# sudo dnf remove -y kafka

**clean reinstall of Kafka 3.8 with ZooKeeper** on **CentOS Stream 9 (aka your CentOS 10)**, using only **`sudo` commands**.

# 1a. Install `wget`

```
sudo dnf install -y wget
```

(If you're on CentOS 7, use `yum` instead of `dnf`.)

# 1b. Install Java 21 (OpenJDK)

On RHEL/CentOS 9 or newer:

```
sudo dnf install -y java-21-openjdk java-21-openjdk-devel
```

# 1c. Verify Java installation

```
java -version
```

Expected output (something like):

```
openjdk version "21.0.2" 2024-01-16
OpenJDK Runtime Environment (Red_Hat-21.0.2+12)
OpenJDK 64-Bit Server VM (build 21.0.2+12, mixed mode,
sharing)
```

# Step 2: Create Kafka directory

```
sudo mkdir -p /opt/kafka
cd /opt/kafka
```

# Step 3: Download Kafka 3.8.0

```
sudo wget https://downloads.apache.org/kafka/3.8.0/
kafka_2.13-3.8.0.tgz -O kafka.tgz
sudo tar -xvzf kafka.tgz --strip 1 -C /opt/kafka
sudo rm -f kafka.tgz
```

1. **Set appropriate permissions:**

   ```
   sudo useradd -r -m -s /sbin/nologin kafka
   ```

2. `sudo chown -R kafka:kafka /opt/kafka`

3. **Start ZooKeeper:**

   ```
   sudo -u kafka /opt/kafka/bin/zookeeper-server-start.sh
   -daemon /opt/kafka/config/zookeeper.properties
   ```

4. **Start Kafka Broker:**

   ```
   sudo -u kafka /opt/kafka/bin/kafka-server-start.sh
   -daemon /opt/kafka/config/server.properties
   ```

5. **Verify Kafka is running:**

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

6. **Create a test topic:**

   ```
   sudo -u kafka /opt/kafka/bin/kafka-topics.sh --create --
   topic test --bootstrap-server localhost:9092 --
   ```

```
partitions 1 --replication-factor 1
```

7. **Produce a message to the test topic:**

```
sudo -u kafka /opt/kafka/bin/kafka-console-producer.sh
--broker-list localhost:9092 --topic test


Type a message and press Enter.
```

8. **Consume the message from the test topic:**

```
sudo -u kafka /opt/kafka/bin/kafka-console-consumer.sh
--bootstrap-server localhost:9092 --topic test –from-
beginning
```

## Stop Zookeeper and Kafka Broker

```
sudo pkill -f kafka
sudo pkill -f zookeeper
```

### Configure Kafka broker

Edit `config/server.properties`:

## Key parameters:

```
broker.id=0                          # unique ID for each broker
listeners=PLAINTEXT://:9092    # Kafka broker port
log.dirs=/tmp/kafka-logs       # log storage directory
zookeeper.connect=localhost:2181  # Zookeeper connection
```

- For **multiple brokers**, change `broker.id` to 1, 2, 3… and `log.dirs` to unique directories per broker.
- Update `zookeeper.connect` to include all Zookeeper nodes if using a cluster.

### Start Kafka broker

```
sudo -u kafka /opt/kafka/bin/kafka-server-start.sh config/
server.properties
```

- For **multiple brokers**, repeat with each `server.properties` file (different `broker.id` and `log.dirs`)

## Verifying a Kafka Cluster

### Step 1: Check broker status

```
bin/kafka-broker-api-versions.sh --bootstrap-server
localhost:9092
```

- Should list the broker's API versions.

### Step 2: Create a test topic

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

- For a multi-broker cluster, replication factor can be >1.

# 1. Create a topic with partitions & replication

Example: **topic with 3 partitions and replication factor 2**

```
sudo /opt/kafka/bin/kafka-topics.sh \
  --create \
  --topic test-topic1 \
  --partitions 3 \
  --replication-factor 2 \
  --bootstrap-server localhost:9092
```

# 2. Describe a topic (to check partitions & replication)

```
sudo /opt/kafka/bin/kafka-topics.sh \
```

```
  --describe \
  --topic test-topic1 \
  --bootstrap-server localhost:9092
```

You'll see output like:

```
Topic: test-topic1   PartitionCount: 3   ReplicationFactor: 2
    Partition: 0    Leader: 1    Replicas: 1,2    Isr: 1,2
    Partition: 1    Leader: 2    Replicas: 2,1    Isr: 2,1
    Partition: 2    Leader: 1    Replicas: 1,2    Isr: 1,2
```

# 3. Increase partitions for an existing topic

Example: increase `test-topic` from 3 → 5 partitions:

```
sudo /opt/kafka/bin/kafka-topics.sh \
  --alter \
  --topic test-topic1 \
  --partitions 5 \
  --bootstrap-server localhost:9092
```

You can **only increase partitions**, never decrease.

