

# Confluent Advanced Skills for Optimizing Apache Kafka®

Version 6.0.0-v2.0.1



# CONFLUENT

# Table of Contents

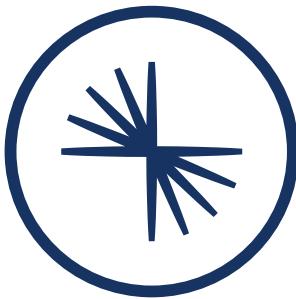
<b>Introduction</b>	1
<b>Branch 1: Monitoring - Overview</b>	10
<b>01: Business Needs and SLAs</b>	12
a: What are the Pillars of Optimization and How Do You Measure?	14
<b>02: Monitoring Basics</b>	21
a: Monitoring Basics	23
<b>03: Generic Monitoring</b>	31
a: Generic Monitoring	33
Lab: Introduction	55
Lab: Monitoring via JMX	56
Lab: Monitoring librdkafka based Clients	57
Lab: Monitoring with Prometheus	58
<b>04: Monitoring with CCC</b>	59
a: Monitoring with Confluent Control Center	61
Lab: Monitoring with CCC - Data Streams	83
<b>Branch 2: General Troubleshooting &amp; Tuning - Overview</b>	84
<b>05: General Troubleshooting</b>	86

a: Troubleshooting Intro	88
b: Production Down! Troubleshooting Strategies	95
c: Troubleshooting Toolbelt	101
Lab: Troubleshooting Toolbelt	109
<b>06: Where are my System Log Files?</b>	<b>110</b>
a: Where are my System Log Files?	112
Lab: Where are my Log Files?	131
<b>Branch 3: Troubleshooting &amp; Tuning Central Services - Overview</b>	<b>132</b>
<b>07: Troubleshooting &amp; Tuning ZooKeeper</b>	<b>134</b>
a: Troubleshooting ZooKeeper	136
Lab: Troubleshooting ZooKeeper	144
<b>08: Troubleshooting &amp; Tuning Brokers</b>	<b>145</b>
a: Troubleshooting Brokers	147
b: Replication Concerns	164
Lab: Troubleshooting Brokers	170
Lab: Not Enough ISRs	171
c: Tuning Brokers: General Concepts & Best Practices	172
d: Optimization of a Message's Life Cycle on a Broker	178
e: Miscellaneous Matters	185
f: Confluent Auto Data Balancer	189

g: Message Delivery Guarantees	193
Lab: Tuning Brokers	200
<b>09: Troubleshooting &amp; Tuning Schema Registry</b>	<b>201</b>
a: Troubleshooting the Schema Registry	203
Lab: Troubleshooting the Schema Registry	216
<b>Branch 4: Troubleshooting &amp; Tuning Clients - Overview</b>	<b>217</b>
<b>10: Troubleshooting &amp; Tuning Producers</b>	<b>219</b>
a: Troubleshooting Producers	221
Lab: Troubleshooting Producers	228
b: Tuning Producers	229
Lab: Tuning Producers	253
Lab: Tuning Producers - Selecting the Best Partition Strategy	254
<b>11: Troubleshooting &amp; Tuning Consumers</b>	<b>255</b>
a: Troubleshooting Consumers	257
Lab: Troubleshooting Consumers	270
b: Tuning Consumers	271
Lab: Tuning Consumers	285
<b>12: Troubleshooting &amp; Tuning Streams Apps</b>	<b>286</b>
a: Troubleshooting Kafka Streams & ksqlDB Apps	288
Lab: Troubleshooting Kafka Streams & ksqlDB Apps	

b: Tuning Kafka Streams & ksqlDB Apps	299
Lab: Tuning Kafka Streams & ksqlDB Apps	313
<b>13: Troubleshooting &amp; Tuning Kafka Connect</b>	<b>314</b>
a: Troubleshooting Kafka Connect	316
b: Tuning Kafka Connect	329
Lab: Tuning Kafka Connect	335
<b>Conclusion</b>	<b>336</b>
<b>Appendix: Additional Problems</b>	<b>343</b>
Problem A: Reviewing Message Sending and Broker Arrival	345
Problem B: Replication Review	346
Problem C: Consumer Offsets and Consumer Lag	347
Problem D: Monitoring Disk Usage	348
Problem E: Assessing Discrepancies in Settings	349
Problem F: Troubleshooting Producers and Consumers	350

# Introduction



# CONFLUENT Global Education

# Class Logistics and Overview

## Copyright & Trademarks

Copyright © Confluent, Inc. 2014-2022. [Privacy Policy](#) | [Terms & Conditions](#).

Apache, Apache Kafka, Kafka, and the Kafka logo are trademarks of the  
[Apache Software Foundation](#)

All other trademarks, product names, and company names or logos cited herein are the property of their respective owners.

## Prerequisite

This course requires a working knowledge of the Apache Kafka architecture.

New to Kafka? Need a refresher?

Sign up for free ***Confluent Fundamentals for Apache Kafka*** course at  
<https://confluent.io/training>

## Additional Prerequisite

This course also requires that you have completed the course Apache Kafka® Administration By Confluent.

An understanding of the content from that course is assumed.

Some content in this course will review concepts from this prerequisite, but you will experience the greatest success in this course if you have completed this prerequisite.

# Agenda



This course consists of these major parts:

- Monitoring
- General Troubleshooting & Tuning
- Troubleshooting & Tuning Central Services
  - ZooKeeper, Brokers, Schema Registry
- Troubleshooting & Tuning Clients
  - Producers, Consumers, Streams Apps, Kafka Connect

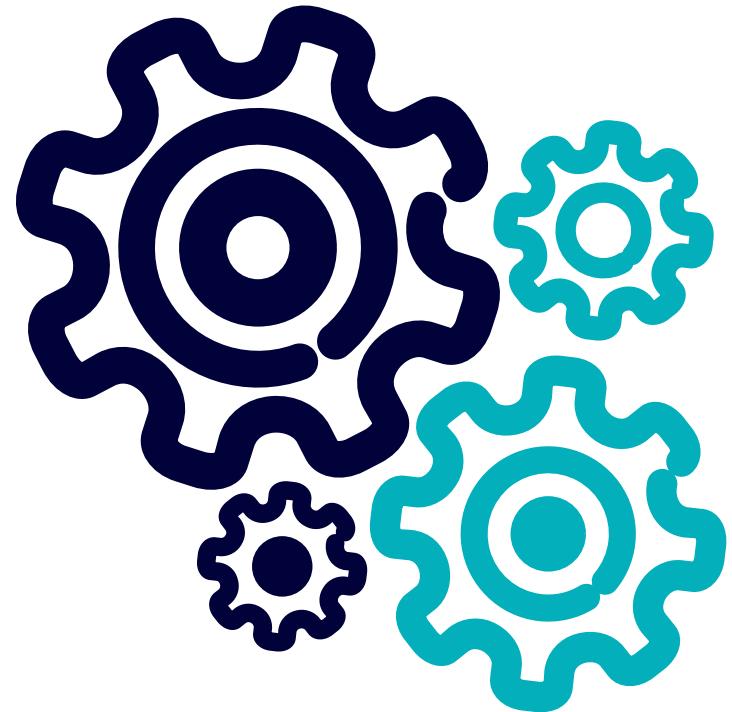
## Course Objectives

Upon completion of this course, you should be able to:

- Formulate the Apache Kafka® Confluent Platform specific needs of your company
- Monitor all essential aspects of your Confluent Platform
- Tune the Confluent Platform according to your specific needs
- Provide first level production support for your Confluent Platform

Throughout the course, Hands-On Exercises will reinforce the topics being discussed.

# Class Logistics



- Timing
  - Start and end times
  - Can I come in early/stay late?
  - Breaks
  - Lunch
- Physical Class Concerns
  - Restrooms
  - Wi-Fi and other information
  - Emergency procedures
  - Don't leave belongings unattended



No recording, please!

## How to get the courseware?



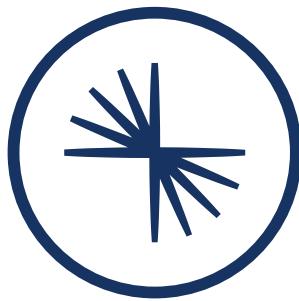
1. Register at **training.confluent.io**
2. Verify your email
3. Log in to **training.confluent.io** and enter your **license activation key**
4. Go to the **Classes** dashboard and select your class

# Introductions



- About you:
  - What is your name, your company, and your role?
  - Where are you located (city, timezone)?
  - What is your experience with Kafka?
  - Which other Confluent courses have you attended, if any?
  - Optional talking points:
    - What are some other distributed systems you like to work with?
    - What technology most excited you early in your life?
    - Anything else you want to share?
- About your instructor

# Branch 1: Monitoring - Overview



CONFLUENT  
**Global Education**

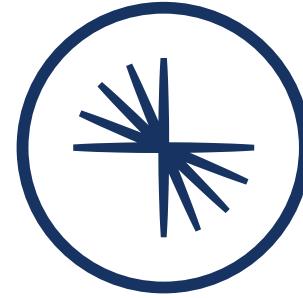
# Agenda



This is a branch of our CAO content on monitoring.  
It is broken down into the following modules:

1. Business Needs & SLAs
2. Monitoring Basics
3. Generic Monitoring
4. Monitoring with CCC

# 01: Business Needs and SLAs



CONFLUENT  
**Global Education**

# Module Overview



This module contains one lesson:

1. What are the Pillars of Optimization and How Do You Measure?

## a: What are the Pillars of Optimization and How Do You Measure?

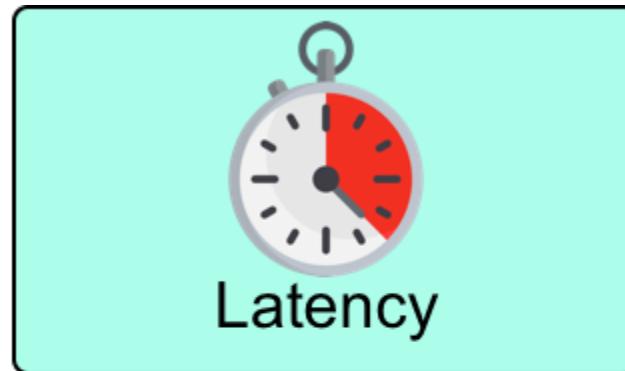
### Description

Establishing what is meant by throughput, latency, durability, availability. SLIs, SLOs/SLTs, SLAs.

# Deciding Which Service Criteria to Optimize



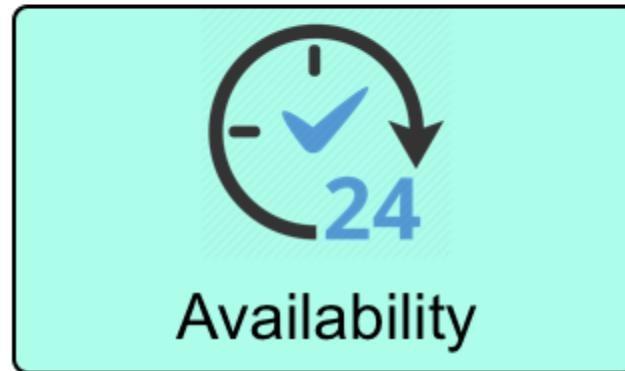
Throughput



Latency



Durability



Availability

## Measuring Aspects of Performance

There are a few different measurements people use to describe performance:

- Service Level Indicators
- Service Level Thresholds a.k.a Service Level Objectives
- Service Level Agreements

We'll look at each in turn and build up...

## Service Level Indicators (SLIs)

Metric describing one aspect of a service's reliability; make objective & measurable

**Example:** Overall throughput

## Service Level Objectives a.k.a. Thresholds (SLOs / SLTs)

SLI with target value

**Example:** Throughput > 200 MB/sec

## Service Level Agreements (SLAs)

- Contract between service provider and client
- Usu. several SLOs, interaction agreements
- Usu. what happens if not met

### Example:

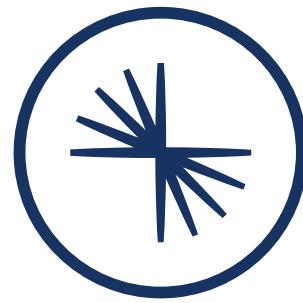


- **Availability:** > 99.95%
- **Throughput:** > 200 MB/sec
- **Latency:** < 50 ms (95th percentile)
- **Durability:** Exactly Once Semantics
- **Retention:** 1 year
- **Cost:**
- **Monitoring:**
- etc.

## Further Reading

- [Kafka the Definitive Guide](#), v2

## 02: Monitoring Basics



CONFLUENT  
**Global Education**

# Module Overview



This module contains one lesson:

## 1. Monitoring Basics

## a: Monitoring Basics

### Description

Motivation for and basics of monitoring.

## Motivation

You cannot fix problems you cannot see!

## What Does Your Business Care For?

Are all of your services behaving properly and meeting SLAs?

Questions you need to be able to answer:

- Are applications **receiving all data**?
- Are my business applications showing the **latest data**?
- Why are the applications **running slowly**?
- Do we need to **scale up**?
- Can any **data get lost**?
- Will there be **service interruptions**?
- Are there assurances in case of a **disaster event**?

