

# Mémento Kafka Management



## Table des Matières

- [Kafka CLI](#)
- [Kafka Producer](#)
- [Kafka Consumer](#)
- [Kafka Streams](#)
- [KSQLDB](#)
- [Kafka Management](#)
- [Configurations Importantes](#)
- [Patterns et Bonnes Pratiques](#)

## Kafka CLI

### Commandes de Base

Commande	Description	Exemple
<code>kafka-topics.sh -create</code>	Créer un topic	<code>kafka-topics.sh --bootstrap-server localhost:9092 --create --topic mon-topic --partitions 3 --replication-factor 1</code>
<code>kafka-topics.sh -list</code>	Lister les topics	<code>kafka-topics.sh --bootstrap-server localhost:9092 --list</code>
<code>kafka-topics.sh -describe</code>	Décrire un topic	<code>kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic mon-topic</code>
<code>kafka-topics.sh -delete</code>	Supprimer un topic	<code>kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic mon-topic</code>

### Production et Consommation CLI

Commande	Description	Exemple
<code>kafka-console-producer.sh</code>	Producer en ligne de commande	<code>kafka-console-producer.sh --bootstrap-server localhost:9092 --topic mon-topic</code>
<code>kafka-console-consumer.sh</code>	Consommer en ligne de commande	<code>kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic mon-topic --from-beginning</code>
<code>kafka-console-consumer.sh --group</code>	Consommer avec groupe	<code>kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic mon-topic --group mon-groupe</code>

# Gestion des Consumer Groups

Commande	Description	Exemple
--list	Lister les groupes	kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list
--describe	Détails d'un groupe	kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group mon-groupe --describe
--reset-offsets	Reset des offsets	kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group mon-groupe --reset-offsets --to-earliest --topic mon-topic --execute

## Commandes Avancées

Commande	Description	Exemple
kafka-log-dirs.sh	Analyser les logs	kafka-log-dirs.sh --bootstrap-server localhost:9092 --describe --json
kafka-configs.sh	Configuration dynamique	kafka-configs.sh --bootstrap-server localhost:9092 --entity-type topics --entity-name mon-topic --alter --add-config retention.ms=86400000
kafka-replica-verification.sh	Vérifier les réplicas	kafka-replica-verification.sh --broker-list localhost:9092 --topic-white-list "mon-topic"

# Kafka Producer

## Configuration Python (kafka-python)

Paramètre	Description	Valeur par défaut	Exemple
bootstrap_servers	Liste des brokers	['localhost:9092']	['broker1:9092', 'broker2:9092']
value_serializer	Sérialiseur pour les valeurs	None	lambda v: json.dumps(v).encode('utf-8')
key_serializer	Sérialiseur pour les clés	None	lambda k: str(k).encode('utf-8')
acks	Niveau d'accusé de réception	1	'all' (plus fiable)
retries	Nombre de tentatives	2147483647	3
batch_size	Taille des lots	16384	32768
linger_ms	Attente avant envoi	0	5

Paramètre	Description	Valeur par défaut	Exemple
<code>buffer_memory</code>	Mémoire tampon	<code>33554432</code>	<code>67108864</code>

## Configuration Java (Apache Kafka Client)

Paramètre	Description	Valeur par défaut	Exemple
<code>bootstrap.servers</code>	Liste des brokers	N/A (requis)	<code>"localhost:9092"</code>
<code>key.serializer</code>	Classe sérialiseur clé	N/A (requis)	<code>StringSerializer.class</code>
<code>value.serializer</code>	Classe sérialiseur valeur	N/A (requis)	<code>StringSerializer.class</code>
<code>acks</code>	Niveau d'accusé	<code>"1"</code>	<code>"all"</code>
<code>retries</code>	Nombre de tentatives	<code>Integer.MAX_VALUE</code>	<code>3</code>
<code>max.in.flight.requests.per.connection</code>	Requêtes en vol	<code>5</code>	<code>1</code> (ordre garanti)
<code>enable.idempotence</code>	Idempotence	<code>false</code>	<code>true</code>
<code>compression.type</code>	Type de compression	<code>"none"</code>	<code>"snappy"</code>

## Patterns Producer

Pattern	Python	Java
Envoi Asynchrone	<code>producer.send(topic, value)</code>	<code>producer.send(record, callback)</code>
Envoi Synchrone	<code>producer.send().get(timeout=10)</code>	<code>producer.send().get()</code>
Callback	Via Future	<code>(metadata, exception) -&gt; {...}</code>
Partitioning	<code>partition=1</code> dans <code>send</code>	Custom <code>Partitioner</code>
Headers	<code>headers=[('key', b'value')]</code>	<code>record.headers().add()</code>

## Kafka Consumer

### Configuration Python (kafka-python)

Paramètre	Description	Valeur par défaut	Exemple
<code>bootstrap_servers</code>	Liste des brokers	<code>['localhost:9092']</code>	<code>['broker1:9092']</code>
<code>group_id</code>	ID du groupe de consommateurs	None	<code>'mon-consumer-group'</code>
<code>auto_offset_reset</code>	Position initiale	<code>'latest'</code>	<code>'earliest'</code>
<code>enable_auto_commit</code>	Commit automatique	True	False (manuel)
<code>auto_commit_interval_ms</code>	Intervalle commit auto	5000	1000
<code>max_poll_records</code>	Records max par poll	500	100
<code>session_timeout_ms</code>	Timeout session	10000	30000
<code>heartbeat_interval_ms</code>	Intervalle heartbeat	3000	10000