# 4. Change replication factor (advanced)

Kafka doesn't allow changing replication factor with a single flag.
You need to use a **replica reassignment JSON**.

1. Generate current assignment:

```
sudo /opt/kafka/bin/kafka-reassign-partitions.sh \
  --bootstrap-server localhost:9092 \
  --topics-to-move-json-file <(echo '{"topics":
[{"topic":"test-topic"}],"version":1}') \
  --broker-list "1,2,3" \
  --generate
```

2.

3. Save the JSON output (e.g., `/tmp/reassign.json`), edit `replicas` to include more brokers.
4. Apply reassignment:

```
sudo /opt/kafka/bin/kafka-reassign-partitions.sh \
  --bootstrap-server localhost:9092 \
  --reassignment-json-file /tmp/reassign.json \
  --execute
```

# 5. Verify replication changes

```
sudo /opt/kafka/bin/kafka-topics.sh \
  --describe \
  --topic test-topic \
  --bootstrap-server localhost:9092
```

With this, you can:

- Create topics with any partitions + replication factor
- Increase partitions dynamically
- Change replication factor using reassignment

# Enable PostgreSQL module

```
sudo dnf module list postgresql
```
Choose **postgresql:15** (latest supported on CentOS Stream 9):

```
sudo dnf module enable postgresql:15 -y
```

# 2. Install PostgreSQL server and client

```
sudo dnf install -y postgresql-server postgresql-contrib
postgresql-devel
```

# 3. Initialize the database

```
sudo postgresql-setup --initdb
```

# 4. Start and enable PostgreSQL service

```
sudo systemctl enable postgresql
sudo systemctl start postgresql
```

Check status:

```
sudo systemctl status postgresql
```

# 5. Configure PostgreSQL user and database

Switch to the `postgres` user:

```
sudo -i -u postgres
```

Create a database and user for Kafka Connect:

```
psql
```

Inside `psql`:

```
CREATE DATABASE kafkadb;
CREATE USER kafkauser WITH ENCRYPTED PASSWORD 'kafkapass';
GRANT ALL PRIVILEGES ON DATABASE kafkadb TO kafkauser;
\q
```

Exit `postgres` user:

```
exit
```

# 6. Allow local connections (optional)

Edit `pg_hba.conf` if needed:

```
sudo nano /var/lib/pgsql/data/pg_hba.conf
```

Ensure the line for local connections is:

```
host    all      all             127.0.0.1/32            md5
```

Then reload PostgreSQL:

```
sudo systemctl reload postgresql
```

# 7. Test connection

```
psql -U kafkauser -d kafkadb -h localhost -W
```

Enter password `kafkapass`.

You should connect successfully.

## Switch to postgres user

```
sudo -i -u postgres
```

## Connect to your database

```
psql -d kafkadb
```

## Create the `users` table

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    firstname VARCHAR(50),
    lastname VARCHAR(50),
```

```
    email VARCHAR(100),
    phone VARCHAR(20),
    active BOOLEAN
);
```

## Insert sample data

```
INSERT INTO users (firstname, lastname, email, phone, active)
VALUES
('John', 'Doe', 'john@example.com', '1234567890', TRUE),
('Jane', 'Smith', 'jane@example.com', '0987654321', TRUE);
```

## Verify table content

```
SELECT * FROM users;
```

Expected output:

```
 id | firstname | lastname |       email        |    phone
| active
----+-----------+---------+-----------------+------------
+--------
  1 | John      | Doe     | john@example.com | 1234567890 | t
  2 | Jane      | Smith   | jane@example.com | 0987654321 | t
```

## Exit Postgres

```
\q
exit
```

## Create the Kafka topic

```
sudo /opt/kafka/bin/kafka-topics.sh \
  --create \
  --topic postgres-users \
  --partitions 3 \
  --replication-factor 1 \
  --bootstrap-server localhost:9092
```

- **`--partitions 3`** → topic will have 3 partitions

- **`--replication-factor 1`** → only one broker (adjust if multiple brokers exist)

## List all topics to confirm

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

You should see:

```
postgres-users
```

## Describe the topic (check partitions & replication)