## What's Important to You?

- Message retention
- Message throughput
- Producer performance
- Consumer performance
- Availability
- Durability



## Monitoring the Foundation

- CPU load
- Network inbound and outbound
- File handle usage for Kafka
- Memory utilization
- Disk
- Garbage collection

## The Metrics Swamp

- Hundreds of metrics per broker available
- Cannot monitor all
- Will concentrate on most critical ones



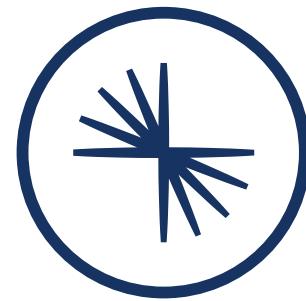
### Question:

What is your company's highest priority, throughput or minimal latency?

## Further Reading

- Monitoring Your Apache Kafka® Deployment End-to-End:  
<https://www.confluent.io/monitoring-your-apache-kafka-deployment>
- The Blog Post on Monitoring an Apache Kafka Deployment to End Most Blog Posts:  
<https://www.confluent.io/blog/blog-post-on-monitoring-an-apache-kafka-deployment-to-end-most-blog-posts>
- Kafka Protocol Guide:  
<http://kafka.apache.org/protocol.html>

## 03: Generic Monitoring



CONFLUENT  
**Global Education**

# Module Overview



This module contains one lessons:

1. Generic Monitoring

## a: Generic Monitoring

### Description

JMX. Other relevant monitoring tools. Early considerations for monitoring various clients.

# Monitoring JMX Metrics

- CLI tools: jmxterm
- GUI tools: jconsole
- Use Grafana, Graphite, CloudWatch, Datadog, etc.



## Monitoring ZooKeeper - The Four Letter Words

- ZooKeeper emits operational data in response to a limited set of commands known as "the four letter words"
  - Usage: `echo "<command>" | nc <host> <port>`
  - `mntr` command response includes the following metrics:

<code>zk_outstanding_requests</code>	Number of requests queued
<code>zk_avg_latency</code>	Amount of time it takes to respond to a client request (in ms)
<code>zk_num_alive_connections</code>	Number of clients connected to ZooKeeper
<code>zk_followers</code>	Number of active followers
<code>zk_pending_syncs</code>	Number of pending syncs from followers
<code>zk_open_file_descriptor_count</code>	Number of file descriptors in use

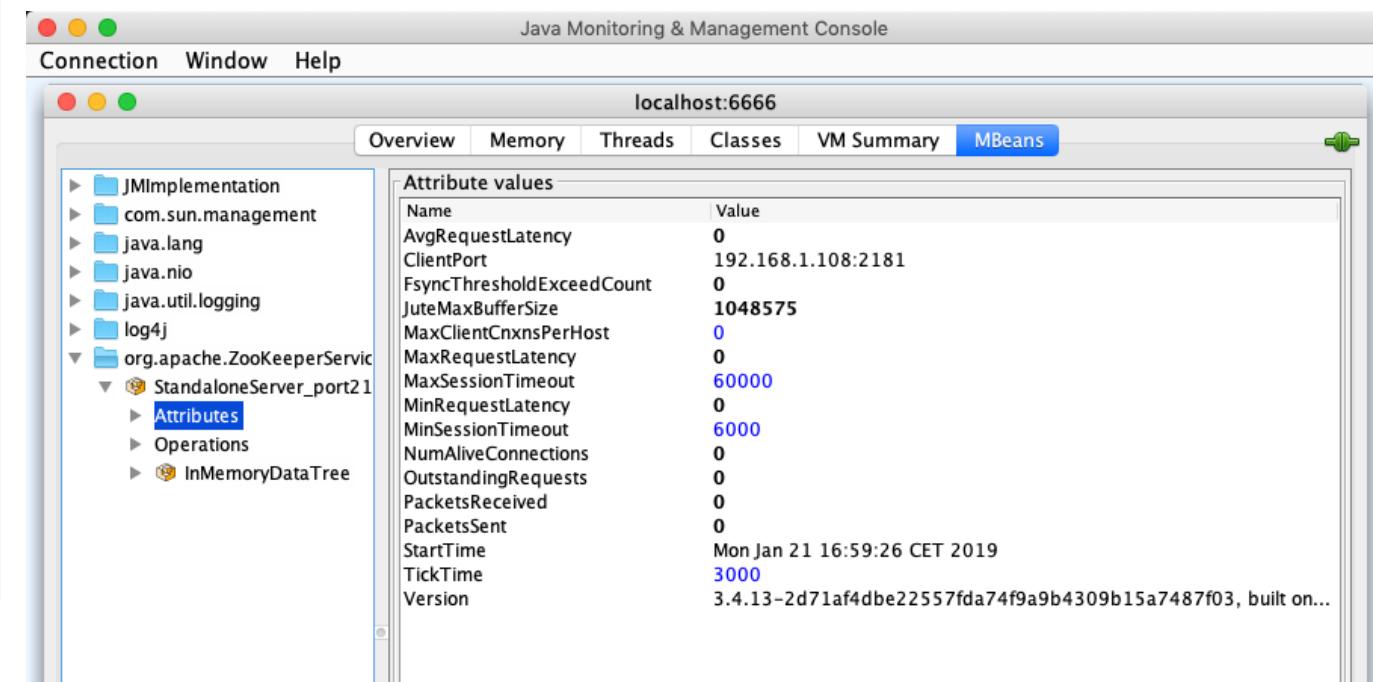
# Monitoring ZooKeeper - JMX Metrics

## Enable Monitoring (Docker):

```
zookeeper:  
  image: confluentinc/cp-zookeeper:6.0.0-1-ubi8  
  hostname: zookeeper  
  networks:  
    - confluent  
  ports:  
    - 9999:9999  
  environment:  
    ZOOKEEPER_CLIENT_PORT: 2181  
    ZOOKEEPER_TICK_TIME: 2000  
    KAFKA_JMX_PORT: 9999  
    KAFKA_JMX_HOSTNAME: 127.0.0.1
```

## Enable Monitoring (native):

```
export ZOOKEEPER_JMX_PORT=9999  
bin/confluent start zookeeper
```



# Monitoring ZooKeeper - UI

## Docker:

```
docker container run -it --rm \
--net <network name> \
-p 8080:8080 \
goodguide/zk-web
```

## Native:

```
git clone git://github.com/qiuxiaafei/zk-web.git
cd zk-web
lein deps # run this if you're using lein 1.x
lein run
```

The screenshot shows a web browser window titled "ZK-Web | Make zookeeper simpler" with the URL "localhost:8080/node?path=/". The page displays a table of node statistics. The left column lists node names, and the right column lists their corresponding values.

Children	Node Stat
schema_registry	numChildren: 12
cluster	ephemeralOwner: 0
controller	cversion: 10
controller_epoch	mzxid: 0
brokers	czxid: 0
zookeeper	dataLength: 0
admin	ctime: 0
isr_change_notification	version: 0
consumers	aversion: 0
log_dir_event_notification	mtime: 0
latest_producer_id_block	pzxid: 54
config	

## Monitoring Brokers - System Metrics

### Observe:

- CPU usage
- Memory usage
- Available disk space
- Disk IO
- Network IO
- Open file handles

### Alert:

- 60% disk usage for disks
- 60% disk IO usage
- 60% network IO usage
- 60% file handle usage

## Kinds of JMX Metrics

There are two classes of JMX metrics:

- **guage** - a measure of something *right now*
  - e.g., number of offline partitions
- **meter** - a measure of something over a time sample
  - e.g., throughput

## More on Meter Metrics

"Over a time sample" really means over a set of time samples.

So how can we control that?

All meter metrics can be configured by these properties:

Name	Meaning	Default
<code>metrics.sample.window.ms</code>	Size of each sample window	30 sec.
<code>metrics.num.samples</code>	Number of samples maintained and included in reports	2

# Monitoring Brokers - JMX Metrics (1)

## Broker Load:

kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec

kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec

kafka.network:type=RequestMetrics,name=RequestsPerSec,request=<type>

kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec

## Monitoring Brokers - JMX Metrics (2)

### Enable Monitoring (native):

```
# Add to /etc/kafka/kafka.properties:  
jmx.port=9999  
jmx.hostname=127.0.0.1  
  
# Start Kafka  
bin/kafka-server-start.sh
```

### Enable Monitoring (Docker):

```
kafka:  
  image: "confluentinc/cp-enterprise-kafka:6.0.0-1-ubi8"  
  networks:  
    - confluent  
  ports:  
    - 9999:9999  
  environment:  
    KAFKA_BROKER_ID: 101  
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181  
    ...  
    KAFKA_JMX_PORT: 9999  
    KAFKA_JMX_HOSTNAME: 127.0.0.1  
    ...
```

# Monitoring Kafka with Burrow

- Developed by LinkedIn
- Monitor Consumer Lag in Kafka
- HTTP Endpoints to get info about cluster

```
$ curl -s localhost:8000/v3/kafka/local
{
  "error": false,
  "message": "cluster module detail returned",
  "module": {
    "class-name": "kafka",
    "servers": [
      "kafka:9092"
    ],
    "client-profile": {
      "name": "",
      "client-id": "burrow-lagchecker",
      "kafka-version": "0.8",
      "tls": null,
      "sasl": null
    },
    "topic-refresh": 60,
    "offset-refresh": 30
  },
  "request": {
    "url": "/v3/kafka/local",
    "host": "burrow"
  }
}
```

# Monitoring Kafka Clients

Common per-broker Metrics for:

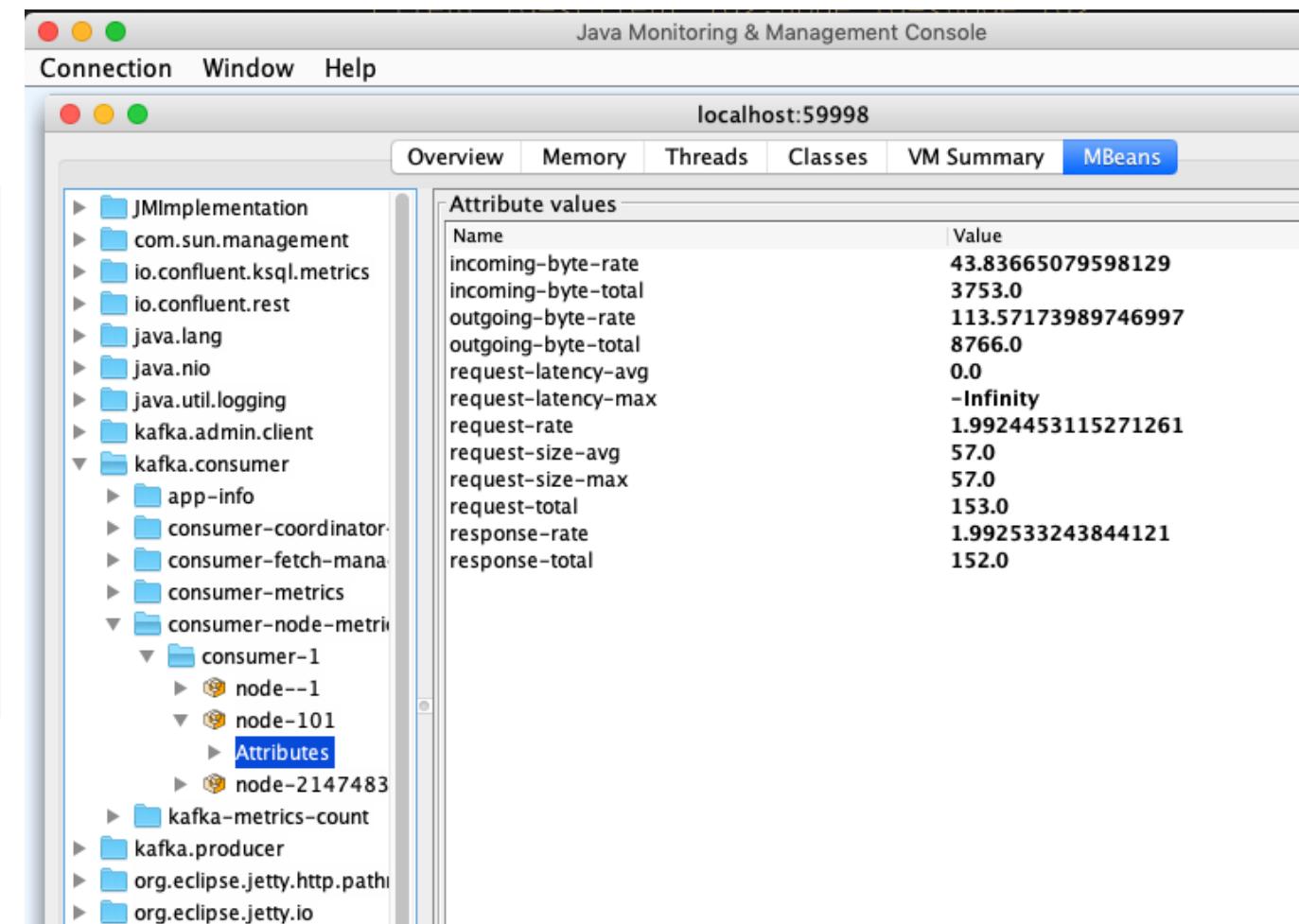
Consumer, Producer, Connect, Kafka Streams, ksqlDB

```
kafka.producer:type=producer-node-metrics,  
client-id=<client-id>,node-id=<node-id>
```

```
kafka.consumer:type=consumer-node-metrics,  
client-id=<client-id>,node-id=<node-id>
```

```
kafka.connect:type=connect-node-metrics,  
client-id=<client-id>,node-id=<node-id>
```

