret"
   user_tenant2="tenant2-secret";
};
```

2. Update `docker-compose.yml`:

```
version: '3.8'
services:
  zookeeper:
    image: wurstmeister/zookeeper:latest
    ports:
      - "2181:2181"
  kafka:
    image: wurstmeister/kafka:latest
    ports:
      - "9092:9092"
      - "9093:9093"
      - "9094:9094"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: localhost
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENERS: PLAINTEXT://:9092,SSL://:9093,SASL_SSL://:9094
      KAFKA_ADVERTISED_LISTENERS:
          PLAINTEXT://localhost:9092,SSL://localhost:9093,SASL_SSL://localhost:9094
      KAFKA_SSL_KEYSTORE_LOCATION: /etc/kafka/certs/kafka.keystore.jks
      KAFKA_SSL_KEYSTORE_PASSWORD: changeit
      KAFKA_SSL_KEY_PASSWORD: changeit
      KAFKA_SSL_TRUSTSTORE_LOCATION: /etc/kafka/certs/kafka.truststore.jks
      KAFKA_SSL_TRUSTSTORE_PASSWORD: changeit
      KAFKA_AUTHORIZER_CLASS_NAME: kafka.security.authorizer.AclAuthorizer
      KAFKA_ALLOW_EVERYONE_IF_NO_ACL_FOUND: "false"
      KAFKA_SASL_ENABLED_MECHANISMS: PLAIN
      KAFKA_SASL_MECHANISM_INTER_BROKER_PROTOCOL: PLAIN
      KAFKA_OPTS: "-Djava.security.auth.login.config=/etc/kafka/kafka-jaas.conf"
    volumes:
      - ./certs:/etc/kafka/certs
      - ./kafka-jaas.conf:/etc/kafka/kafka-jaas.conf
      - /var/run/docker.sock:/var/run/docker.sock
    depends_on:
      - zookeeper
  kafdrop:
    image: obsidiandynamics/kafdrop:latest
    ports:
      - "9000:9000"
    environment:
      KAFKA_BROKERCONNECT: kafka:9092
    depends_on:
      - kafka
```

3. Update `KafkaProducerExample.java` and `KafkaConsumerExample.java`:

```
props.put("bootstrap.servers", "localhost:9094");
props.put("security.protocol", "SASL_SSL");
props.put("sasl.mechanism", "PLAIN");
props.put("sasl.jaas.config",
    "org.apache.kafka.common.security.plain.PlainLoginModule required
    username=\"client\" password=\"client-secret\";");
```

4. Restart Docker:

```
docker-compose down && docker-compose up -d
```

6.2.1 Acceptance Criteria

- Kafka listens on 9094 with SASL/PLAIN over SSL.
- Producer/consumer authenticate with `client/client-secret`.
- Unauthorized clients rejected.

6.2.2 Troubleshooting

- **Issue:** Authentication fails. **Solution:** Verify JAAS credentials match client config.
- **Issue:** SASL not enabled. **Solution:** Check `KAFKA`$_SASL_ENABLED_MECHANISMS$.

Astuce (Trick): Use environment variables for JAAS credentials in production.

6.3 Basic Authorization with ACLs

1. Access Kafka shell:

```
docker exec -it <kafka-container-id> bash
```

2. Add ACL for `client`:

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 \
--add --allow-principal User:client --operation Read --operation Write --topic
    orders
```

3. Verify ACLs:

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181
    --list --topic orders
```

4. (Advanced) Restrict by host:

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 \
--add --allow-principal User:client --operation Read --operation Write --topic
    orders --allow-host 127.0.0.1
```

6.3.1 Acceptance Criteria

- `client` can read/write `orders`.
- ACLs listed correctly.
- Unauthorized users denied.

6.3.2 Troubleshooting

- **Issue:** ACL command fails. **Solution:** Verify `AclAuthorizer`; check Zookeeper.
- **Issue:** Client access denied. **Solution:** Check ACL principal and permissions.

Astuce (Trick): Use `-allow-host` for IP-based restrictions in production.