```
sudo /opt/kafka/bin/kafka-topics.sh \
  --describe \
  --topic postgres-users \
  --bootstrap-server localhost:9092
```

Expected output:

```
Topic: postgres-users   PartitionCount: 3
ReplicationFactor: 1
    Partition: 0    Leader: 1    Replicas: 1    Isr: 1
    Partition: 1    Leader: 1    Replicas: 1    Isr: 1
    Partition: 2    Leader: 1    Replicas: 1    Isr: 1
```

## Test producing a message to the topic

```
echo '{"id":1,"name":"John Doe","email":"john@example.com"}'
| \
sudo /opt/kafka/bin/kafka-console-producer.sh \
  --topic postgres-users \
  --bootstrap-server localhost:9092
```

## Test consuming messages from the topic

```
sudo /opt/kafka/bin/kafka-console-consumer.sh \
  --bootstrap-server localhost:9092 \
  --topic postgres-users \
  --from-beginning \
  --timeout-ms 5000
```

You should see:

```
{"id":1,"name":"John Doe","email":"john@example.com"}
```

# 4. Install JDBC Connector

```
cd /opt/kafka/libs
sudo wget https://repo1.maven.org/maven2/io/confluent/kafka-
connect-jdbc/10.7.5/kafka-connect-jdbc-10.7.5.jar
sudo wget https://jdbc.postgresql.org/download/
postgresql-42.7.3.jar
```

# 5. Configure Kafka Connect Distributed Worker

```
sudo tee /opt/kafka/config/connect-distributed.properties > /
dev/null <<'EOF'
bootstrap.servers=localhost:9092
group.id=connect-cluster
config.storage.topic=connect-configs
offset.storage.topic=connect-offsets
status.storage.topic=connect-status
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=true
plugin.path=/opt/kafka/libs
EOF
```

# 6. Create Internal Topics (required for distributed mode)

```
# Configs topic
sudo /opt/kafka/bin/kafka-topics.sh --create --topic connect-
configs --partitions 1 --replication-factor 1 --bootstrap-
server localhost:9092

# Offsets topic
sudo /opt/kafka/bin/kafka-topics.sh --create --topic connect-
offsets --partitions 1 --replication-factor 1 --bootstrap-
server localhost:9092

# Status topic
sudo /opt/kafka/bin/kafka-topics.sh --create --topic connect-
status --partitions 1 --replication-factor 1 --bootstrap-
server localhost:9092
```

# 7. Start Kafka Connect Distributed

```
cd /opt/kafka
sudo bin/connect-distributed.sh config/connect-
distributed.properties > /tmp/connect.log 2>&1 &
```

Check logs:

```
tail -f /tmp/connect.log
```

# Create the connector configuration

```
sudo tee /opt/kafka/config/postgres-source.json > /dev/null
<<'EOF'
{
  "name": "postgres-source",
  "config": {
```

```
    "connector.class":
"io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:postgresql://localhost:5432/
kafkadb",
    "connection.user": "kafkauser",
    "connection.password": "kafkapass",
    "mode": "incrementing",
    "incrementing.column.name": "id",
    "topic.prefix": "postgres-",
    "table.whitelist": "users",
    "poll.interval.ms": "5000"
  }
}
EOF
```

**Explanation of important fields:**

- `connector.class` → JDBC source connector class
- `connection.url` → JDBC URL to your PostgreSQL DB
- `mode=incrementing` → Connector uses the `id` column to track new rows
- `topic.prefix=postgres-` → Kafka topic name will be `postgres-users`
- `table.whitelist=users` → Only read from `users` table
- `poll.interval.ms=5000` → Check for new rows every 5 seconds

# Register the connector with Kafka Connect

Assuming your Kafka Connect **distributed mode REST API** is running on `http://localhost:8083`:

```
sudo curl -X POST -H "Content-Type: application/json" \
  --data @/opt/kafka/config/postgres-source.json \
  http://localhost:8083/connectors
```

Expected response:

```
{"name":"postgres-source","config":{"..."},"tasks":
[],"type":"source"}
```

## Verify connector status

```
sudo curl http://localhost:8083/connectors/postgres-source/
status
```

You should see:

- Connector is `RUNNING`
- Task(s) are `RUNNING`
- No errors

## Consume messages from Kafka

```
sudo /opt/kafka/bin/kafka-console-consumer.sh \
  --bootstrap-server localhost:9092 \
  --topic postgres-users \
  --from-beginning \
  --timeout-ms 5000
```

Expected output:

```
{"id":1,"firstname":"John","lastname":"Doe","email":"john@exa
mple.com","phone":"1234567890","active":true}
{"id":2,"firstname":"Jane","lastname":"Smith","email":"jane@e
xample.com","phone":"0987654321","active":true}
```

## Create the connector configuration

```
sudo tee /opt/kafka/config/postgres-source.json > /dev/null
<<'EOF'
{
  "name": "postgres-source",
  "config": {
    "connector.class":
"io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:postgresql://localhost:5432/
kafkadb",
    "connection.user": "kafkauser",
    "connection.password": "kafkapass",
    "mode": "incrementing",
    "incrementing.column.name": "id",
    "topic.prefix": "postgres-",
    "table.whitelist": "users",
    "poll.interval.ms": "5000"
  }
}
EOF
```

**Explanation of important fields:**

- `connector.class` → JDBC source connector class
- `connection.url` → JDBC URL to your PostgreSQL DB
- `mode=incrementing` → Connector uses the `id` column to track new rows
- `topic.prefix=postgres-` → Kafka topic name will be `postgres-users`
- `table.whitelist=users` → Only read from `users` table
- `poll.interval.ms=5000` → Check for new rows every 5 seconds

# Register the connector with Kafka Connect

Assuming your Kafka Connect **distributed mode REST API** is running on `http://localhost:8083`:

```
sudo curl -X POST -H "Content-Type: application/json" \
  --data @/opt/kafka/config/postgres-source.json \
  http://localhost:8083/connectors
```

Expected response:

```
{"name":"postgres-source","config":{"..."},"tasks":
[],"type":"source"}
```

## Verify connector status

```
sudo curl http://localhost:8083/connectors/postgres-source/
status
```

You should see:

- Connector is `RUNNING`
- Task(s) are `RUNNING`
- No errors

## Consume messages from Kafka

```
sudo /opt/kafka/bin/kafka-console-consumer.sh \
  --bootstrap-server localhost:9092 \
  --topic postgres-users \
  --from-beginning \
  --timeout-ms 5000
```

Expected output:

```
{"id":1,"firstname":"John","lastname":"Doe","email":"john@exa
mple.com","phone":"1234567890","active":true}
{"id":2,"firstname":"Jane","lastname":"Smith","email":"jane@e
xample.com","phone":"0987654321","active":true}
```

## Create the sink table in PostgreSQL

You can create a new table (e.g., `users_sink`) or use the same `users` table. Here's an example with a new table:

```
sudo -i -u postgres
psql -d kafkadb
```

```
CREATE TABLE users_sink (
    id INT PRIMARY KEY,
    firstname VARCHAR(50),
    lastname VARCHAR(50),
    email VARCHAR(100),
    phone VARCHAR(20),
    active BOOLEAN
);
\q
exit
```

## Create the Kafka Connect Sink Connector JSON

```
sudo tee /opt/kafka/config/postgres-sink.json > /dev/null
<<'EOF'
{
  "name": "postgres-sink",
  "config": {
    "connector.class":
"io.confluent.connect.jdbc.JdbcSinkConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:postgresql://localhost:5432/
kafkadb",
    "connection.user": "kafkauser",
    "connection.password": "kafkapass",
    "topics": "postgres-users",
    "auto.create": false,
    "insert.mode": "upsert",
    "pk.mode": "record_value",
    "pk.fields": "id",
    "table.name.format": "users_sink",
    "batch.size": 100
  }
}
EOF
```

**Explanation of important fields:**

- `connector.class` → JDBC Sink Connector class
- `topics` → Kafka topic to read from (`postgres-users`)

- `auto.create=false` → Don't create table automatically (we created `users_sink`)
- `insert.mode=upsert` → Insert new rows or update existing rows based on primary key
- `pk.mode=record_value` → Primary key comes from the Kafka record value
- `pk.fields=id` → Use `id` field as primary key
- `table.name.format=users_sink` → Sink table name

## Register the Sink Connector with Kafka Connect

```
sudo curl -X POST -H "Content-Type: application/json" \
  --data @/opt/kafka/config/postgres-sink.json \
  http://localhost:8083/connectors
```

Expected response:

```
{"name":"postgres-sink","config":{"..."},"tasks":
[],"type":"sink"}
```

## Verify Sink Connector status

`sudo curl http://localhost:8083/connectors/postgres-sink/status`

- Connector should be `RUNNING`
- Task(s) should be `RUNNING`

## Test data flow

## Produce a new message into `postgres-users` topic:

```
echo
'{"id":3,"firstname":"Alice","lastname":"Wonder","email":"ali
ce@example.com","phone":"1112223333","active":true}' | \
```