JMX Metrics for ksqlDB Server:



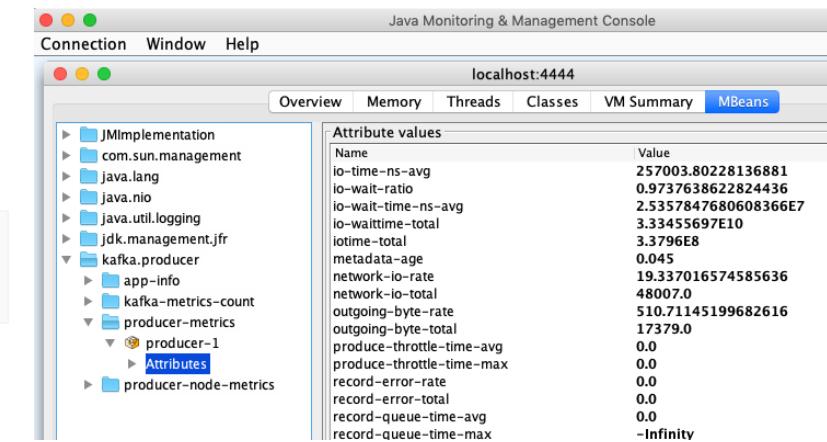
# Monitoring Producers

- Common Kafka Client metrics
- Producer metrics

```
kafka.producer:type=producer-metrics,client-id=<client-id>
```

- Producer Sender metrics
- Key Producer metrics

Response rate	Average number of responses received per second
Request rate	Average number of requests sent per second
Request latency avg	Average request latency (in ms)
Outgoing byte rate	Average number of outgoing/incoming bytes per second
IO wait time ns avg	Average length of time the I/O thread spent waiting for a socket (in ns)



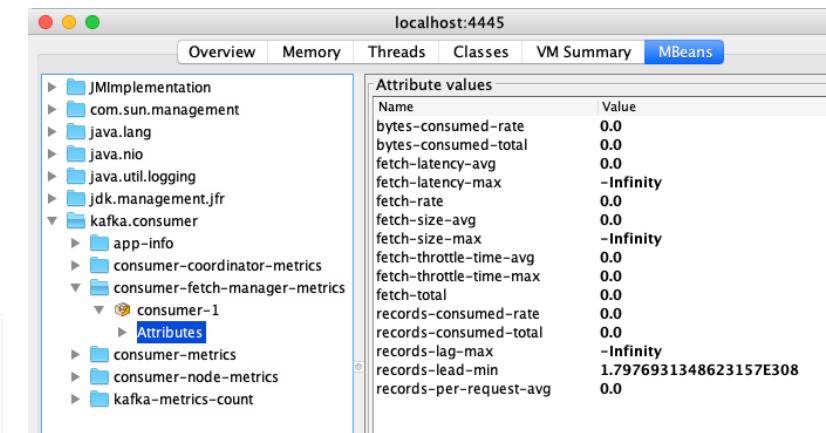
# Monitoring Consumers

- Common Kafka Client metrics
- Consumer Group metrics
- Consumer Fetch metrics

```
kafka.consumer:type=consumer-fetch-manager-metrics,  
client-id=<client-id>
```

- Key Consumer metrics

ConsumerLag	Number of messages consumer is behind producer
MaxLag	Maximum observed value of ConsumerLag
BytesPerSec	Bytes consumed per second
MessagesPerSec	Messages consumed per second
MinFetchRate	Minimum rate a consumer fetches requests to the broker



# Monitoring Consumers - Consumer Lag

Monitor Consumer Lag for real-time apps:

- JMX: `kafka.consumer:type=consumer-fetch-manager-metrics,client-id=<client-id>`  
attribute: `records-lag-max`
- `kafka-consumer-groups` tool

```
$ kafka-consumer-groups \
  --bootstrap-server kafka:9092 \
  --describe \
  --group my-group
```

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	...
my-topic	0	2	4	2	consumer-1-029...	
my-topic	1	2	3	1	consumer-1-029...	
my-topic	2	2	3	1	consumer-2-42c...	

# Monitoring Kafka Connect

## Monitor:

- Common Kafka Client metrics
- Connect specific metrics:  
see: [https://kafka.apache.org/documentation/#connect\\_monitoring](https://kafka.apache.org/documentation/#connect_monitoring)
  - Connector metrics
  - Common task metric
  - Source task metrics
  - Sink task metrics
  - Worker metrics
  - Worker rebalance metrics
- Hosts where workers run
- The source/sink system
- Workers through REST interface

Distinguish standalone vs. distributed workers

# Monitoring Kafka Connect

The screenshot shows the Java Management Interface (JConsole) running on localhost:59997. The 'MBeans' tab is selected. On the left, a tree view shows various MBean categories, with 'kafka.connect' expanded to reveal sub-categories like 'app-info', 'connect-coordinator-metrics', 'connect-metrics', 'connect-node-metrics', 'connect-worker-metrics', 'kafka.consumer', and 'kafka.producer'. Under 'connect-worker-metrics', the 'Attributes' node is selected. On the right, a table titled 'Attribute values' lists several metrics with their current values set to 0.0.

Name	Value
connector-count	0.0
connector-startup-attempts-total	0.0
connector-startup-failure-percentage	0.0
connector-startup-failure-total	0.0
connector-startup-success-percentage	0.0
connector-startup-success-total	0.0
task-count	0.0
task-startup-attempts-total	0.0
task-startup-failure-percentage	0.0
task-startup-failure-total	0.0
task-startup-success-percentage	0.0
task-startup-success-total	0.0

# Monitoring Kafka Streams & ksqlDB Apps

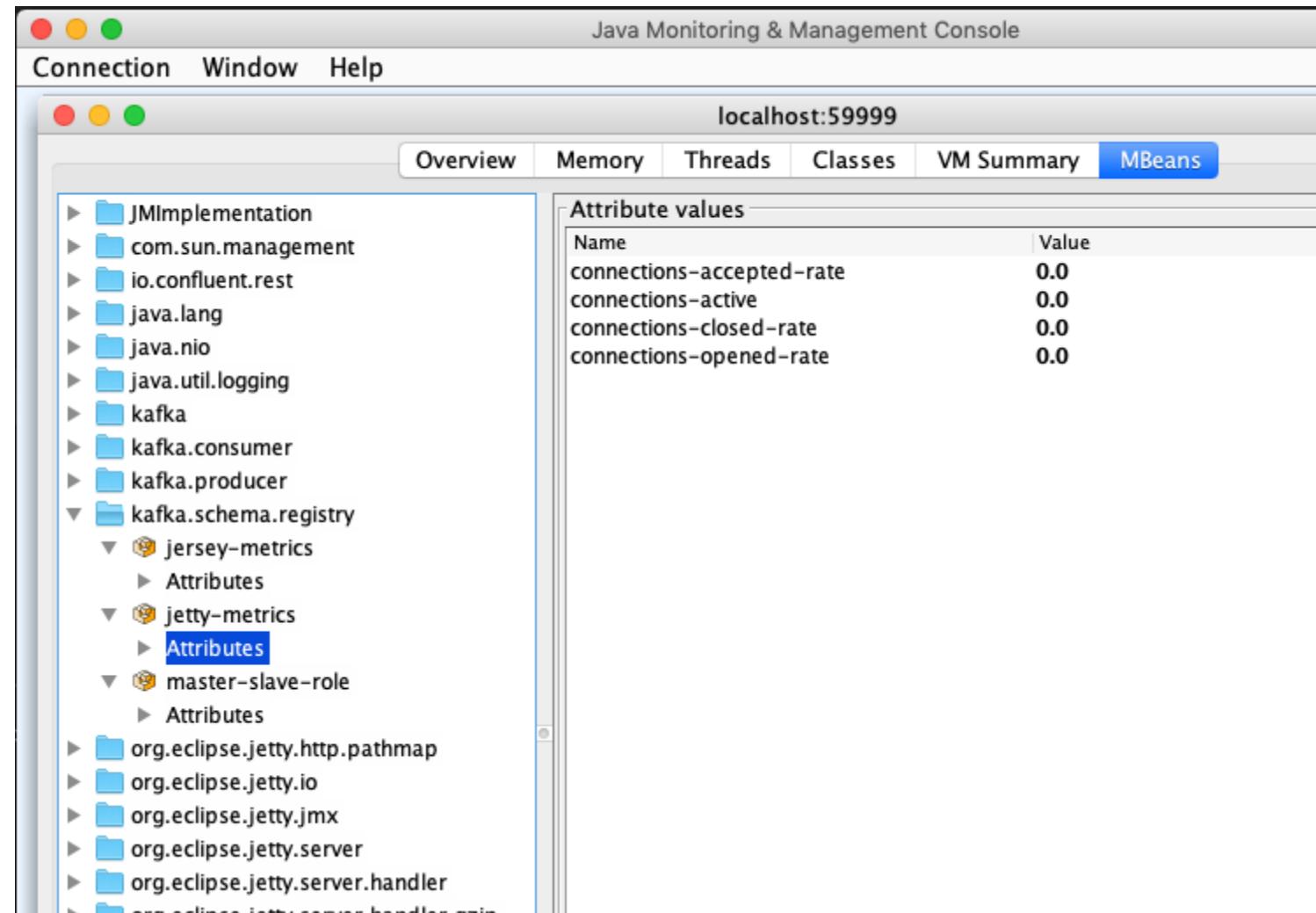
## Monitor:

- Thread metrics
- Task metrics
- Processor Node metrics
- State Store metrics
- Record Cache metrics

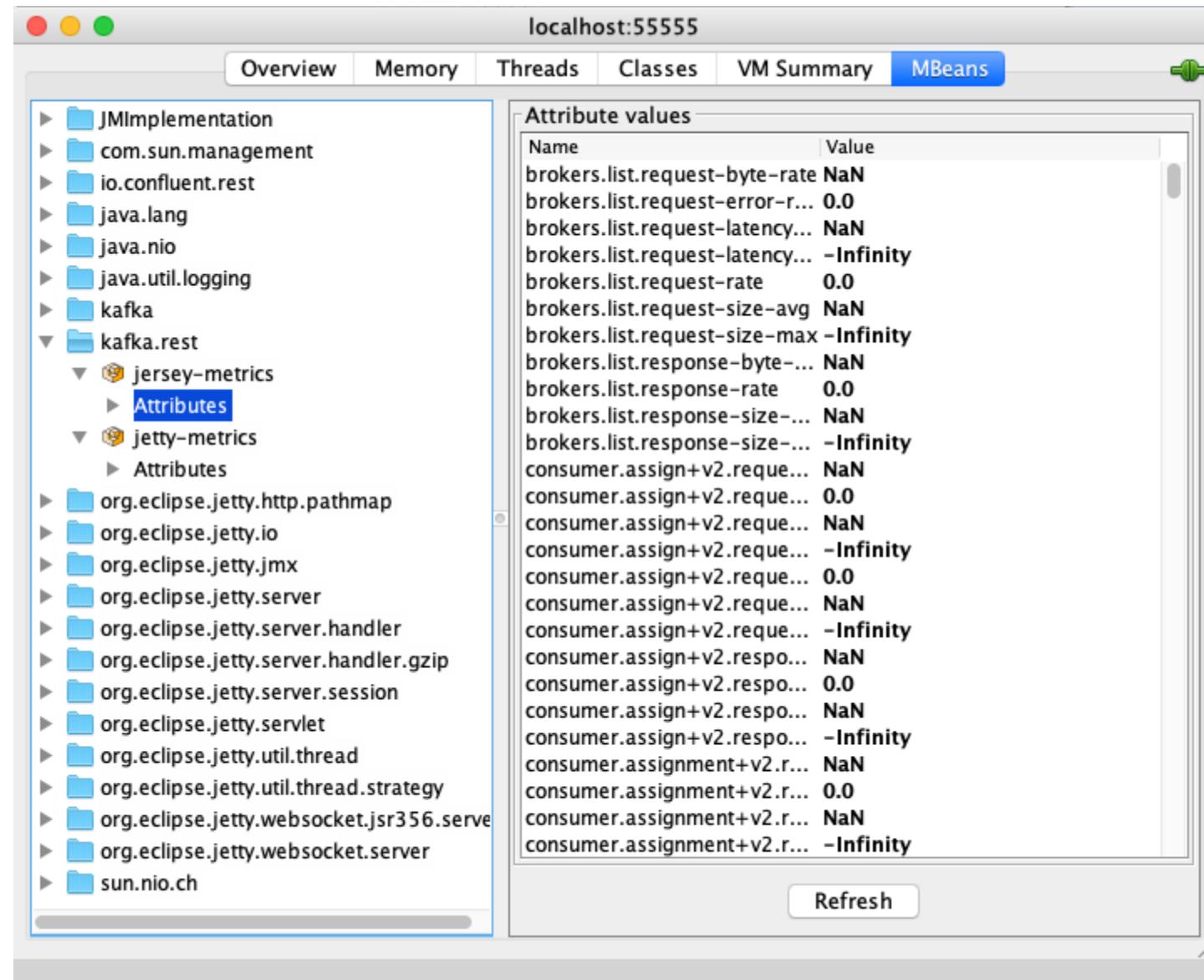


ksqlDB has some additional "debug"-level metrics not enabled by default!