## Configuration Java (Apache Kafka Client)

Paramètre	Description	Valeur par défaut	Exemple
<code>bootstrap.servers</code>	Liste des brokers	N/A (requis)	<code>"localhost:9092"</code>
<code>group.id</code>	ID du groupe	<code>null</code>	<code>"mon-consumer-group"</code>
<code>key.deserializer</code>	Désérialiseur clé	N/A (requis)	<code>StringDeserializer.class</code>
<code>value.deserializer</code>	Désérialiseur valeur	N/A (requis)	<code>StringDeserializer.class</code>
<code>auto.offset.reset</code>	Position initiale	<code>"latest"</code>	<code>"earliest"</code>
<code>enable.auto.commit</code>	Commit automatique	<code>true</code>	<code>false</code>
<code>max.poll.records</code>	Records max par poll	500	100
<code>fetch.min.bytes</code>	Taille min fetch	1	1024

## Strategies de Commit

Stratégie	Python	Java	Use Case
Auto Commit	<code>enable_auto_commit=True</code>	<code>enable.auto.commit=true</code>	Simple, perte possible
Commit Manuel Sync	<code>consumer.commit()</code>	<code>consumer.commitSync()</code>	Fiabilité, performance moindre
Commit Manuel Async	<code>consumer.commit_async()</code>	<code>consumer.commitAsync()</code>	Performance, callback requis
Commit par Record	Après traitement	Après chaque record	Très fiable, lent

# Exemples de Code Consumer

## Python - Auto Commit

```
from kafka import KafkaConsumer
import json
import logging

# Configuration avec auto-commit
consumer = KafkaConsumer(
    'mon-topic',
    bootstrap_servers=['localhost:9092'],
    group_id='auto-commit-group',
    enable_auto_commit=True,          # Auto-commit activé
    auto_commit_interval_ms=5000,     # Commit toutes les 5s
    auto_offset_reset='earliest',
    value_deserializer=lambda m: json.loads(m.decode('utf-8'))
)

print("Consumer avec auto-commit démarré...")
try:
    for message in consumer:
        print(f"Reçu: {message.value}")

        # Traitement du message
        process_message(message.value)

        # Pas besoin de commit manuel, fait automatiquement

except Exception as e:
    logging.error(f"Erreur: {e}")
finally:
    consumer.close()

def process_message(data):
    # Simulation traitement
    print(f"Traitement message ID: {data.get('id')}")
```

## Python - Commit Manuel

```

from kafka import KafkaConsumer, TopicPartition, OffsetAndMetadata
import json
import logging

# Configuration avec commit manuel
consumer = KafkaConsumer(
    'mon-topic',
    bootstrap_servers=['localhost:9092'],
    group_id='manual-commit-group',
    enable_auto_commit=False,          # Auto-commit désactivé
    auto_offset_reset='earliest',
    value_deserializer=lambda m: json.loads(m.decode('utf-8'))
)

print("Consumer avec commit manuel démarré...")
try:
    for message in consumer:
        try:
            print(f"Reçu: {message.value}")

            # Traitement du message
            if process_message(message.value):
                # Commit synchrone après traitement réussi
                consumer.commit()
                print(f"Message committé: offset {message.offset}")

        except Exception as e:
            logging.error(f"Erreur traitement: {e}")
            # En cas d'erreur, on ne commit pas

except KeyboardInterrupt:
    print("Arrêt du consumer")
finally:
    consumer.close()

def process_message(data):
    try:
        # Simulation traitement pouvant échouer
        if data.get('id', 0) % 10 == 0:
            raise Exception("Erreur simulée")
    
```

```
        print(f"Traitement réussi pour ID: {data.get('id')}")
        return True
    except Exception as e:
        logging.error(f"Échec traitement: {e}")
        return False
```

## Python - Commit par Batch

```
from kafka import KafkaConsumer
import json

consumer = KafkaConsumer(
    'mon-topic',
    bootstrap_servers=['localhost:9092'],
    group_id='batch-commit-group',
    enable_auto_commit=False,
    max_poll_records=10,          # Max 10 messages par poll
    auto_offset_reset='earliest',
    value_deserializer=lambda m: json.loads(m.decode('utf-8'))
)

batch_size = 5
message_count = 0

print("Consumer avec commit par batch démarré...")
try:
    for message in consumer:
        print(f"Reçu: {message.value}")
        process_message(message.value)

        message_count += 1

        # Commit tous les N messages
        if message_count % batch_size == 0:
            consumer.commit()
            print(f"Batch de {batch_size} messages committé")

except KeyboardInterrupt:
    # Commit final avant fermeture
```

```
consumer.commit()

print("Commit final effectué")

finally:
    consumer.close()
```

## Java - Auto Commit

```
import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.serialization.StringDeserializer;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

public class AutoCommitConsumer {
    private KafkaConsumer<String, String> consumer;
    private ObjectMapper objectMapper = new ObjectMapper();

    public AutoCommitConsumer() {
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "auto-commit-group");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, true); // Auto-commit activ  
        props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, 5000); // Commit toutes les 5
s
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        consumer = new KafkaConsumer<>(props);
    }

    public void consume() {
        consumer.subscribe(Arrays.asList("mon-topic"));

        System.out.println("Consumer avec auto-commit d  marr  ...");

        try {
            while (true) {
```



```

        ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(10
0));

        for (ConsumerRecord<String, String> record : records) {
            System.out.println("Reçu: " + record.value());

            // Traitement du message
            processMessage(record.value());

            // Pas de commit manuel nécessaire
        }
    }
} catch (Exception e) {
    System.err.println("Erreur: " + e.getMessage());
} finally {
    consumer.close();
}
}

private void processMessage(String jsonMessage) {
    try {
        Message message = objectMapper.readValue(jsonMessage, Message.class);
        System.out.println("Traitement message ID: " + message.id);
    } catch (Exception e) {
        System.err.println("Erreur parsing: " + e.getMessage());
    }
}

public static void main(String[] args) {
    new AutoCommitConsumer().consume();
}
}