```
sudo /opt/kafka/bin/kafka-console-producer.sh \
  --topic postgres-users \
  --bootstrap-server localhost:9092
```

Check the data in PostgreSQL:

```
sudo -i -u postgres
psql -d kafkadb
SELECT * FROM users_sink;
\q
exit
```

**Expected output:**

```
 id | firstname | lastname |       email       |    phone
| active
----+-----------+---------+-----------------+------------
+--------
  1 | John      | Doe     | john@example.com | 1234567890 | t
  2 | Jane      | Smith   | jane@example.com | 0987654321 | t
  3 | Alice     | Wonder  | alice@example.com| 1112223333 | t
```

## Kafka Security for kafka and kafka Connect

# Generate SSL certificates

Create a directory for certificates:

```
sudo mkdir -p /opt/kafka/certs
cd /opt/kafka/certs
```

Generate a self-signed certificate for the broker:

```
sudo keytool -genkey -noprompt \
  -alias kafka-broker \
```

```
  -dname "CN=localhost, OU=Kafka, O=Company, L=City, S=State,
C=IN" \
  -keystore kafka.server.keystore.jks \
  -keyalg RSA -storepass brokerpass -keypass brokerpass
-validity 365
```

Export the certificate:

```
sudo keytool -export \
  -alias kafka-broker \
  -file kafka-broker.cer \
  -keystore kafka.server.keystore.jks \
  -storepass brokerpass
```

Create a truststore (for clients):

```
sudo keytool -import -noprompt \
  -alias kafka-broker \
  -file kafka-broker.cer \
  -keystore kafka.server.truststore.jks \
  -storepass brokerpass
```

# Configure Kafka Broker for SASL_SSL

### Edit `/opt/kafka/config/server.properties`:

```
sudo tee -a /opt/kafka/config/server.properties > /dev/null
<<'EOF'

listeners=SASL_SSL://:9093
advertised.listeners=SASL_SSL://localhost:9093
listener.security.protocol.map=SASL_SSL:SASL_SSL
ssl.keystore.location=/opt/kafka/certs/
kafka.server.keystore.jks
ssl.keystore.password=brokerpass
ssl.key.password=brokerpass
ssl.truststore.location=/opt/kafka/certs/
kafka.server.truststore.jks
ssl.truststore.password=brokerpass
sasl.mechanism.inter.broker.protocol=PLAIN
security.inter.broker.protocol=SASL_SSL
```

```
sasl.enabled.mechanisms=PLAIN
EOF
```

# Configure JAAS for broker

### Create `/opt/kafka/config/kafka_server_jaas.conf`:

```
sudo tee /opt/kafka/config/kafka_server_jaas.conf > /dev/null
<<'EOF'
KafkaServer {
   org.apache.kafka.common.security.plain.PlainLoginModule
required
   username="admin"
   password="admin-secret"
   user_admin="admin-secret"
   user_kafkauser="kafkapass";
};
EOF
```

Start Kafka with JAAS:

```
sudo KAFKA_OPTS="-Djava.security.auth.login.config=/opt/
kafka/config/kafka_server_jaas.conf" \
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/
config/server.properties
```

# Configure Kafka Connect for SASL_SSL

Edit `connect-distributed.properties`:

```
sudo tee -a /opt/kafka/config/connect-distributed.properties
> /dev/null <<'EOF'

bootstrap.servers=SASL_SSL://localhost:9093
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
ssl.truststore.location=/opt/kafka/certs/
kafka.server.truststore.jks
ssl.truststore.password=brokerpass
EOF
```

**Create JAAS file for Connect `/opt/kafka/config/kafka_connect_jaas.conf`:**

```
sudo tee /opt/kafka/config/kafka_connect_jaas.conf > /dev/
null <<'EOF'
KafkaClient {
   org.apache.kafka.common.security.plain.PlainLoginModule
required
   username="kafkauser"
   password="kafkapass";
};
EOF
```

**Start Kafka Connect with JAAS:**

```
sudo KAFKA_OPTS="-Djava.security.auth.login.config=/opt/
kafka/config/kafka_connect_jaas.conf" \
/opt/kafka/bin/connect-distributed.sh /opt/kafka/config/
connect-distributed.properties > /tmp/connect.log 2>&1 &
```

# Register JDBC Source/Sink Connectors over SASL_SSL

**Update connector JSONs with:**