# Monitoring Confluent Schema Registry



# Monitoring Confluent REST Proxy





### Question:

Justify why it **doesn't make sense** to monitor all possible metrics of any component of the Confluent Platform.

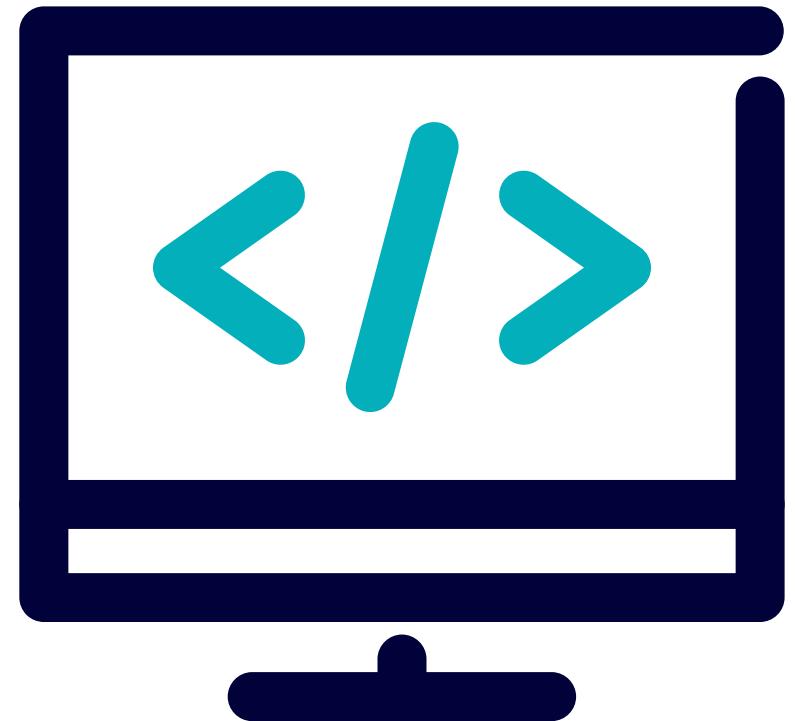
## Further Reading

- Monitoring Kafka:  
<https://docs.confluent.io/current/kafka/monitoring.html>
- JMX Reporters:  
<https://cwiki.apache.org/confluence/display/KAFKA/JMX+Reporters>
- Monitoring Apache Kafka with Grafana / InfluxDB via JMX  
<https://softwaremill.com/monitoring-apache-kafka-with-influxdb-grafana/>
- Monitoring Kafka Streams Metrics via JMX:  
<https://www.madewithtea.com/posts/monitoring-metrics-kafka-streams>
- Monitoring Kafka in Production:  
<https://logz.io/blog/monitoring-kafka-in-production/>
- ZK-Web: <https://github.com/qiuxiafei/zk-web>

# Lab: Introduction

Please work on **Lab 3a: Introduction**

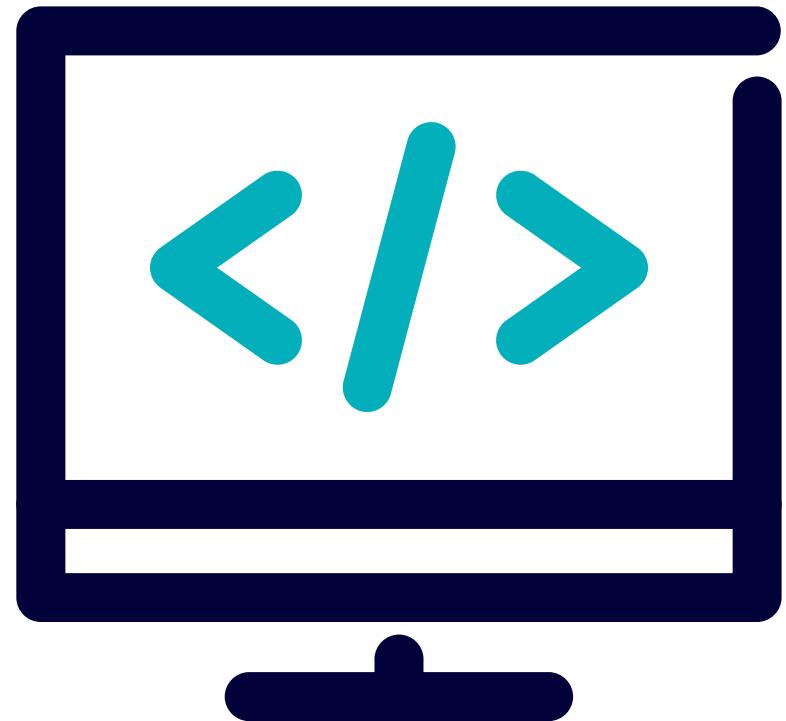
Refer to the Exercise Guide



# Lab: Monitoring via JMX

Please work on **Lab 3b: Monitoring via JMX**

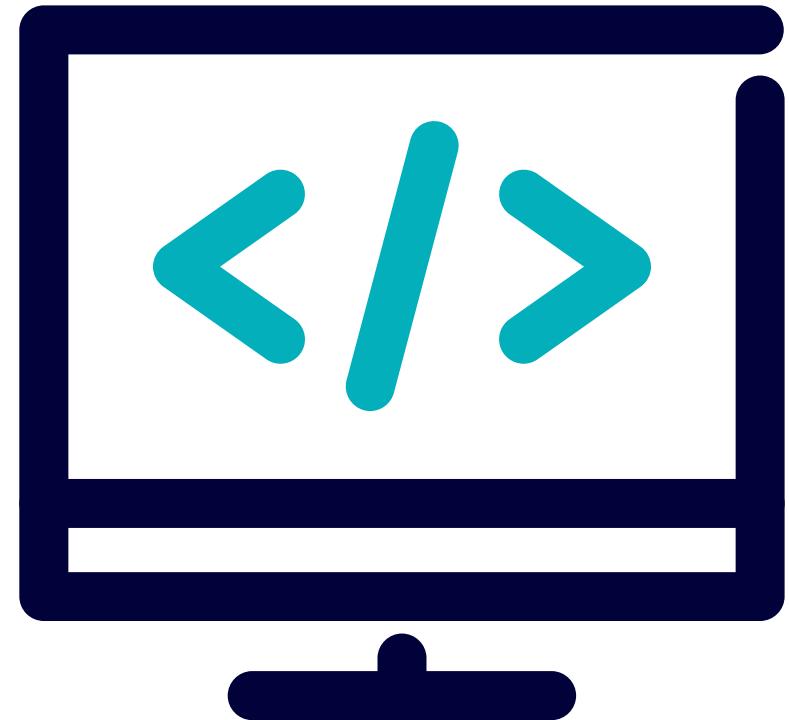
Refer to the Exercise Guide



# Lab: Monitoring librdkafka based Clients

Please work on **Lab 3c: Monitoring librdkafka based Clients**

Refer to the Exercise Guide



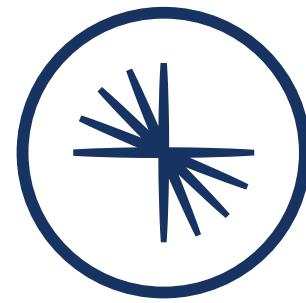
# Lab: Monitoring with Prometheus

Please work on **Lab 3d: Monitoring with Prometheus**

Refer to the Exercise Guide



## 04: Monitoring with CCC



CONFLUENT  
**Global Education**

# Module Overview



This module contains one lesson:

1. Monitoring with CCC

## a: Monitoring with Confluent Control Center

### Description

Tour of CCC and its monitoring capabilities.

# Control Center

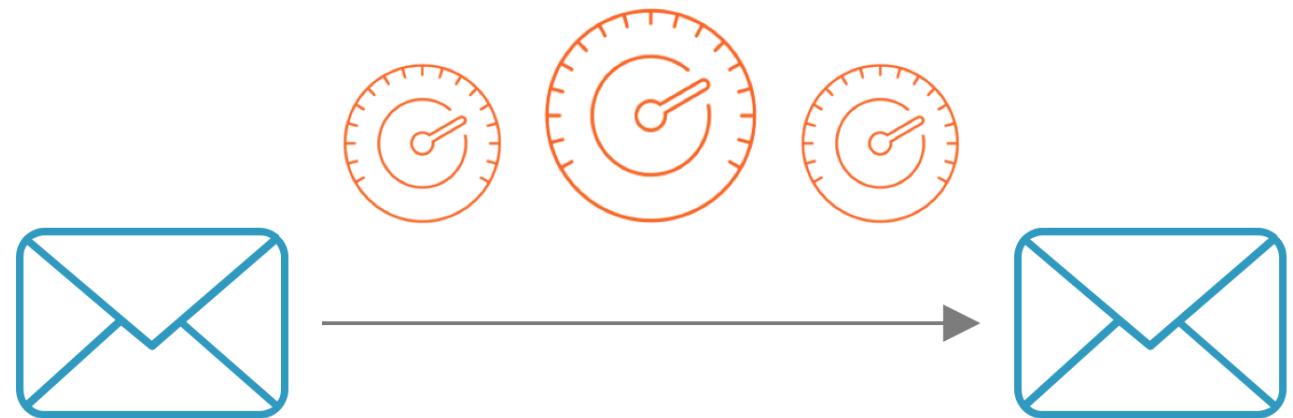
## Expert Kafka Monitoring for the Enterprise

### System Health



Are all brokers and topics **available**?  
**How much data** is being processed?  
What can be tuned to improve  
**performance**?

### End-to-End SLA Monitoring

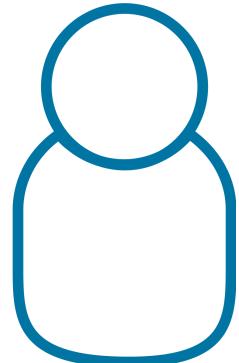
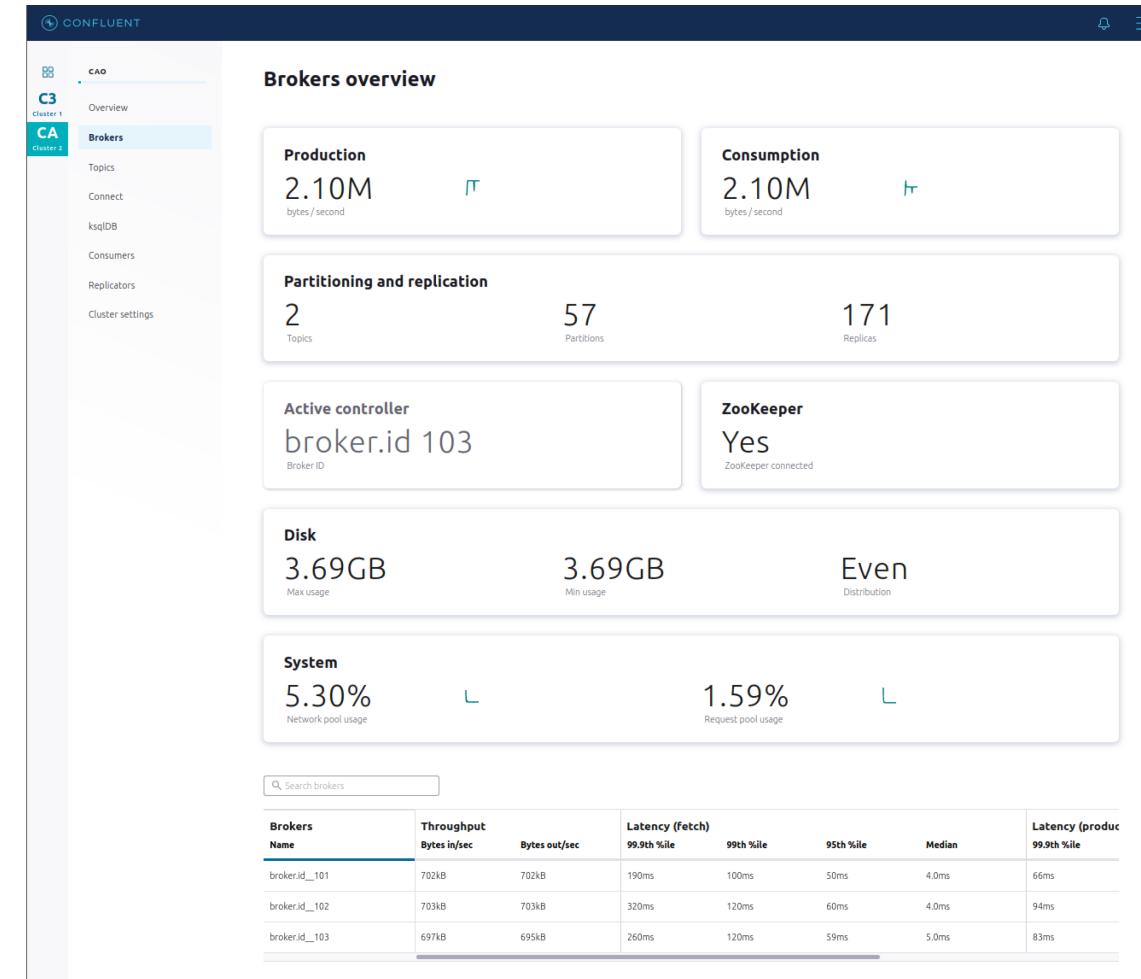


Does Kafka process all events **<15 seconds**?  
Is the 8am report **missing data**?  
Are there **duplicate** events?