```

## Java - Commit Manuel

```

import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.TopicPartition;

import java.time.Duration;
import java.util.Arrays;

```

```

import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

public class ManualCommitConsumer {
    private KafkaConsumer<String, String> consumer;
    private ObjectMapper objectMapper = new ObjectMapper();

    public ManualCommitConsumer() {
        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "manual-commit-group");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, false);    // Auto-commit désacti
vé

        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        consumer = new KafkaConsumer<>(props);
    }

    public void consumeWithSyncCommit() {
        consumer.subscribe(Arrays.asList("mon-topic"));

        System.out.println("Consumer avec commit manuel synchrone démarré...");

        try {
            while (true) {
                ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(10
0));

                for (ConsumerRecord<String, String> record : records) {
                    try {
                        System.out.println("Reçu: " + record.value());

                        // Traitement du message
                        if (processMessage(record.value())) {
                            // Commit synchrone après chaque message réussi
                            Map<TopicPartition, OffsetAndMetadata> offsets = new HashMap<>();
                            offsets.put(

```

```

        new TopicPartition(record.topic(), record.partition()),
        new OffsetAndMetadata(record.offset() + 1)
    );
    consumer.commitSync(offsets);
    System.out.println("Message committé: offset " + record.offset());
}

} catch (Exception e) {
    System.err.println("Erreur traitement: " + e.getMessage());
    // En cas d'erreur, on ne commit pas
}

}

} finally {
    consumer.close();
}
}

```

```

public void consumeWithAsyncCommit() {
    consumer.subscribe(Arrays.asList("mon-topic"));

    System.out.println("Consumer avec commit manuel asynchrone démarré...");

    try {
        while (true) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(10
0));

            for (ConsumerRecord<String, String> record : records) {
                System.out.println("Reçu: " + record.value());
                processMessage(record.value());
            }

            // Commit asynchrone après le batch
            if (!records.isEmpty()) {
                consumer.commitAsync((offsets, exception) -> {
                    if (exception != null) {
                        System.err.println("Erreur commit: " + exception.getMessage());
                    } else {
                        System.out.println("Batch committé avec succès");
                    }
                });
            }
        }
    } catch (Exception e) {
        System.err.println("Erreur lors du démarrage du consumer: " + e.getMessage());
    }
}

```

```

        }

        });

    }

}

} finally {
    // Commit synchrone final pour garantir la persistance
    try {
        consumer.commitSync();
    } finally {
        consumer.close();
    }
}

}

}

private boolean processMessage(String jsonMessage) {
    try {
        Message message = objectMapper.readValue(jsonMessage, Message.class);

        // Simulation d'un traitement pouvant échouer
        if (message.id % 10 == 0) {
            throw new RuntimeException("Erreur simulée");
        }

        System.out.println("Traitement réussi pour ID: " + message.id);
        return true;

    } catch (Exception e) {
        System.err.println("Échec traitement: " + e.getMessage());
        return false;
    }
}

}
}

```

## Exemples Consumer Groups

### Python - Groupe de Consumers

```

from kafka import KafkaConsumer
import json
import threading

```

```
import time
import signal
import sys

class ConsumerGroup:
    def __init__(self, group_id, topic, num_consumers=3):
        self.group_id = group_id
        self.topic = topic
        self.num_consumers = num_consumers
        self.consumers = []
        self.threads = []
        self.running = True

    def create_consumer(self, consumer_id):
        """Crée un consumer pour le groupe"""
        consumer = KafkaConsumer(
            self.topic,
            bootstrap_servers=['localhost:9092'],
            group_id=self.group_id,
            client_id=f'consumer-{consumer_id}',
            enable_auto_commit=True,
            auto_commit_interval_ms=1000,
            auto_offset_reset='earliest',
            value_deserializer=lambda m: json.loads(m.decode('utf-8'))
        )
        return consumer

    def consumer_worker(self, consumer_id):
        """Worker pour un consumer individuel"""
        consumer = self.create_consumer(consumer_id)
        self.consumers.append(consumer)

        print(f"Consumer {consumer_id} démarré")

        try:
            while self.running:
                msg_pack = consumer.poll(timeout_ms=1000)

                for topic_partition, messages in msg_pack.items():
                    for message in messages:
```

```

        print(f"[Consumer {consumer_id}] Partition {message.partition}, "
              f"Offset {message.offset}: {message.value}")

        # Simulation traitement
        time.sleep(0.1)

        # Traitement spécifique par consumer
        self.process_message(consumer_id, message.value)

    except Exception as e:
        print(f"Consumer {consumer_id} erreur: {e}")
    finally:
        consumer.close()
        print(f"Consumer {consumer_id} fermé")

def process_message(self, consumer_id, data):
    """Traite un message avec l'ID du consumer"""
    message_id = data.get('id', 'unknown')
    print(f"[Consumer {consumer_id}] Traitement message {message_id}")

    # Simulation de traitement différencié
    if consumer_id == 0:
        print(f" -> Traitement prioritaire par consumer {consumer_id}")
    else:
        print(f" -> Traitement standard par consumer {consumer_id}")

def start(self):
    """Démarré le groupe de consumers"""
    print(f"Démarrage du groupe '{self.group_id}' avec {self.num_consumers} consumers")

    # Créer et démarrer les threads consumers
    for i in range(self.num_consumers):
        thread = threading.Thread(target=self.consumer_worker, args=(i,))
        thread.daemon = True
        thread.start()
        self.threads.append(thread)
        time.sleep(1) # Étalement du démarrage

def stop(self):
    """Arrête proprement le groupe de consumers"""