```
"bootstrap.servers": "SASL_SSL://localhost:9093",
"security.protocol": "SASL_SSL",
"sasl.mechanism": "PLAIN",
"ssl.truststore.location": "/opt/kafka/certs/
kafka.server.truststore.jks",
"ssl.truststore.password": "brokerpass"
```

**Then register connectors as before:**

```
sudo curl -X POST -H "Content-Type: application/json" \
  --data @/opt/kafka/config/postgres-source.json \
  http://localhost:8083/connectors
```

# Test

### Consume messages securely:

```
sudo /opt/kafka/bin/kafka-console-consumer.sh \
  --bootstrap-server SASL_SSL://localhost:9093 \
  --topic postgres-users \
  --from-beginning \
  --consumer.config /opt/kafka/config/kafka_connect_jaas.conf
```

# Download Schema Registry

```
cd /opt
sudo wget https://packages.confluent.io/archive/7.5/
confluent-7.5.0.tar.gz -O confluent.tar.gz
sudo tar -xvzf confluent.tar.gz
sudo mv confluent-7.5.0 /opt/confluent
sudo rm -f confluent.tar.gz
```

Schema Registry binary path will be:

```
/opt/confluent/bin/schema-registry-start
```

# Create Schema Registry configuration

```
sudo tee /opt/confluent/etc/schema-registry/schema-
registry.properties > /dev/null <<'EOF'
listeners=http://0.0.0.0:8081
kafkastore.bootstrap.servers=SASL_SSL://localhost:9093
kafkastore.security.protocol=SASL_SSL
kafkastore.sasl.mechanism=PLAIN
kafkastore.ssl.truststore.location=/opt/kafka/certs/
kafka.server.truststore.jks
kafkastore.ssl.truststore.password=brokerpass
kafkastore.topic=_schemas
debug=false
EOF
```

**Explanation:**

- `listeners` → Schema Registry will listen on port 8081 HTTP
- `kafkastore.bootstrap.servers` → Kafka cluster with SASL_SSL
- `kafkastore.topic` → internal Kafka topic to store schemas

# Create Schema Registry internal topic

```
sudo /opt/kafka/bin/kafka-topics.sh \
  --create \
  --topic _schemas \
  --partitions 1 \
  --replication-factor 1 \
  --bootstrap-server SASL_SSL://localhost:9093 \
  --command-config /opt/kafka/config/kafka_connect_jaas.conf
```

# Start Schema Registry

```
sudo /opt/confluent/bin/schema-registry-start /opt/confluent/
etc/schema-registry/schema-registry.properties > /tmp/schema-
registry.log 2>&1 &
```

### Check logs:

```
sudo tail -f /tmp/schema-registry.log
```

# Test Schema Registry

Register a sample Avro schema for `users` table:

Create `/opt/confluent/user.avsc`:

```
sudo tee /opt/confluent/user.avsc > /dev/null <<'EOF'
```

```
{
  "type": "record",
  "name": "User",
  "namespace": "com.example",
  "fields": [
    {"name": "id", "type": "int"},
    {"name": "firstname", "type": "string"},
    {"name": "lastname", "type": "string"},
    {"name": "email", "type": "string"},
    {"name": "phone", "type": "string"},
    {"name": "active", "type": "boolean"}
  ]
}
EOF
```

Register schema:

```
sudo curl -X POST -H "Content-Type: application/
vnd.schemaregistry.v1+json" \
  --data '{"schema": "'"$(sudo cat /opt/confluent/user.avsc |
jq -c .)"'"}' \
  http://localhost:8081/subjects/User/versions
```

Expected response:

```
{"id":1}
```

# Configure Kafka Connect to use Schema Registry

Add to your connector JSON (`postgres-source.json` or `sink.json`):

```
"value.converter": "io.confluent.connect.avro.AvroConverter",
"value.converter.schema.registry.url": "http://
localhost:8081",
"key.converter": "io.confluent.connect.avro.AvroConverter",
"key.converter.schema.registry.url": "http://localhost:8081"
```

Restart Kafka Connect to pick up the changes:

```
sudo pkill -f connect-distributed
sudo KAFKA_OPTS="-Djava.security.auth.login.config=/opt/
kafka/config/kafka_connect_jaas.conf" \
```

```
/opt/kafka/bin/connect-distributed.sh /opt/kafka/config/
connect-distributed.properties > /tmp/connect.log 2>&1 &
```

Now you have **Schema Registry running with secure Kafka**.

- Kafka Connect can produce/consume **Avro data**
- Schemas are centrally managed in `_schemas` topic

### Monitoring Kafka, Kafka Connect, and Schema Registry using Prometheus on port 9095 + Grafana with all sudo commands for RHEL/CentOS.

# Install Prometheus