# Control Center

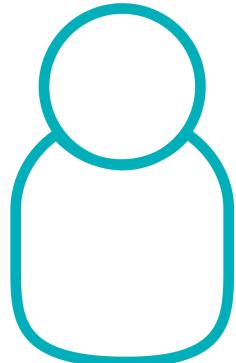


{dev}



{ops}

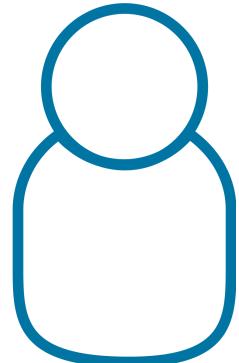
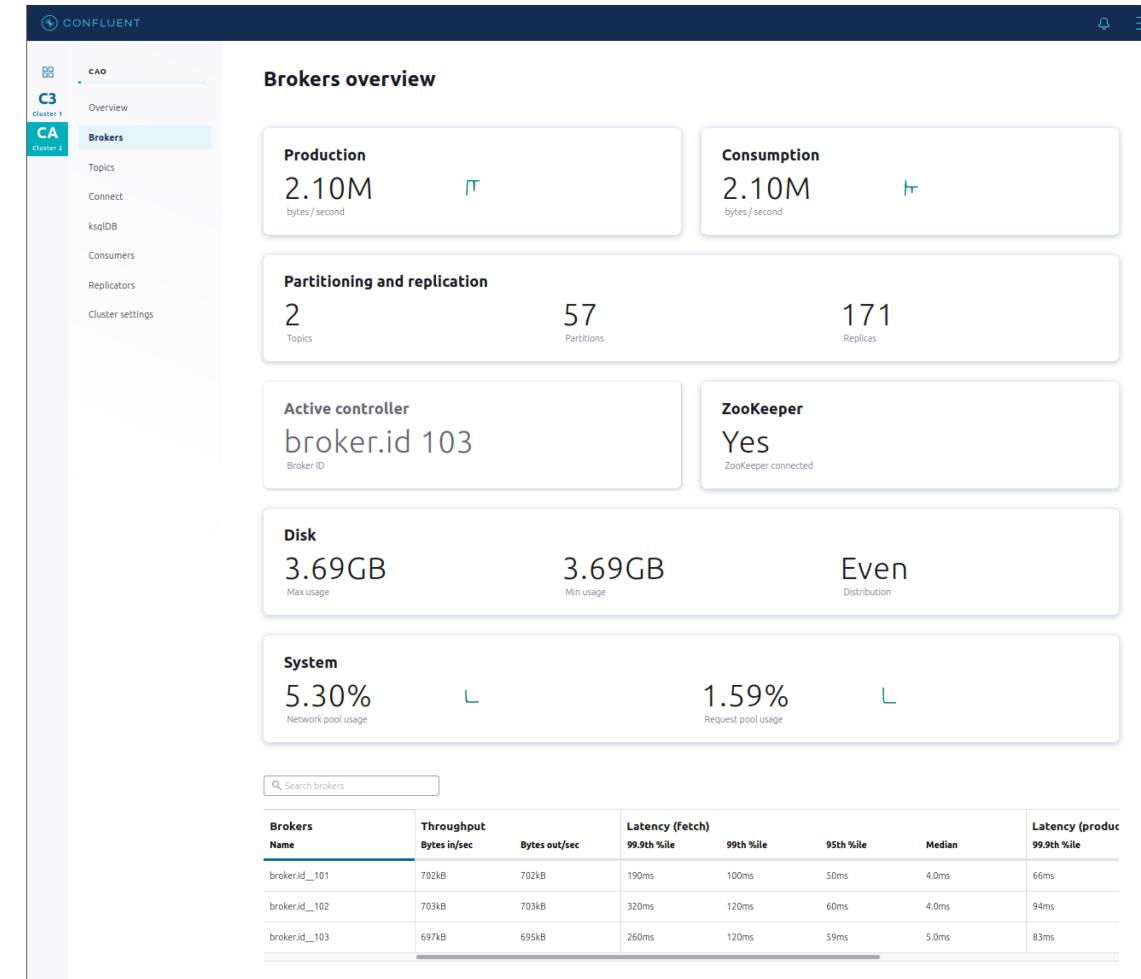
# Control Center



{dev}

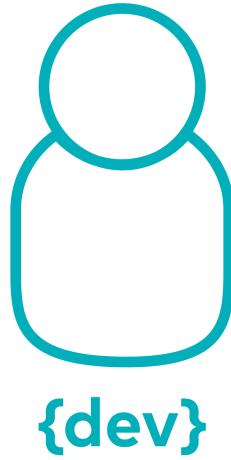
## Look inside Kafka

- Inspect messages in topics
- View changes to Schema Registry



{ops}

# Control Center

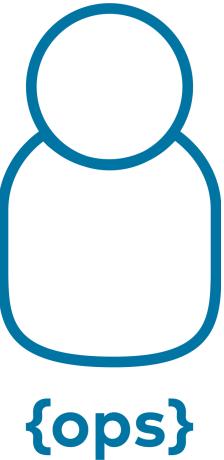
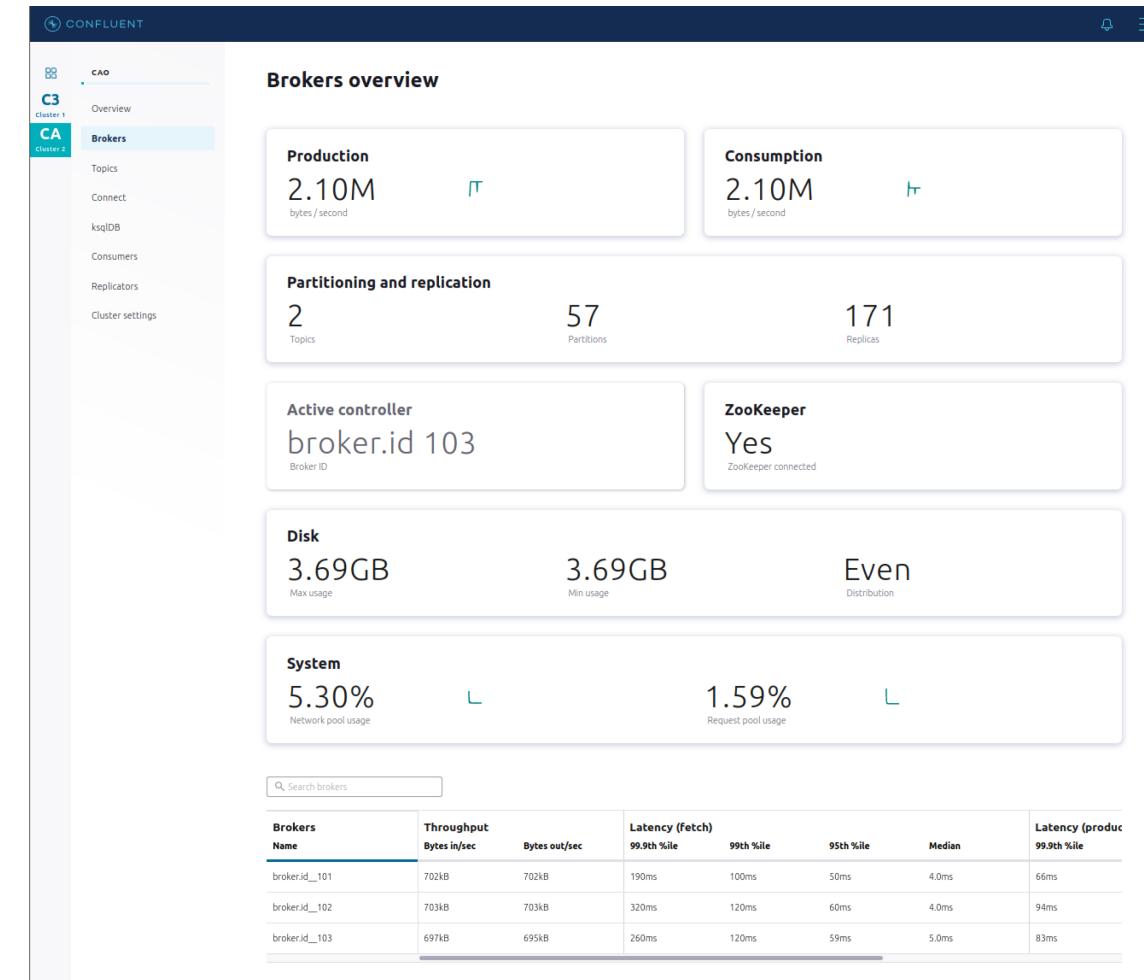


## Look inside Kafka

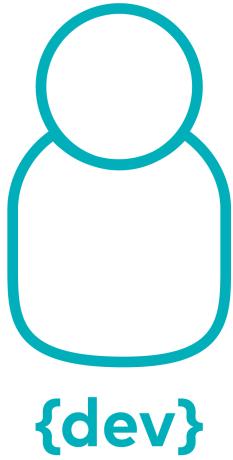
- Inspect messages in topics
- View changes to Schema Registry

## Build pipelines and process streams

- Configure Kafka Connect and connectors
- Write ksqlDB queries



# Control Center

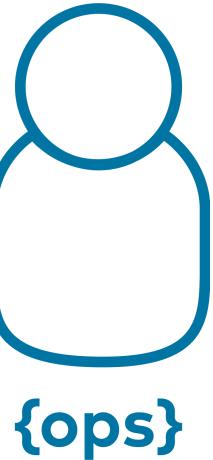
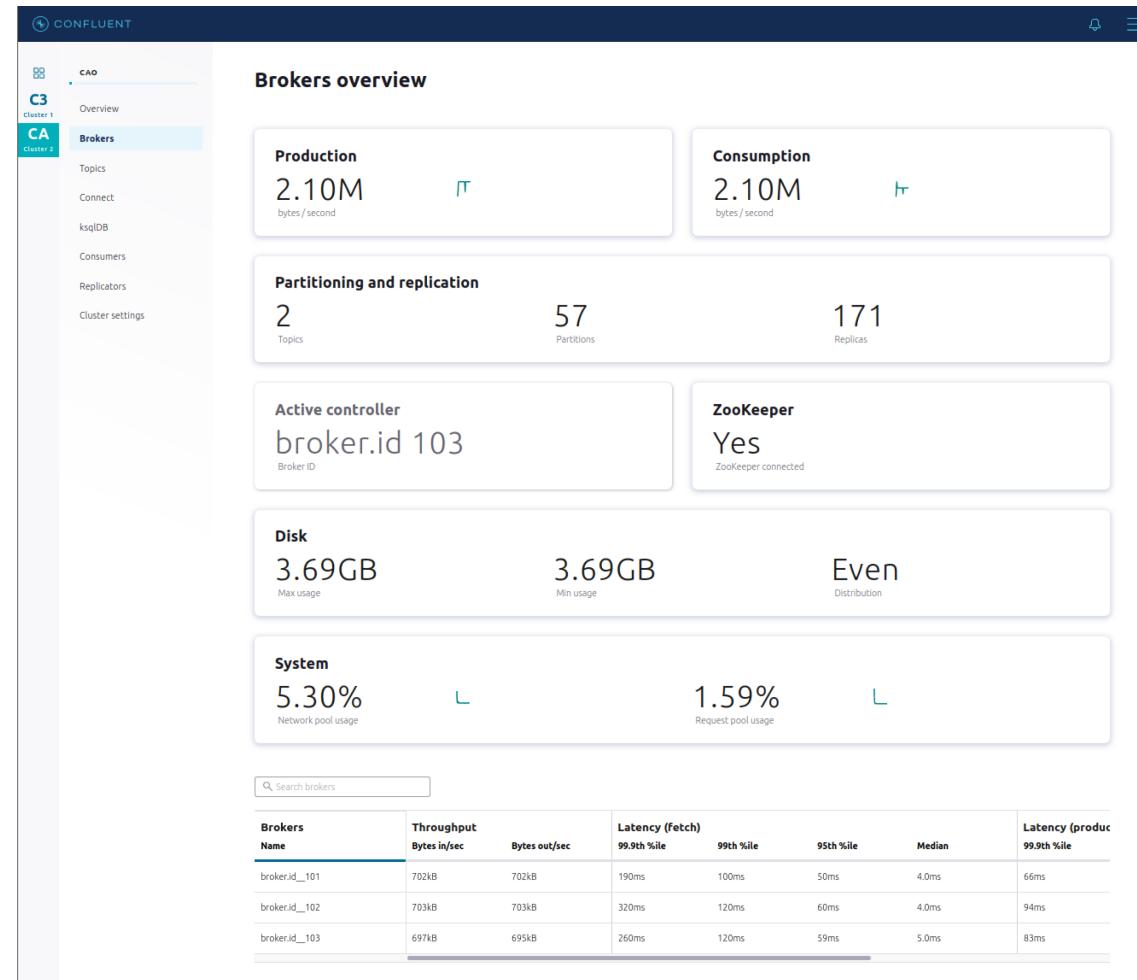