```

```

print("Arrêt du groupe de consumers...")
self.running = False

# Fermer tous les consumers
for consumer in self.consumers:
    consumer.close()

# Attendre la fin des threads
for thread in self.threads:
    thread.join(timeout=5)

print("Groupe de consumers arrêté")

# Gestionnaire de signal pour arrêt propre
def signal_handler(sig, frame, consumer_group):
    consumer_group.stop()
    sys.exit(0)

# Utilisation
if __name__ == "__main__":
    group = ConsumerGroup("demo-consumer-group", "mon-topic", num_consumers=3)

    # Configuration du gestionnaire de signal
    signal.signal(signal.SIGINT, lambda s, f: signal_handler(s, f, group))

    try:
        group.start()

        # Maintenir le programme en vie
        while True:
            time.sleep(1)

    except KeyboardInterrupt:
        group.stop()

```

## Java - Groupe de Consumers

```

import org.apache.kafka.clients.consumer.*;
import org.apache.kafka.common.serialization.StringDeserializer;
import com.fasterxml.jackson.databind.ObjectMapper;

```

```
import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;

public class ConsumerGroupExample {
    private final String groupId;
    private final String topic;
    private final int numConsumers;
    private final AtomicBoolean running = new AtomicBoolean(true);
    private ExecutorService executorService;
    private ObjectMapper objectMapper = new ObjectMapper();

    public ConsumerGroupExample(String groupId, String topic, int numConsumers) {
        this.groupId = groupId;
        this.topic = topic;
        this.numConsumers = numConsumers;
        this.executorService = Executors.newFixedThreadPool(numConsumers);
    }

    private Properties createConsumerConfig(int consumerId) {
        Properties props = new Properties();
        props.put(ConsumerConfig.BootstrapServersConfig, "localhost:9092");
        props.put(ConsumerConfig.GroupIdConfig, groupId);
        props.put(ConsumerConfig.ClientIdConfig, "consumer-" + consumerId);
        props.put(ConsumerConfig.KeyDeserializerClassConfig, StringDeserializer.class);
        props.put(ConsumerConfig.ValueDeserializerClassConfig, StringDeserializer.class);
        props.put(ConsumerConfig.EnableAutoCommitConfig, true);
        props.put(ConsumerConfig.AutoCommitIntervalMsConfig, 1000);
        props.put(ConsumerConfig.AutoOffsetResetConfig, "earliest");
        props.put(ConsumerConfig.MaxPollRecordsConfig, 10);

        return props;
    }

    private class ConsumerWorker implements Runnable {
```



```

private final int consumerId;

private KafkaConsumer<String, String> consumer;

public ConsumerWorker(int consumerId) {
    this.consumerId = consumerId;
}

@Override
public void run() {
    consumer = new KafkaConsumer<>(createConsumerConfig(consumerId));
    consumer.subscribe(Arrays.asList(topic));

    System.out.println("Consumer " + consumerId + " démarré");

    try {
        while (running.get()) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis
(1000));

            for (ConsumerRecord<String, String> record : records) {
                System.out.printf("[Consumer %d] Partition %d, Offset %d: %s\n",
                    consumerId, record.partition(), record.offset(), record.value
());

                // Simulation traitement
                processMessage(consumerId, record.value());

                // Simulation latence
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                }
            }
        }
    } catch (Exception e) {
        System.err.println("Consumer " + consumerId + " erreur: " + e.getMessage());
    } finally {
        consumer.close();
    }
}

```

```

        System.out.println("Consumer " + consumerId + " fermé");
    }
}

private void processMessage(int consumerId, String jsonMessage) {
    try {
        Message message = objectMapper.readValue(jsonMessage, Message.class);

        System.out.printf("[Consumer %d] Traitement message ID: %d%n",
            consumerId, message.id);

        // Traitement différencié par consumer
        if (consumerId == 0) {
            System.out.println(" -> Traitement prioritaire par consumer " + consumerI
d);
        } else {
            System.out.println(" -> Traitement standard par consumer " + consumerId);
        }

    } catch (Exception e) {
        System.err.println("Erreur parsing par consumer " + consumerId + ": " + e.getM
essage());
    }
}

public void start() {
    System.out.printf("Démarrage du groupe '%s' avec %d consumers%n",
        groupId, numConsumers);

    // Démarrer tous les consumers
    for (int i = 0; i < numConsumers; i++) {
        executorService.submit(new ConsumerWorker(i));

        // Étalement du démarrage
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            break;
        }
    }
}