```
cd /opt
sudo wget https://github.com/prometheus/prometheus/releases/
download/v2.47.0/prometheus-2.47.0.linux-amd64.tar.gz
sudo tar -xvzf prometheus-2.47.0.linux-amd64.tar.gz
sudo mv prometheus-2.47.0.linux-amd64 prometheus
sudo rm -f prometheus-2.47.0.linux-amd64.tar.gz
```

**Prometheus binary path: `/opt/prometheus/prometheus`**

# Configure Prometheus (port 9095)

```
sudo tee /opt/prometheus/prometheus.yml > /dev/null <<'EOF'
global:
  scrape_interval: 10s

scrape_configs:
  - job_name: 'kafka'
    static_configs:
      - targets: ['localhost:7071']  # JMX exporter port
```

```
  - job_name: 'connect'
    static_configs:
      - targets: ['localhost:8083']

  - job_name: 'schema-registry'
    static_configs:
      - targets: ['localhost:8081']

web:
  listen-address: ":9095"
EOF
```

# Install Kafka JMX Exporter

```
cd /opt/kafka/libs
sudo wget https://repo1.maven.org/maven2/io/prometheus/jmx/
jmx_prometheus_javaagent/0.17.2/
jmx_prometheus_javaagent-0.17.2.jar
sudo wget https://raw.githubusercontent.com/prometheus/
jmx_exporter/master/example_configs/kafka-2_0_0.yml -O kafka-
jmx.yml
```

# Start Kafka Broker with JMX Exporter

```
sudo KAFKA_OPTS="-javaagent:/opt/kafka/libs/
jmx_prometheus_javaagent-0.17.2.jar=7071:/opt/kafka/libs/
kafka-jmx.yml \
-Djava.security.auth.login.config=/opt/kafka/config/
kafka_server_jaas.conf" \
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/
config/server.properties
```

- `7071` → port for JMX metrics

# Start Prometheus

```
cd /opt/prometheus
sudo ./prometheus --config.file=prometheus.yml > /tmp/
prometheus.log 2>&1 &
```

Check logs:

```
tail -f /tmp/prometheus.log
```

Access web UI: `http://<server-ip>:9095`

# Install Grafana

```
sudo dnf install -y grafana
sudo systemctl enable grafana-server
sudo systemctl start grafana-server
```

Access Grafana web UI: `http://localhost:3000`

- **Default login: `admin/admin`**

# Add Prometheus as Grafana Data Source

1. Login → Settings → Data Sources → Add → Prometheus
2. URL: `http://localhost:9095`
3. Click **Save & Test**

# Import Kafka Dashboards

- Grafana dashboard IDs:
    - **721** → Kafka Overview
    - **7588** → Kafka Connect
- Import dashboard → select Prometheus as data source → metrics populate automatically

Now you have **full monitoring setup**:

- Prometheus scrapes **Kafka (via JMX)**, **Kafka Connect**, and **Schema Registry**
- Grafana visualizes metrics in dashboards
- Prometheus runs on **port 9095**

**C**reate **two Kafka clusters** (source + target) on a single RHEL/ CentOS machine for testing **MirrorMaker 2, all step-by-step `sudo` commands** with SASL_SSL security, so you can run MM2 between these clusters.

# Directory structure for clusters

```
cd /opt/kafka
sudo mkdir -p clusterA clusterB
sudo cp -r config clusterA/
sudo cp -r config clusterB/
```

- `clusterA` → Source cluster
- `clusterB` → Target cluster

# Configure clusterA (source)

**Edit `/opt/kafka/clusterA/config/server.properties`:**

```
sudo tee /opt/kafka/clusterA/config/server.properties > /dev/
null <<'EOF'
broker.id=1
listeners=SASL_SSL://:9093
advertised.listeners=SASL_SSL://localhost:9093
log.dirs=/opt/kafka/clusterA/logs
zookeeper.connect=localhost:2181
listener.security.protocol.map=SASL_SSL:SASL_SSL
ssl.keystore.location=/opt/kafka/certs/
kafka.server.keystore.jks
ssl.keystore.password=brokerpass
ssl.key.password=brokerpass
ssl.truststore.location=/opt/kafka/certs/
kafka.server.truststore.jks
ssl.truststore.password=brokerpass
sasl.mechanism.inter.broker.protocol=PLAIN
security.inter.broker.protocol=SASL_SSL
sasl.enabled.mechanisms=PLAIN
EOF
```

# Configure clusterB (target)

**Edit `/opt/kafka/clusterB/config/server.properties`:**