## Look inside Kafka

- Inspect messages in topics
- View changes to Schema Registry

## Build pipelines and process streams

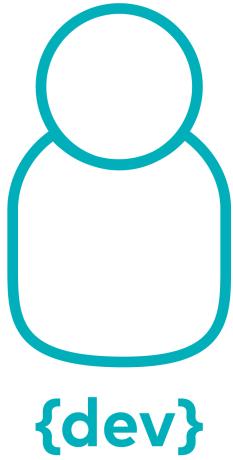
- Configure Kafka Connect and connectors
- Write ksqlDB queries



## Meet event stream SLAs

- Track KPIs for event streams
- View consumer lag
- Set and receive alerts

# Control Center

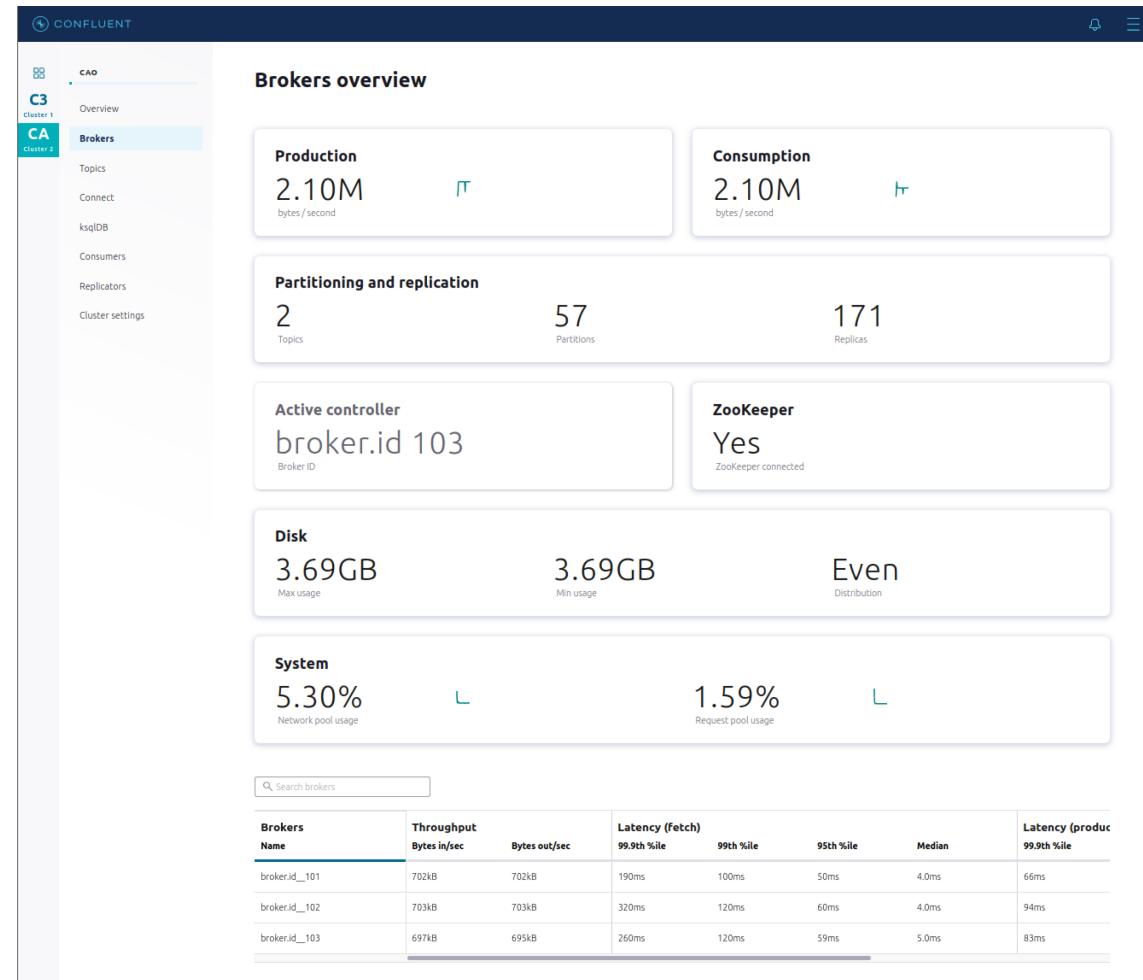


## Look inside Kafka

- Inspect messages in topics
- View changes to Schema Registry

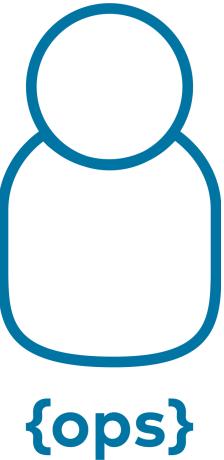
## Build pipelines and process streams

- Configure Kafka Connect and connectors
- Write ksqlDB queries



## Meet event stream SLAs

- Track KPIs for event streams
- View consumer lag
- Set and receive alerts



## View Kafka clusters at a glance

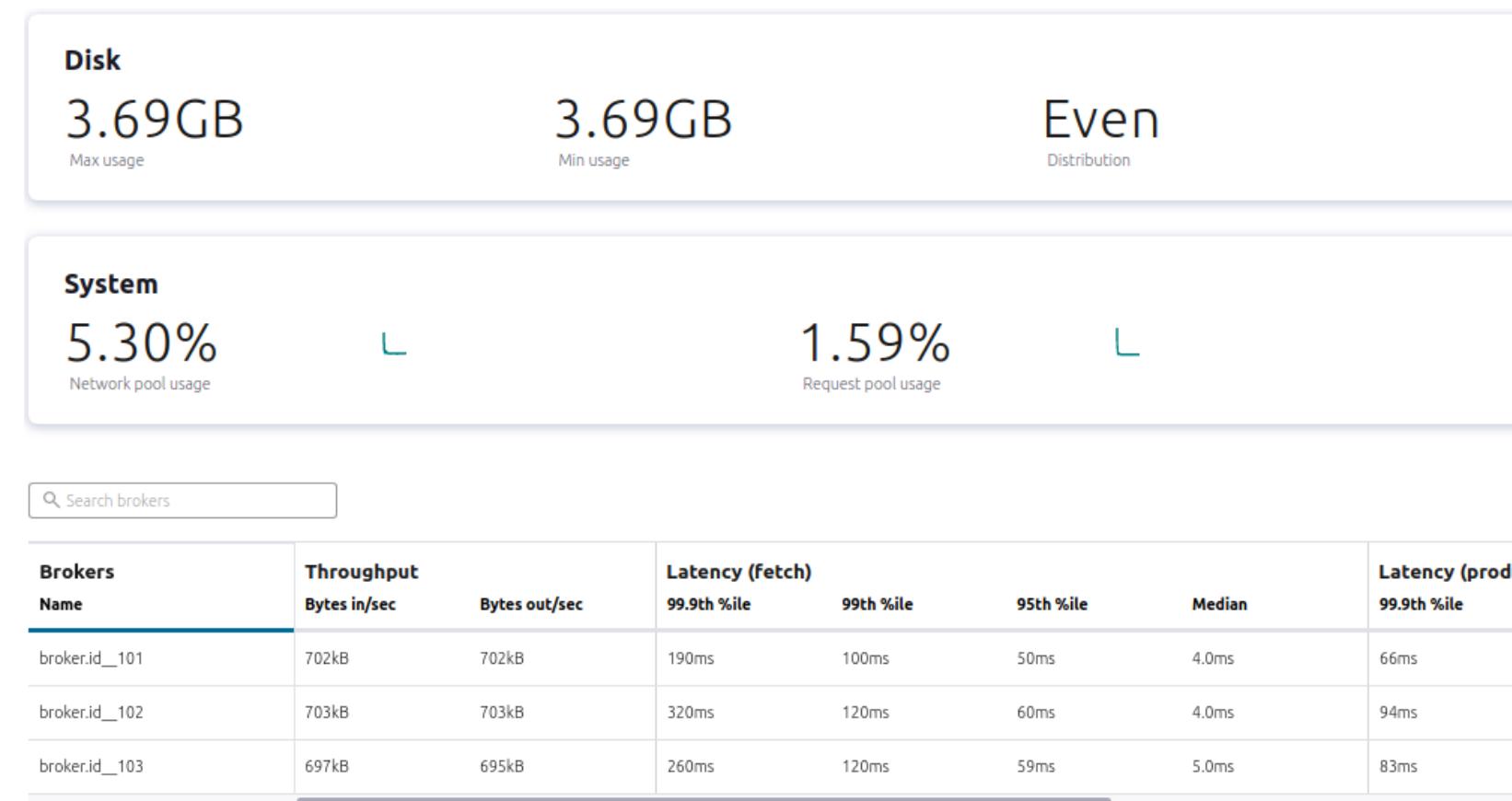
- Configure Kafka Connect and connectors
- Write ksqlDB queries

## The Control Center Advantage

1. Monitor what's important
2. Inherits performance and scalability improvements in Kafka
3. 2-for-1 efficiency in operations
4. Unified security configuration experience
5. Confluent Support for production cluster & monitoring solution

# Monitoring what's important

- Brokers added?
- Topics added?
- Network and request pool usage
- Disk utilization across cluster
- Underreplicated & offline partitions



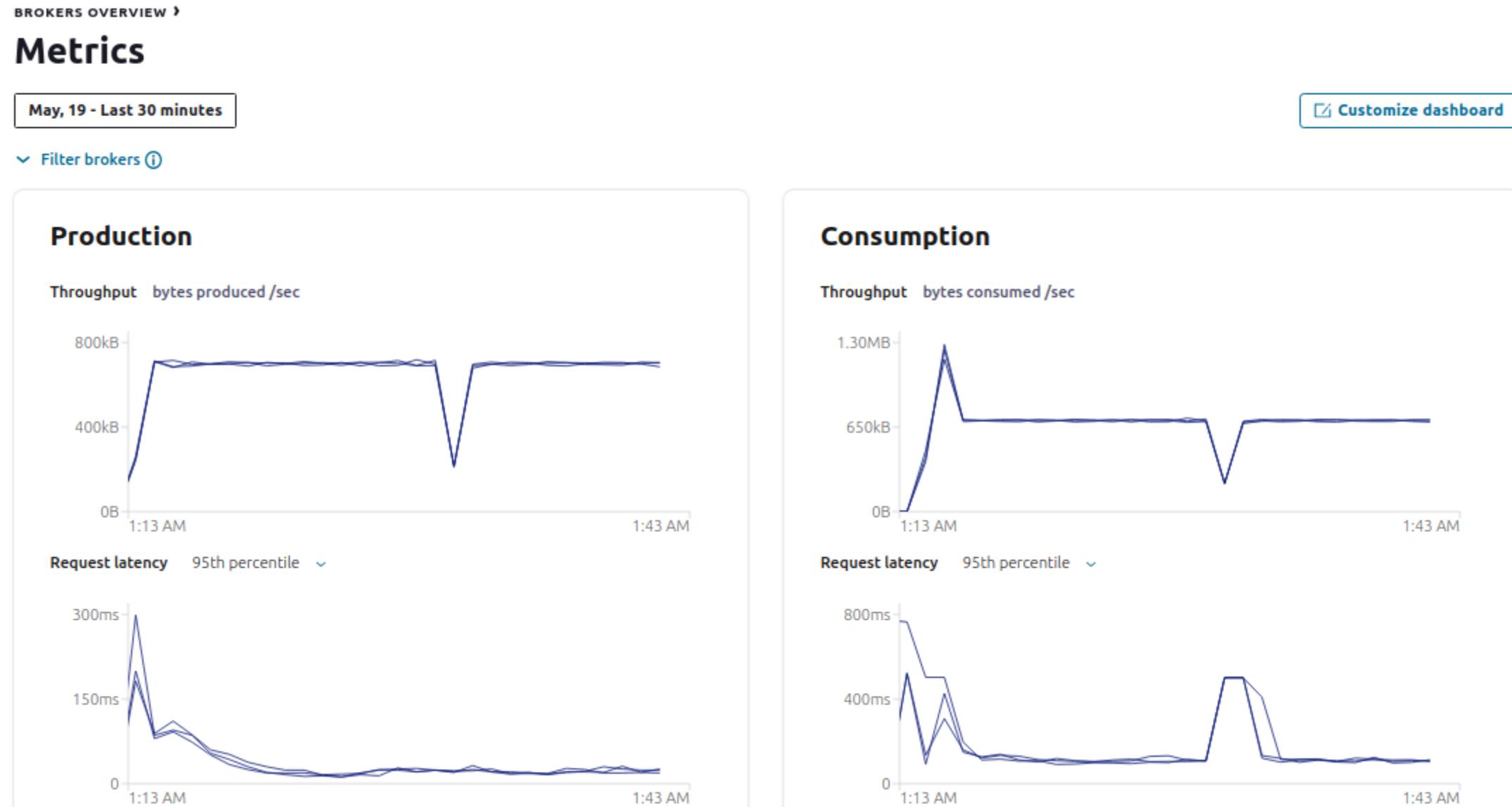
## Production and Consumption

- These numbers can be seen in Control Center aggregated over:
  - the whole cluster
  - individual topic
  - individual consumer group
  - individual consumer instance and partition of a given consumer group