Chapter 7

Kafka Multi-Tenancy Configuration

7.1 Creating Tenant-Specific Topics

1. Access Kafka shell:

   ```
   docker exec -it <kafka-container-id> bash
   ```

2. Create tenant topics:

   ```
   bin/kafka-topics.sh --create --topic tenant1.orders --bootstrap-server
       localhost:9092 --partitions 3 --replication-factor 1
   bin/kafka-topics.sh --create --topic tenant2.orders --bootstrap-server
       localhost:9092 --partitions 3 --replication-factor 1
   ```

3. Verify:

   ```
   bin/kafka-topics.sh --list --bootstrap-server localhost:9092
   ```

4. (Advanced) Set retention:

   ```
   bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type topics
       --entity-name tenant1.orders --alter --add-config retention.ms=86400000
   bin/kafka-configs.sh --bootstrap-server localhost:9092 --entity-type topics
       --entity-name tenant2.orders --alter --add-config retention.ms=86400000
   ```

7.1.1 Acceptance Criteria

- `tenant1.orders`, `tenant2.orders` listed.
- Each has 3 partitions (`-describe`).
- Retention set to 24 hours (if advanced).

7.1.2 Troubleshooting

- **Issue:** Topic creation fails. **Solution:** Verify Kafka running; check topic syntax.
- **Issue:** Topics not listed. **Solution:** Re-run creation; verify with `-list`.

Astuce (Trick): Use prefixes (e.g., `tenant1.`) for namespace organization.

7.2 Configuring ACLs for Tenant Isolation

1. Update `w1d1-lab/kafka-jaas.conf`:

```
KafkaServer {
   org.apache.kafka.common.security.plain.PlainLoginModule required
   username="admin"
   password="admin-secret"
   user_admin="admin-secret"
   user_client="client-secret"
   user_tenant1="tenant1-secret"
   user_tenant2="tenant2-secret";
};
```

2. Add tenant ACLs:

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 \
--add --allow-principal User:tenant1 --operation Read --operation Write --topic
    tenant1.orders
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 \
--add --allow-principal User:tenant2 --operation Read --operation Write --topic
    tenant2.orders
```

3. Verify:

```
bin/kafka-acls.sh --authorizer-properties zookeeper.connect=localhost:2181 --list
```

7.2.1 Acceptance Criteria

- `tenant1` accesses only `tenant1.orders`, `tenant2` only `tenant2.orders`.
- ACLs listed correctly.

7.2.2 Troubleshooting

- **Issue:** Tenant access denied. **Solution:** Verify JAAS credentials and ACLs.
- **Issue:** Cross-tenant access. **Solution:** Ensure `KAFKA`$_ALLOW_EVERYONE_IF_NO_ACL_FOUNL$
  $false$.

Astuce (Trick): Use topic patterns (e.g., `tenant*`) for scalable ACL management.

7.3 Testing Multi-Tenant Producers and Consumers

1. Create `src/main/java/com/isygo/KafkaTenantProducer.java`:

```
package com.isygo;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.Properties;

public class KafkaTenantProducer {
    private static final Logger logger =
        LoggerFactory.getLogger(KafkaTenantProducer.class);

    public static void main(String[] args) {
        String tenant = args.length > 0 ? args[0] : "tenant1";
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9094");
        props.put("key.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer",
            "org.apache.kafka.common.serialization.StringSerializer");
        props.put("security.protocol", "SASL_SSL");
        props.put("sasl.mechanism", "PLAIN");
        props.put("sasl.jaas.config",
            "org.apache.kafka.common.security.plain.PlainLoginModule required
            username=\"" + tenant + "\" password=\"" + tenant + "-secret\";");
        props.put("ssl.keystore.location",
            "C:/Users/<YourUser>/Desktop/w1d1-lab/certs/kafka.keystore.jks");
        props.put("ssl.keystore.password", "changeit");
        props.put("ssl.key.password", "changeit");
        props.put("ssl.truststore.location",
            "C:/Users/<YourUser>/Desktop/w1d1-lab/certs/kafka.truststore.jks");
        props.put("ssl.truststore.password", "changeit");
        props.put("acks", "all");
        props.put("retries", 3);
        props.put("compression.type", "snappy");

        try (KafkaProducer<String, String> producer = new
            KafkaProducer<>(props)) {
            for (int i = 1; i <= 5; i++) {
                ProducerRecord<String, String> record = new
                    ProducerRecord<>(tenant + ".orders", "order" + i, tenant + "
                    Order data " + i);
                producer.send(record, (metadata, exception) -> {
                    if (exception != null) {
                        logger.error("Error sending: {}", exception.getMessage());
                    } else {
                        logger.info("Sent to {}.orders: key=order{}, partition={},
                            offset={}", tenant, i, metadata.partition(),
                            metadata.offset());
                    }
                });
            }
            producer.flush();
        } catch (Exception e) {
            logger.error("Producer error: {}", e.getMessage());
        }
    }
}
```

2. Create `src/main/java/com/isygo/KafkaTenantConsumer.java`:

```
package com.isygo;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.time.Duration;
import java.util.Collections;
import java.util.Properties;

public class KafkaTenantConsumer {
    private static final Logger logger =
        LoggerFactory.getLogger(KafkaTenantConsumer.class);

    public static void main(String[] args) {
        String tenant = args.length > 0 ? args[0] : "tenant1";
        Properties props = new Properties();
        props.put("bootstrap.servers", "localhost:9094");
        props.put("group.id", tenant + "-consumer-group");
        props.put("key.deserializer",
            "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("value.deserializer",
            "org.apache.kafka.common.serialization.StringDeserializer");
        props.put("auto.offset.reset", "earliest");
        props.put("security.protocol", "SASL_SSL");
        props.put("sasl.mechanism", "PLAIN");
        props.put("sasl.jaas.config",
            "org.apache.kafka.common.security.plain.PlainLoginModule required
            username=\"" + tenant + "\" password=\"" + tenant + "-secret\";");
        props.put("ssl.keystore.location",
            "C:/Users/<YourUser>/Desktop/w1d1-lab/certs/kafka.keystore.jks");
        props.put("ssl.keystore.password", "changeit");
        props.put("ssl.key.password", "changeit");
        props.put("ssl.truststore.location",
            "C:/Users/<YourUser>/Desktop/w1d1-lab/certs/kafka.truststore.jks");
        props.put("ssl.truststore.password", "changeit");

        try (KafkaConsumer<String, String> consumer = new
            KafkaConsumer<>(props)) {
            consumer.subscribe(Collections.singletonList(tenant + ".orders"));
            while (true) {
                ConsumerRecords<String, String> records =
                    consumer.poll(Duration.ofMillis(100));
                for (ConsumerRecord<String, String> record : records) {
                    logger.info("Received from {}.orders: key={}, value={},
                        partition={}, offset={}",
                           tenant, record.key(), record.value(),
                               record.partition(), record.offset());
                }
            }
        } catch (Exception e) {
            logger.error("Consumer error: {}", e.getMessage());
        }
    }
}
```

3. Run `tenant1` producer:

```
mvn exec:java -Dexec.mainClass="com.isygo.KafkaTenantProducer"
    -Dexec.args="tenant1"
```

4. Run `tenant1` consumer:

```
mvn exec:java -Dexec.mainClass="com.isygo.KafkaTenantConsumer"
    -Dexec.args="tenant1"
```

5. Repeat for `tenant2`.
6. Verify isolation in Kafdrop.

7.3.1 Acceptance Criteria

- `tenant1` accesses only `tenant1.orders`.
- `tenant2` accesses only `tenant2.orders`.
- Kafdrop shows tenant-specific messages.

7.3.2 Troubleshooting

- **Issue:** Cross-tenant access. **Solution:** Verify ACLs and subscriptions.
- **Issue:** Authentication errors. **Solution:** Check JAAS credentials.

Astuce (Trick): Test cross-tenant access to ensure ACL enforcement.

Reflection Questions

1. How does Kafka ensure message durability?
2. What happens if two consumers share the same partition?
3. How would you scale Kafka for high-volume order processing?
4. How does partitioning improve parallelism?
5. Why is SSL critical for securing Kafka in production?
6. How do ACLs enhance access control in multi-tenant Kafka setups?

Glossary

- **ACL (Access Control List):** Defines permissions for Kafka users/resources.
- **Consumer Group:** Consumers sharing `group.id` for load balancing.
- **Event-Driven Architecture (EDA):** Architecture using events for communication.
- **Kafka:** Distributed streaming platform for high-throughput events.
- **Multi-Tenancy:** Sharing infrastructure with isolated tenant data.
- **Namespace:** Logical grouping of topics (e.g., `tenant1.`).
- **Offset:** Message position identifier in a partition.
- **Partition:** Topic subset for ordered, parallel processing.
- **SASL (Simple Authentication and Security Layer):** Authentication framework for Kafka.
- **SSL (Secure Sockets Layer):** Encryption protocol for Kafka communication.
- **Tenant:** A client or organization with isolated data in a shared system.
- **Topic:** Logical channel for grouping events.

References

- Apache Kafka Documentationkafka2025
- Apache Kafka Security Documentationkafka2025
- Baeldung Kafka Java Guide
- Kafka Quickstart Guide
- Confluent: Kafka Security Best Practices
- Microsoft Learn: Multi-Tenancy in Event Hubs