```
sudo tee /opt/kafka/clusterB/config/server.properties > /dev/
null <<'EOF'
broker.id=2
listeners=SASL_SSL://:9094
advertised.listeners=SASL_SSL://localhost:9094
log.dirs=/opt/kafka/clusterB/logs
zookeeper.connect=localhost:2181
listener.security.protocol.map=SASL_SSL:SASL_SSL
ssl.keystore.location=/opt/kafka/certs/
kafka.server.keystore.jks
ssl.keystore.password=brokerpass
ssl.key.password=brokerpass
ssl.truststore.location=/opt/kafka/certs/
kafka.server.truststore.jks
ssl.truststore.password=brokerpass
sasl.mechanism.inter.broker.protocol=PLAIN
security.inter.broker.protocol=SASL_SSL
```

```
sasl.enabled.mechanisms=PLAIN
EOF
```

# Create JAAS file (common for both)

**/opt/kafka/config/kafka_server_jaas.conf**:

```
sudo tee /opt/kafka/config/kafka_server_jaas.conf > /dev/null
<<'EOF'
KafkaServer {
   org.apache.kafka.common.security.plain.PlainLoginModule
required
   username="admin"
   password="admin-secret"
   user_admin="admin-secret"
   user_kafkauser="kafkapass";
};
EOF
```

# Start ZooKeeper

```
sudo /opt/kafka/bin/zookeeper-server-start.sh -daemon /opt/
kafka/config/zookeeper.properties
```

# Start Kafka clusters

### ClusterA (source):

```
sudo KAFKA_OPTS="-Djava.security.auth.login.config=/opt/
kafka/config/kafka_server_jaas.conf" \
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/
clusterA/config/server.properties
```

### ClusterB (target):

```
sudo KAFKA_OPTS="-Djava.security.auth.login.config=/opt/
kafka/config/kafka_server_jaas.conf" \
/opt/kafka/bin/kafka-server-start.sh -daemon /opt/kafka/
clusterB/config/server.properties
```

# Verify clusters

**List topics in each cluster:**

```
sudo /opt/kafka/bin/kafka-topics.sh --bootstrap-server
localhost:9093 --list  # clusterA
sudo /opt/kafka/bin/kafka-topics.sh --bootstrap-server
localhost:9094 --list  # clusterB
```

# Configure MirrorMaker 2

**Create `/opt/kafka/config/mm2.properties`:**

```
sudo tee /opt/kafka/config/mm2.properties > /dev/null <<'EOF'
clusters = clusterA, clusterB

clusterA.bootstrap.servers = localhost:9093
clusterA.security.protocol = SASL_SSL
clusterA.sasl.mechanism = PLAIN
clusterA.ssl.truststore.location = /opt/kafka/certs/
kafka.server.truststore.jks
clusterA.ssl.truststore.password = brokerpass

clusterB.bootstrap.servers = localhost:9094
clusterB.security.protocol = SASL_SSL
clusterB.sasl.mechanism = PLAIN
clusterB.ssl.truststore.location = /opt/kafka/certs/
kafka.server.truststore.jks
clusterB.ssl.truststore.password = brokerpass

replication.policy.class =
org.apache.kafka.connect.mirror.DefaultReplicationPolicy
topics = .*
heartbeat.interval.seconds = 5
offset-syncs.topic.replication.factor = 1
EOF
```

# Start MirrorMaker 2

```
sudo /opt/kafka/bin/connect-mirror-maker.sh /opt/kafka/
config/mm2.properties > /tmp/mm2.log 2>&1 &
```

Check logs:

```
tail -f /tmp/mm2.log
```

# Test replication

1. Create topic in source cluster:

```
sudo /opt/kafka/bin/kafka-topics.sh --create --topic test-
topic --partitions 1 --replication-factor 1 --bootstrap-
server localhost:9093
```

2. Produce message in source cluster:

```
echo "Hello MirrorMaker 2" | sudo /opt/kafka/bin/kafka-
console-producer.sh --topic test-topic --bootstrap-server
localhost:9093
```

3. Consume from target cluster:

```
sudo /opt/kafka/bin/kafka-console-consumer.sh --topic test-
topic --bootstrap-server localhost:9094 --from-beginning
```

You should see the message replicated.

Now you have:

- **ClusterA** → source Kafka cluster
- **ClusterB** → target Kafka cluster
- **MirrorMaker 2** replicating topics securely via **SASL_SSL**