We will now examine these different Control Center views

# The Cluster View

- Performance metrics (throughput and request latency)

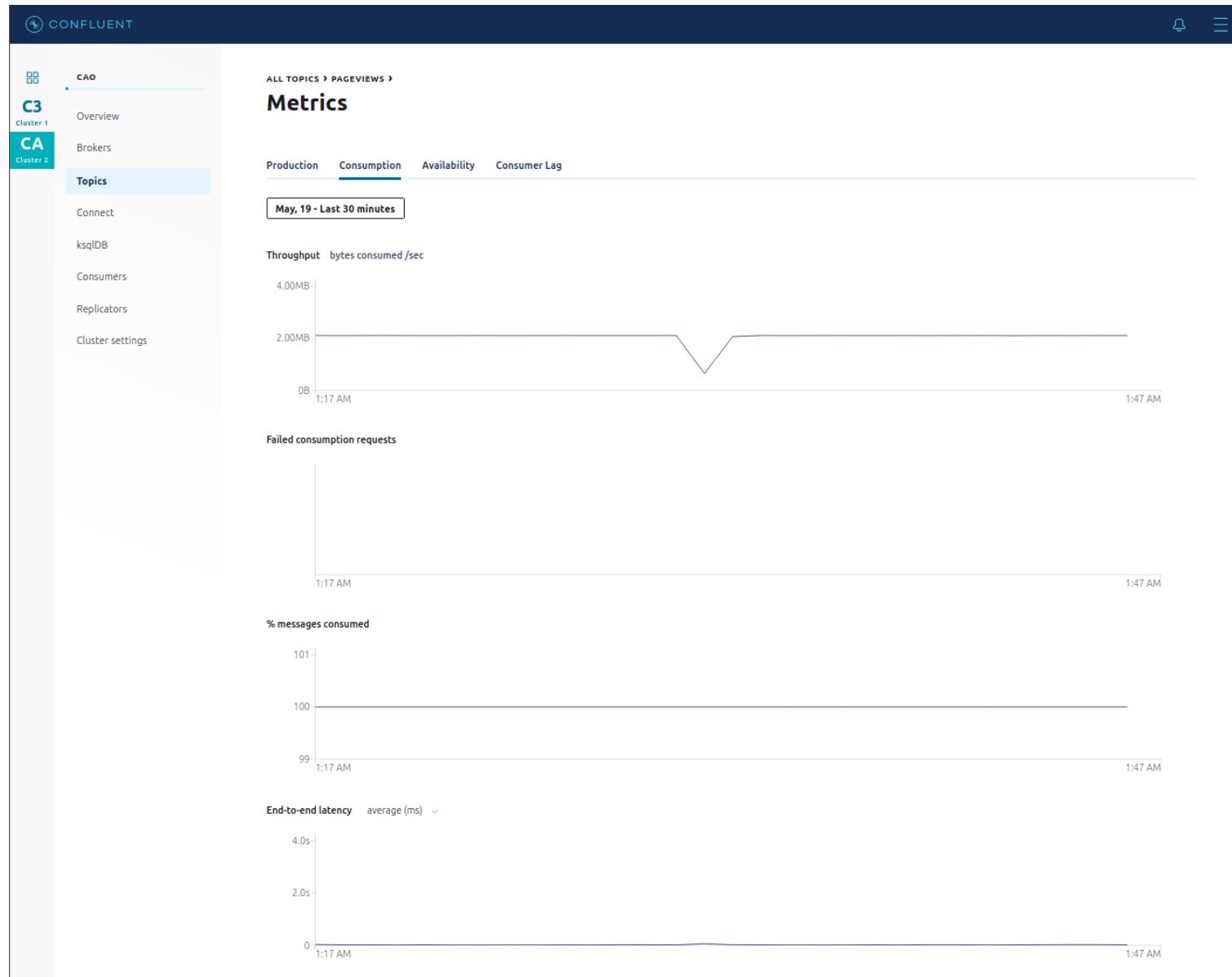


# Topic Overview

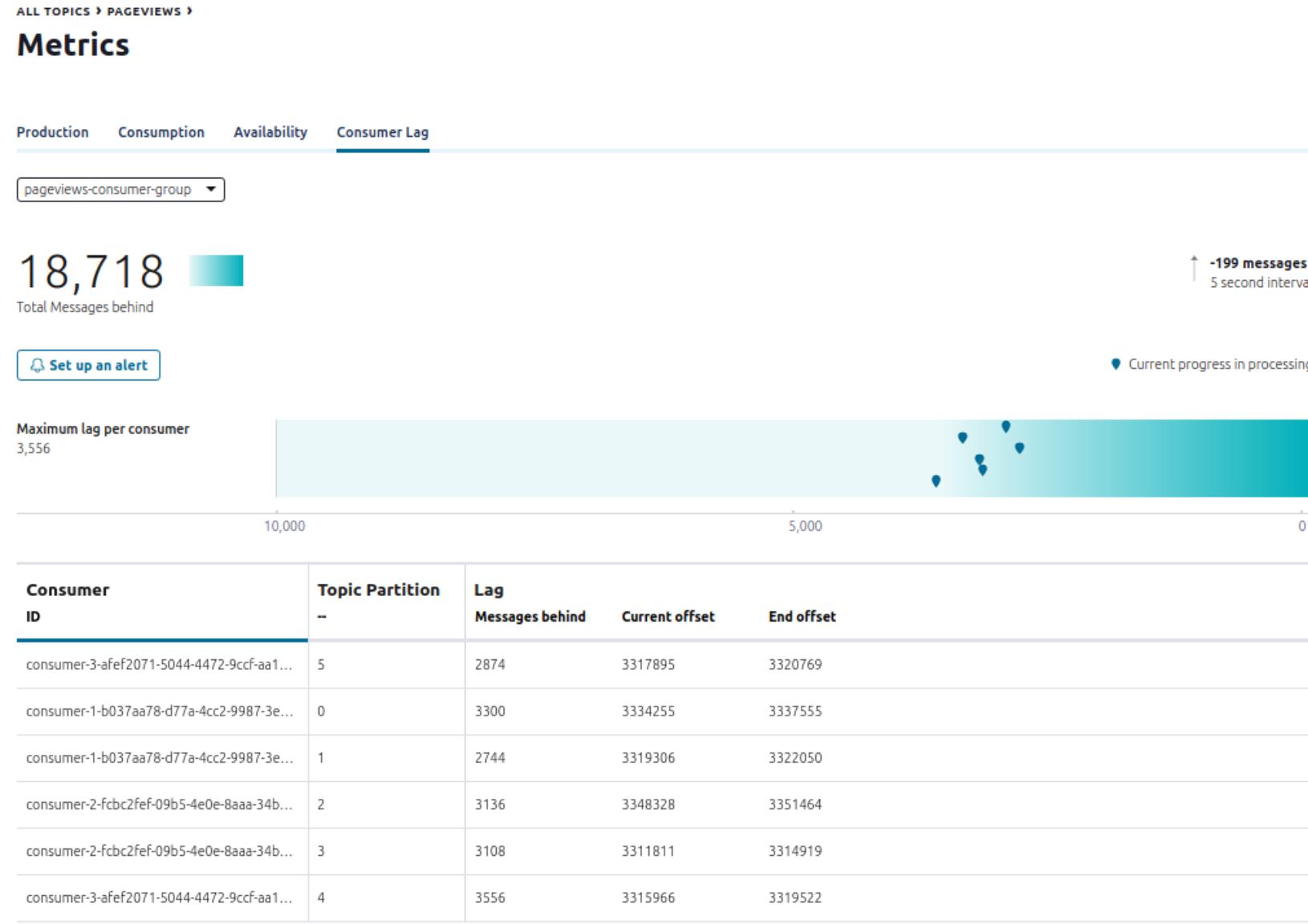
The screenshot shows the Confluent Platform interface for monitoring a Kafka topic named 'pageviews'. The left sidebar lists clusters C3 (Cluster 1) and CA (Cluster 2). The CA cluster is selected, showing its topics: Connect, ksqlDB, Consumers, Replicators, and Cluster settings. The 'Topics' section is currently active. The main content area displays three key metrics: Production at 2099.666K bytes per second, Consumption at 2100.57K bytes per second, and Availability at 0 of 6 under replicated partitions and 0 of 18 out of sync followers. Below these metrics is a search bar labeled 'Search partitions'. A detailed table follows, listing six partitions (0 to 5) with their status, replica placement, offsets, and sizes.

Partitions	Replica placement		Offset	Size
	Partition id	Status		
0	Available	101	103, 102	0 1393809 293MB
1	Available	103	102, 101	0 1378676 290MB
2	Available	102	101, 103	0 1388725 292MB
3	Available	101	102, 103	0 1383275 291MB
4	Available	103	101, 102	0 1389593 292MB
5	Available	102	103, 101	0 1383543 291MB

# Topic - Message Consumption



# Topic - Consumer Lag



## Consumers - All Consumer Groups

- The **All consumer groups** view lists all consumer groups in the cluster and the following metrics for each group:
  - consumer lag
  - number of consumers
  - number of subscribed topics

The screenshot shows the CCC interface with the following details:

- Header:** CONFLUENT logo, CAO navigation bar, C3 Cluster 1, CA Cluster 2, and Consumers tab (highlighted).
- Left Sidebar:** Overview, Brokers, Topics, Connect, ksqlDB.
- Main Title:** All consumer groups
- Search Bar:** Search consumer groups
- Table:** Consumer group statistics

Consumer group	ID	Messages behind	Number of consumers	Number of topics
pageviews-consumer-group	<a href="#">pageviews-consumer-group</a>	41,808	1	1