```

```

        }
    }
}

public void stop() {
    System.out.println("Arrêt du groupe de consumers...");
    running.set(false);

    executorService.shutdown();
    try {
        if (!executorService.awaitTermination(10, TimeUnit.SECONDS)) {
            executorService.shutdownNow();
        }
    } catch (InterruptedException e) {
        executorService.shutdownNow();
    }

    System.out.println("Groupe de consumers arrêté");
}

public static void main(String[] args) {
    ConsumerGroupExample group = new ConsumerGroupExample(
        "demo-consumer-group", "mon-topic", 3);

    // Gestionnaire d'arrêt propre
    Runtime.getRuntime().addShutdownHook(new Thread(group::stop));

    try {
        group.start();

        // Maintenir le programme en vie
        Thread.sleep(Long.MAX_VALUE);

    } catch (InterruptedException e) {
        group.stop();
    }
}

}

class Message {

```

```
public int id;

public long timestamp;

public String data;


// Constructeurs, getters, setters...
}
```

# Kafka Streams

## Topologies de Base

Opération	Description	Java API	Python (Concept)
Source	Lecture depuis topic	<code>builder.stream("topic")</code>	<code>consumer = KafkaConsumer("topic")</code>
Sink	Écriture vers topic	<code>stream.to("output-topic")</code>	<code>producer.send("output-topic", data)</code>
Filter	Filtrage des messages	<code>stream.filter((k,v) -&gt; condition)</code>	<code>if condition: process(message)</code>
Map	Transformation 1:1	<code>stream.map((k,v) -&gt; new KeyValue&lt;&gt;())</code>	<code>transformed = transform(message)</code>
FlatMap	Transformation 1:n	<code>stream.flatMap()</code>	<code>for item in expand(message)</code>

## Agrégations et Fenêtres

Type de Fenêtre	Java API	Description	Use Case
Tumbling	<code>TimeWindows.of(Duration.ofMinutes(5))</code>	Fenêtres fixes sans chevauchement	Compteurs par heure
Hopping	<code>TimeWindows.of().advanceBy()</code>	Fenêtres fixes avec chevauchement	Moyennes mobiles
Session	<code>SessionWindows.with()</code>	Basé sur l'activité	Sessions utilisateur
Sliding	Pour les jointures	Fenêtre glissante	Corrélations temporelles

## Jointures

Type de Jointure	Streams API	Description
Stream-Stream	<code>stream1.join(stream2)</code>	Jointure temporelle
Stream-Table	<code>stream.join(table)</code>	Enrichissement
Table-Table	<code>table1.join(table2)</code>	Jointure de référence
Outer Join	<code>leftJoin()</code> , <code>outerJoin()</code>	Jointures externes

## Configuration Streams

Paramètre	Description	Valeur par défaut	Production
<code>application.id</code>	ID unique de l'application	N/A (requis)	"mon-streams-app"
<code>bootstrap.servers</code>	Brokers Kafka	N/A (requis)	"broker1:9092,broker2:9092"
<code>default.key.serde</code>	Serde par défaut pour clés	<code>ByteArraySerde</code>	<code>Serdes.String()</code>
<code>default.value.serde</code>	Serde par défaut pour valeurs	<code>ByteArraySerde</code>	Custom Serde
<code>num.stream.threads</code>	Nombre de threads	1	4-8 (selon CPU)
<code>processing.guarantee</code>	Garantie de traitement	"at_least_once"	"exactly_once_v2"
<code>commit.interval.ms</code>	Intervalle de commit	30000	10000

## Exemples de Code Kafka Streams

### Java - Stream Processing Complet

```
import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.kstream.*;
import org.apache.kafka.streams.kstream.Materialized;
import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;

import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

public class KafkaStreamsProcessor {
```

```

private static final ObjectMapper mapper = new ObjectMapper();

public static void main(String[] args) {
    // Configuration
    Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-processor");
    props.put(StreamsConfig.BootstrapServersConfig, "localhost:9092");
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());
    props.put(StreamsConfig.PROCESSING_GUARANTEE_CONFIG, StreamsConfig.EXACTLY_ONCE_V2);

    StreamsBuilder builder = new StreamsBuilder();

    // 1. Stream source
    KStream<String, String> orders = builder.stream("orders");

    // 2. Filtrage et transformation
    KStream<String, String> validOrders = orders
        .filter((key, value) -> isValidOrder(value))
        .mapValues(value -> enrichOrder(value));

    // 3. Agrégation par fenêtre (ventes par heure)
    KTable<Windowed<String>, Double> hourlySales = validOrders
        .groupByKey()
        .windowedBy(TimeWindows.of(Duration.ofHours(1)))
        .aggregate(
            () -> 0.0, // Initializer
            (key, newValue, aggregate) -> {
                try {
                    JsonNode order = mapper.readTree(newValue);
                    double amount = order.get("amount").asDouble();
                    return aggregate + amount;
                } catch (Exception e) {
                    return aggregate;
                }
            },
            Materialized.with(Serdes.String(), Serdes.Double())
        );

    // 4. Comptage des commandes par client

```

```

KTable<String, Long> customerOrderCount = validOrders
    .groupBy((key, value) -> {
        try {
            JsonNode order = mapper.readTree(value);
            return order.get("customerId").asText();
        } catch (Exception e) {
            return "unknown";
        }
    })
    .count(Materialized.as("customer-order-counts"));

// 5. Détection des clients VIP (>10 commandes)
KStream<String, String> vipCustomers = customerOrderCount
    .toStream()
    .filter((customerId, count) -> count > 10)
    .mapValues((customerId, count) ->
        String.format("{\"customerId\":\"%s\", \"orderCount\":%d, \"status\":\"VIP\"}",
            customerId, count));

// 6. Jointure avec les données client
KTable<String, String> customers = builder.table("customers");
KStream<String, String> enrichedVips = vipCustomers
    .leftJoin(customers, (vipInfo, customerInfo) -> {
        if (customerInfo != null) {
            return String.format("%s, \"customerInfo\":%s",
                vipInfo.substring(0, vipInfo.length()-1), customerInfo);
        }
        return vipInfo;
    });

// 7. Outputs
validOrders.to("processed-orders");
hourlySales.toStream()
    .map((windowedKey, sales) ->
        new KeyValue<>(windowedKey.key(),
            String.format("{\"hour\":\"%s\", \"sales\":%.2f}",
                windowedKey.window().startTime(), sales)))
    .to("hourly-sales");
enrichedVips.to("vip-customers");

```

```

// 8. Alertes pour grosses commandes
validOrders
    .filter((key, value) -> {
        try {
            JsonNode order = mapper.readTree(value);
            return order.get("amount").asDouble() > 1000.0;
        } catch (Exception e) {
            return false;
        }
    })
    .to("high-value-alerts");

// Démarrage de l'application
KafkaStreams streams = new KafkaStreams(builder.build(), props);

// Gestion gracieuse de l'arrêt
Runtime.getRuntime().addShutdownHook(new Thread(() -> {
    System.out.println("Arrêt de l'application Streams...");
    streams.close();
}));

streams.start();
System.out.println("Application Kafka Streams démarrée");
}

private static boolean isValidOrder(String orderJson) {
    try {
        JsonNode order = mapper.readTree(orderJson);
        return order.has("customerId") &&
            order.has("amount") &&
            order.get("amount").asDouble() > 0;
    } catch (Exception e) {
        return false;
    }
}

private static String enrichOrder(String orderJson) {
    try {
        JsonNode order = mapper.readTree(orderJson);
        String enriched = String.format("%s,\"processedAt\":%d,\"status\":\"validated\"",

```



```

        orderJson.substring(0, orderJson.length()-1),
        System.currentTimeMillis());

    return enriched;
} catch (Exception e) {
    return orderJson;
}
}
}

```

## Python - Stream Processing Simulé

```

from kafka import KafkaConsumer, KafkaProducer
import json
import threading
import time
from collections import defaultdict, deque
from datetime import datetime, timedelta
import logging

class KafkaStreamsProcessor:
    def __init__(self, bootstrap_servers=['localhost:9092']):
        self.bootstrap_servers = bootstrap_servers
        self.running = False

        # Consumers pour différents topics
        self.consumers = {}

        # Producer pour les outputs
        self.producer = KafkaProducer(
            bootstrap_servers=bootstrap_servers,
            value_serializer=lambda v: json.dumps(v).encode('utf-8')
        )

        # États pour les agrégations
        self.hourly_sales = defaultdict(float)
        self.customer_counts = defaultdict(int)
        self.window_data = defaultdict(lambda: deque(maxlen=1000))

        # Lock pour thread safety
        self.lock = threading.Lock()

```

```
def create_consumer(self, topic, group_id):
    """Crée un consumer pour un topic spécifique"""
    return KafkaConsumer(
        topic,
        bootstrap_servers=self.bootstrap_servers,
        group_id=group_id,
        auto_offset_reset='earliest',
        enable_auto_commit=True,
        value_deserializer=lambda m: json.loads(m.decode('utf-8'))
    )

def process_orders(self):
    """Traite le stream des commandes"""
    consumer = self.create_consumer('orders', 'orders-processor')

    for message in consumer:
        if not self.running:
            break

        try:
            order = message.value

            # 1. Validation et filtrage
            if self.is_valid_order(order):
                # 2. Enrichissement
                enriched_order = self.enrich_order(order)

                # 3. Envoi vers topic traité
                self.producer.send('processed-orders', enriched_order)

                # 4. Agrégations
                self.update_aggregations(enriched_order)

                # 5. Détection alertes
                if order.get('amount', 0) > 1000:
                    alert = {
                        'type': 'HIGH_VALUE_ORDER',
                        'order_id': order.get('id'),
                        'amount': order.get('amount'),
```

```

        'customer_id': order.get('customerId'),
        'timestamp': time.time()
    }
    self.producer.send('high-value-alerts', alert)

except Exception as e:
    logging.error(f"Erreur traitement commande: {e}")

consumer.close()

def process_customers(self):
    """Traite le stream des clients (pour jointures)"""
    consumer = self.create_consumer('customers', 'customers-processor')
    customer_data = {}

    for message in consumer:
        if not self.running:
            break

        try:
            customer = message.value
            customer_id = customer.get('id')

            if customer_id:
                with self.lock:
                    customer_data[customer_id] = customer

            # Vérifier si client devient VIP
            if customer_id in self.customer_counts:
                if self.customer_counts[customer_id] > 10:
                    vip_info = {
                        'customerId': customer_id,
                        'orderCount': self.customer_counts[customer_id],
                        'status': 'VIP',
                        'customerInfo': customer,
                        'promotedAt': time.time()
                    }
                    self.producer.send('vip-customers', vip_info)

        except Exception as e:

```

```
        logging.error(f"Erreur traitement client: {e}")

    consumer.close()

def is_valid_order(self, order):
    """Valide une commande"""
    return (order.get('customerId') and
            order.get('amount', 0) > 0 and
            order.get('id'))

def enrich_order(self, order):
    """Enrichit une commande"""
    enriched = order.copy()
    enriched.update({
        'processedAt': time.time(),
        'status': 'validated',
        'processingNode': 'stream-processor-1'
    })
    return enriched

def update_aggregations(self, order):
    """Met à jour les agrégations"""
    with self.lock:
        # Agrégation par heure
        current_hour = datetime.now().replace(minute=0, second=0, microsecond=0)
        hour_key = current_hour.isoformat()
        self.hourly_sales[hour_key] += order.get('amount', 0)

        # Comptage par client
        customer_id = order.get('customerId')
        if customer_id:
            self.customer_counts[customer_id] += 1

        # Fenêtre glissante (5 minutes)
        window_key = datetime.now().replace(second=0, microsecond=0)
        self.window_data[window_key].append({
            'amount': order.get('amount', 0),
            'timestamp': time.time()
        })
```

```

def publish_aggregations(self):
    """Publie périodiquement les agrégations"""
    while self.running:
        time.sleep(60) # Toutes les minutes

        try:
            with self.lock:
                # Publier les ventes par heure
                for hour, sales in self.hourly_sales.items():
                    sales_data = {
                        'hour': hour,
                        'sales': sales,
                        'timestamp': time.time()
                    }
                    self.producer.send('hourly-sales', sales_data)

                # Nettoyer les anciennes données
                cutoff = datetime.now() - timedelta(days=1)
                keys_to_remove = [k for k in self.hourly_sales.keys()
                                   if datetime.fromisoformat(k) < cutoff]
                for key in keys_to_remove:
                    del self.hourly_sales[key]

        except Exception as e:
            logging.error(f"Erreur publication agrégations: {e}")

def start(self):
    """Démarrage le processeur de streams"""
    self.running = True

    print("Démarrage du processeur Kafka Streams...")

    # Threads pour différents processeurs
    threads = [
        threading.Thread(target=self.process_orders, daemon=True),
        threading.Thread(target=self.process_customers, daemon=True),
        threading.Thread(target=self.publish_aggregations, daemon=True)
    ]

    for thread in threads:

```

```

        thread.start()

    return threads

def stop(self):
    """Arrête le processeur"""
    print("Arrêt du processeur Streams...")
    self.running = False
    self.producer.close()

# Utilisation
if __name__ == "__main__":
    processor = KafkaStreamsProcessor()

    try:
        threads = processor.start()

        # Garder l'application en vie
        while True:
            time.sleep(1)

    except KeyboardInterrupt:
        processor.stop()
        print("Processeur arrêté")

```

# KSQldb

## Types de Données

Type KSQL	Description	Exemple
BIGINT	Entier 64 bits	user_id BIGINT
VARCHAR	Chaîne de caractères	name VARCHAR
DOUBLE	Nombre à virgule flottante	price DOUBLE
BOOLEAN	Booléen	is_active BOOLEAN
ARRAY<TYPE>	Tableau	tags ARRAY<VARCHAR>
MAP<TYPE, TYPE>	Map/Dictionnaire	metadata MAP<VARCHAR, VARCHAR>
STRUCT<...>	Structure	address STRUCT<street VARCHAR, city VARCHAR>

## Commandes DDL

Commande	Description	Exemple
CREATE STREAM	Créer un stream	CREATE STREAM events (...) WITH (...)
CREATE TABLE	Créer une table	CREATE TABLE users (...) WITH (...)
DROP STREAM	Supprimer un stream	DROP STREAM events DELETE TOPIC
DROP TABLE	Supprimer une table	DROP TABLE users DELETE TOPIC

## Requêtes et Transformations

Opération	Syntaxe	Exemple
Sélection	SELECT ... FROM ...	SELECT user_id, action FROM events
Filtrage	WHERE condition	WHERE action = 'login'
Transformation	UCASE() , LCASE() , etc.	SELECT UCASE(name) FROM users
Fenêtrage	WINDOW TUMBLING/HOPPING	WINDOW TUMBLING (SIZE 1 HOUR)
Agrégation	COUNT() , SUM() , AVG()	SELECT COUNT(*) FROM events GROUP BY user_id
Jointure	JOIN , LEFT JOIN	FROM stream1 s JOIN table1 t ON s.id = t.id

## Fonctions Intégrées

Catégorie	Fonctions	Exemple
String	UCASE , LCASE , SUBSTRING	UCASE(name)
Math	ABS , ROUND , CEIL	ROUND(price, 2)
Date/Time	TIMESTAMPToString , STRINGToTimestamp	TIMESTAMPToString(ts, 'yyyy-MM-dd')
Array	ARRAY_LENGTH , ARRAY_CONTAINS	ARRAY_LENGTH(tags)
Map	MAP_KEYS , MAP_VALUES	MAP_KEYS(metadata)
JSON	EXTRACTJSONFIELD , JSON_ARRAY_LENGTH	EXTRACTJSONFIELD(data, '\$.name')








## Configuration KSQLDB

Propriété	Description	Valeur par défaut
ksql.streams.bootstrap.servers	Serveurs Kafka	localhost:9092
ksql.streams.application.id	ID application Streams	Auto-généré

Propriété	Description	Valeur par défaut
<code>ksql.streams.auto.offset.reset</code>	Position initiale	<code>latest</code>
<code>ksql.streams.processing.guarantee</code>	Garantie traitement	<code>at_least_once</code>
<code>ksql.service.id</code>	ID du service KSQL	<code>default_</code>

# Kafka Management

## Monitoring et Métriques

Métrique	Description	JMX Bean	Importance
Messages/sec	Débit de messages	<code>kafka.server:type=BrokerTopicMetrics</code>	 Critique
Bytes/sec	Débit en octets	<code>MessagesInPerSec</code> , <code>BytesInPerSec</code>	 Critique
Request Latency	Latence des requêtes	<code>kafka.network:type=RequestMetrics</code>	 Critique
Under Replicated	Partitions sous-répliquées	<code>kafka.server:type=ReplicaManager</code>	 Critique
Leader Election	Élections de leader	<code>kafka.controller:type=ControllerStats</code>	 Important
Log Size	Taille des logs	<code>kafka.log:type=Log,name=Size</code>	 Important
Consumer Lag	Retard des consumers	<code>kafka.consumer:type=consumer-fetch-manager-metrics</code>	 Critique

## Commandes d'Administration

Opération	Commande	Description
Rebalance	<code>kafka-reassign-partitions.sh</code>	Rééquilibrer les partitions
Preferred Replica	<code>kafka-preferred-replica-election.sh</code>	Élection du leader préféré
ACL	<code>kafka-acls.sh</code>	Gestion des permissions
Quotas	<code>kafka-configs.sh --entity-type clients</code>	Quotas par client
Mirror Maker	<code>kafka-mirror-maker.sh</code>	Réplication entre clusters

## Configuration Broker Critique



Paramètre	Description	Production	Développement
num.network.threads	Threads réseau	8	3
num.io.threads	Threads I/O	8	8
socket.send.buffer.bytes	Buffer TCP send	102400	102400
socket.receive.buffer.bytes	Buffer TCP receive	102400	102400
socket.request.max.bytes	Taille max requête	104857600	104857600
log.retention.hours	Rétention par défaut	168 (7j)	24 (1j)
log.segment.bytes	Taille segment	1073741824 (1GB)	536870912 (512MB)
log.retention.check.interval.ms	Check rétention	300000 (5min)	60000 (1min)
zookeeper.connection.timeout.ms	Timeout ZK	18000	6000

## Sécurité

Mécanisme	Configuration	Description
SSL/TLS	security.protocol=SSL	Chiffrement transport
SASL	security.protocol=SASL_SSL	Authentification
ACL	authorizer.class.name	Autorisation
Quotas	quota.producer.default	Limitation débit

## Outils de Management

Outil	Description	Use Case
Kafka Manager	Interface web Yahoo	Monitoring simple
Confluent Control Center	Interface Confluent	Entreprise complète
Kafdrop	Interface web légère	Développement
Kafka Tool	Client GUI	Debug/Admin
Burrow	Consumer lag monitoring	Alerting

---

## Configurations Importantes

### Producer Performance

Configuration	Throughput	Latence	Fiabilité
acks=0	★ ★ ★	★ ★ ★	✖
acks=1	★ ★	★ ★	★
acks=all	★	★	★ ★ ★
batch.size=16384	★	★ ★	★ ★
batch.size=32768	★ ★	★	★ ★
linger.ms=0	★	★ ★ ★	★ ★
linger.ms=5	★ ★ ★	★	★ ★

## Consumer Performance

Configuration	Throughput	Latence	Description
fetch.min.bytes=1	★	★ ★ ★	Réactivité maximale
fetch.min.bytes=1024	★ ★	★ ★	Équilibré
fetch.max.wait.ms=500	★ ★	★ ★	Standard
max.poll.records=500	★ ★	★ ★	Par défaut
max.poll.records=100	★	★ ★ ★	Traitement rapide

## Patterns et Bonnes Pratiques

### Patterns de Messagerie

Pattern	Description	Avantages	Inconvénients
Fire and Forget	Envoi sans attendre	Performance max	Perte possible
Request-Reply	Attente réponse	Fiabilité	Couplage fort
Event Sourcing	Journal d'événements	Auditabilité	Complexité
CQRS	Séparation lecture/écriture	Scalabilité	Architecture complexe

### Anti-Patterns à Éviter

Anti-Pattern	Problème	Solution
Shared Consumer Group	Plusieurs apps même groupe	Groupe par application
Large Messages	Messages > 1MB	Référence externe

Anti-Pattern	Problème	Solution
Synchronous Processing	Blocage sur traitement	Processing asynchrone
No Error Handling	Perte de messages	DLQ + retry
Auto-commit sans contrôle	Perte ou doublon	Commit manuel

## Checklist Production

Élément	✅ Check	Description
Réplication	<code>min.insync.replicas &gt;= 2</code>	Haute disponibilité
Monitoring	Métriques JMX	Observabilité
Backup	Stratégie sauvegarde	Disaster recovery
Security	SSL + SASL	Sécurité
Capacity Planning	Sizing approprié	Performance
Error Handling	DLQ + alertes	Fiabilité
Testing	Tests end-to-end	Qualité

## Formules de Sizing

Métrique	Formule	Exemple
Partitions	<code>max(throughput/partition_throughput, consumers)</code>	<code>max(100MB/s / 10MB/s, 20) = 20</code>
Réplication Factor	<code>min(brokers, 3)</code>	<code>min(5, 3) = 3</code>
Retention	<code>throughput × retention_time</code>	<code>10MB/s × 7j = 6TB</code>
Memory	<code>partitions × 1MB + JVM_heap</code>	<code>1000 × 1MB + 4GB = 5GB</code>

---