# Consumers - Consumer Lag

CONFLUENT

CAO

C3 Cluster 1

CA Cluster 2

Overview

Brokers

Topics

Connect

ksqlDB

Consumers

Replicators

Cluster settings

ALL CONSUMER GROUPS >

## pageviews-consumer-group

Consumer lag Consumption

18,652 Total Messages behind

+7981 messages 5 second interval

Set up an alert

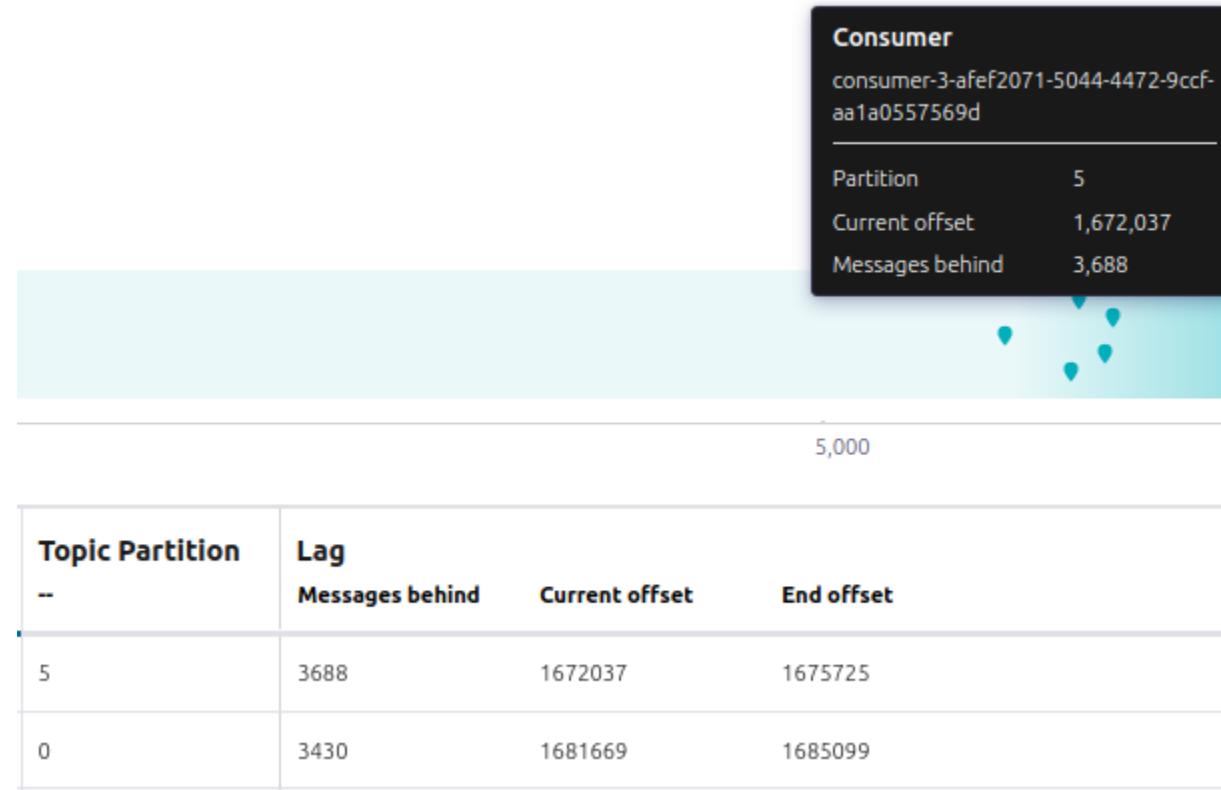
Current progress in processing

pageviews Max lag / consumer: 3,686 messages

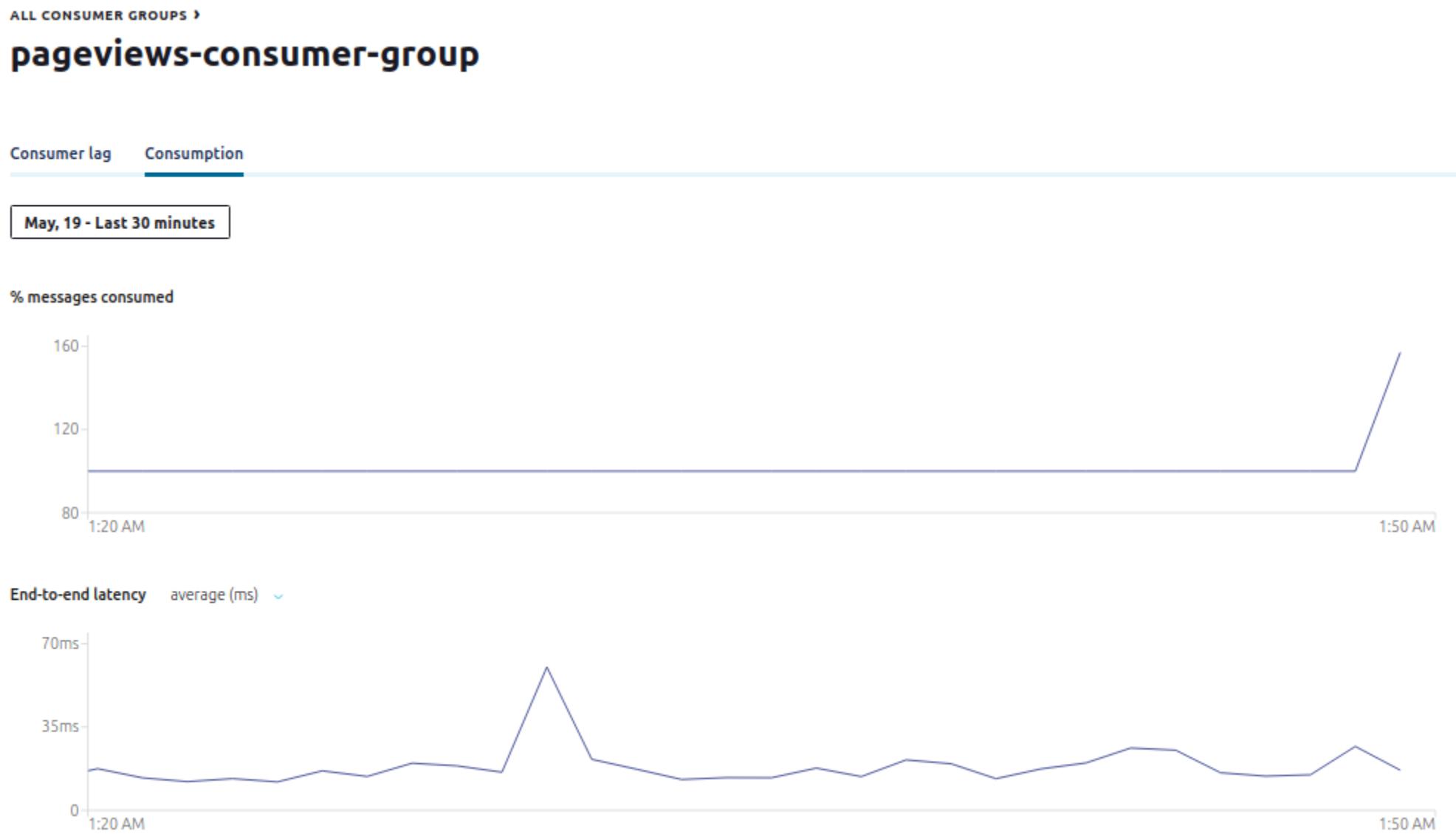
10,000 5,000 0

Consumer ID	Topic Partition	Partition	Lag	Messages behind	Current offset	End offset
Consumer ID	Topic	Partition	Messages behind	Current offset	End offset	
consumer-3-afef2071-5044-4472-9ccf-aa1...	pageviews	5	3556	3332767	3336323	
consumer-1-b037aa78-d77a-4cc2-9987-3e...	pageviews	0	2427	3349413	3351840	
consumer-1-b037aa78-d77a-4cc2-9987-3e...	pageviews	1	2724	3334419	3337143	
consumer-2-fcbc2fef-09b5-4e0e-8aaa-34b...	pageviews	2	2625	3365040	3367665	
consumer-2-fcbc2fef-09b5-4e0e-8aaa-34b...	pageviews	3	3634	3328654	3332288	
consumer-3-afef2071-5044-4472-9ccf-aa1...	pageviews	4	3686	3331840	3335526	

## Consumers - Consumer Lag - Per Partition Details



# Consumers - Consumption



# Consumer Lag Alerts

ALERTS > OVERVIEW > TRIGGERS >

## New trigger

### General

Trigger name\* \_\_\_\_\_

High consumer lag

### Components

Component type\* \_\_\_\_\_

Consumer group

Consumer group name\* \_\_\_\_\_

pageviews-consumer-group [10I0AkqdQvqgUX9VcACQwA]

### Criteria

Metric\* \_\_\_\_\_

Maximum latency (ms)

Buffer (seconds)\* \_\_\_\_\_

60

Condition\* \_\_\_\_\_

Greater than

Value\* \_\_\_\_\_

5000

**Submit**

**Cancel**

# Monitoring Interceptors

## Producers

```
bootstrap.servers=kafka-1:9092,kafka-2:9092  
key.serializer=io.confluent.kafka.serializers.KafkaAvroSerializer  
value.serializer=io.confluent.kafka.serializers.KafkaAvroSerializer  
schema.registry.url=http://schema-registry:8081  
interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor
```

## Consumers

```
...  
interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor
```



### Question:

Control Center puts a special focus on **consumer lag**. Why is it so important to monitor those metrics?

## Further Reading

- Monitoring Your Apache Kafka® Deployment End-to-End:  
<https://www.confluent.io/monitoring-your-apache-kafka-deployment>
- Manage, Monitor and Understand the Apache Kafka Cluster:+  
<https://www.confluent.io/confluent-control-center/>
- 1. Intro | Monitoring Kafka in Confluent Control Center:  
<https://www.youtube.com/watch?v=9myx2FtWQCI>
- Demo: Monitoring Kafka Like a Pro in Confluent Control Center:  
<https://www.youtube.com/watch?v=O9LqDGSoWaU>

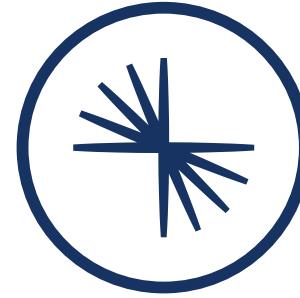
# Lab: Monitoring with CCC - Data Streams

Please work on **Lab 4a: Monitoring with CCC - Data Streams**

Refer to the Exercise Guide



## Branch 2: General Troubleshooting & Tuning - Overview



CONFLUENT  
**Global Education**

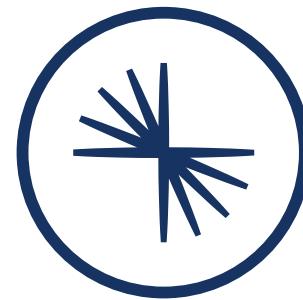
# Agenda



This is a branch of our CAO content on General Troubleshooting & Tuning. It is broken down into the following modules:

4. General Troubleshooting
5. Where are my System Log Files?

## 05: General Troubleshooting



CONFLUENT  
**Global Education**

# Module Overview



This module contains three lessons:

1. Troubleshooting Intro
2. Production Down - Troubleshooting Strategies
3. Troubleshooting Toolkit

## a: Troubleshooting Intro

### Description

Troubleshooting motivation. Overview of data flow. Review of various settings.

## Kafka is a Critical Piece of our Pipeline

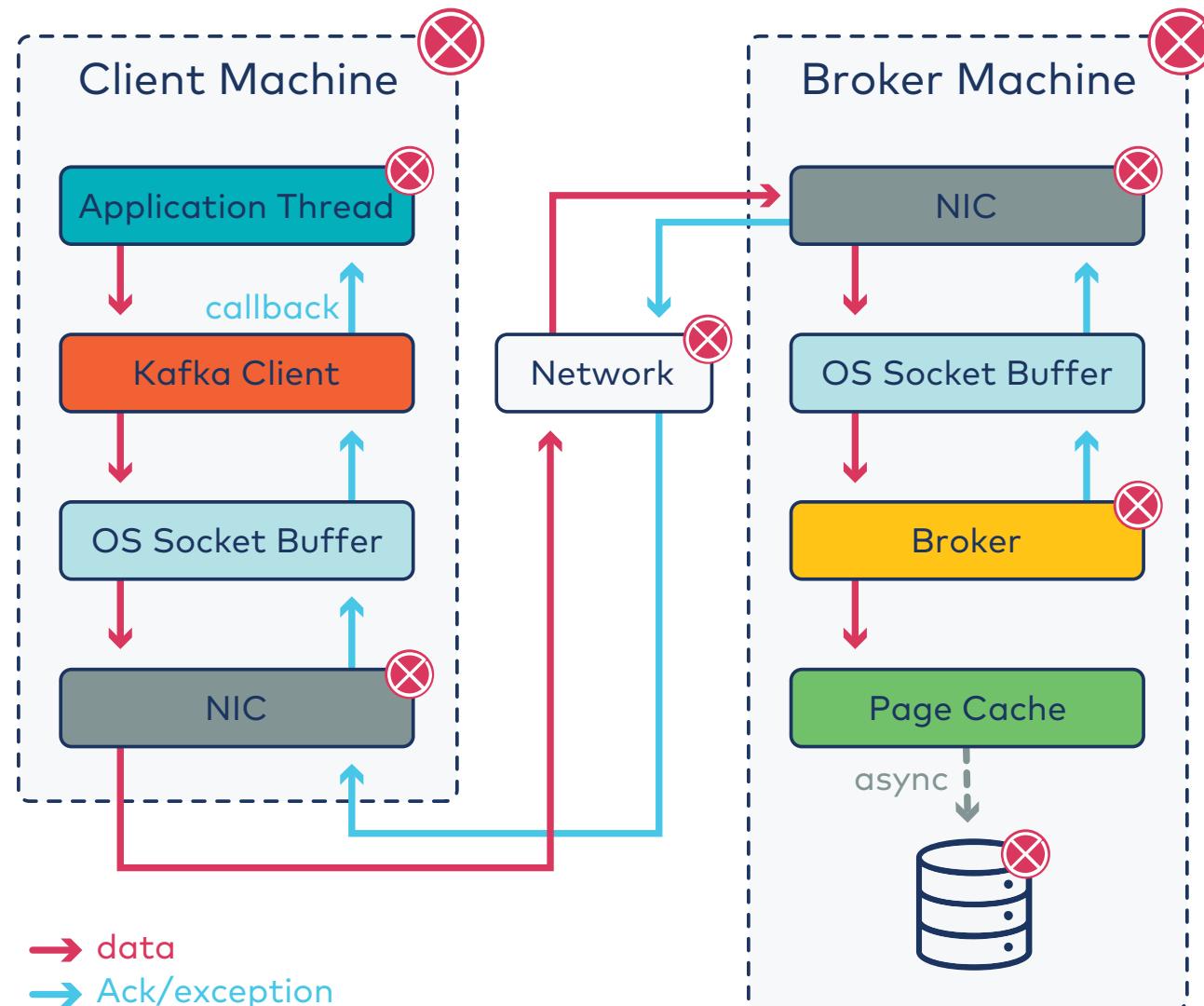
- Can we be 100% sure that our data will get there?
- Can we lose messages?
- How do we verify?
- What is the root cause?



# Distributed Systems

- Things fail
- Systems are designed to tolerate failures
- We must configure our system to handle them

# Data Flow



## Be safe, not sorry

- Producer Settings
  - `max.block.ms` value is appropriate
  - `delivery.timeout.ms` is appropriate
  - `acks=all`
  - `enable.idempotence=true`
  - `producer.close()`
- Broker Settings
  - `unclean.leader.election.enable=false`
- Topic Settings
  - `replication.factor=3` (or more)
  - `min.insync.replicas=2`
- Consumer Settings
  - `auto.offset.commit=false`
- Commit **after** processing
- Monitor!



### Question:

What type of expertise does one need to troubleshoot a streaming platform powered by Kafka?

## Further Reading

- Lessons learned from Kafka in production: <https://www.youtube.com/watch?v=1vLMuWsfMcA>

## b: Production Down! Troubleshooting Strategies

### Description

Overview of dimensions of the problem space for troubleshooting and levels of severity.

## What am I looking for?

Severity:

Symptom	Action
Production down	<b>immediate solution</b>
Production negatively affected	quick solution
Significant friction	open support ticket
Minor issue	report & find work around

Category:

- Infrastructure
- Confluent Platform
- Application

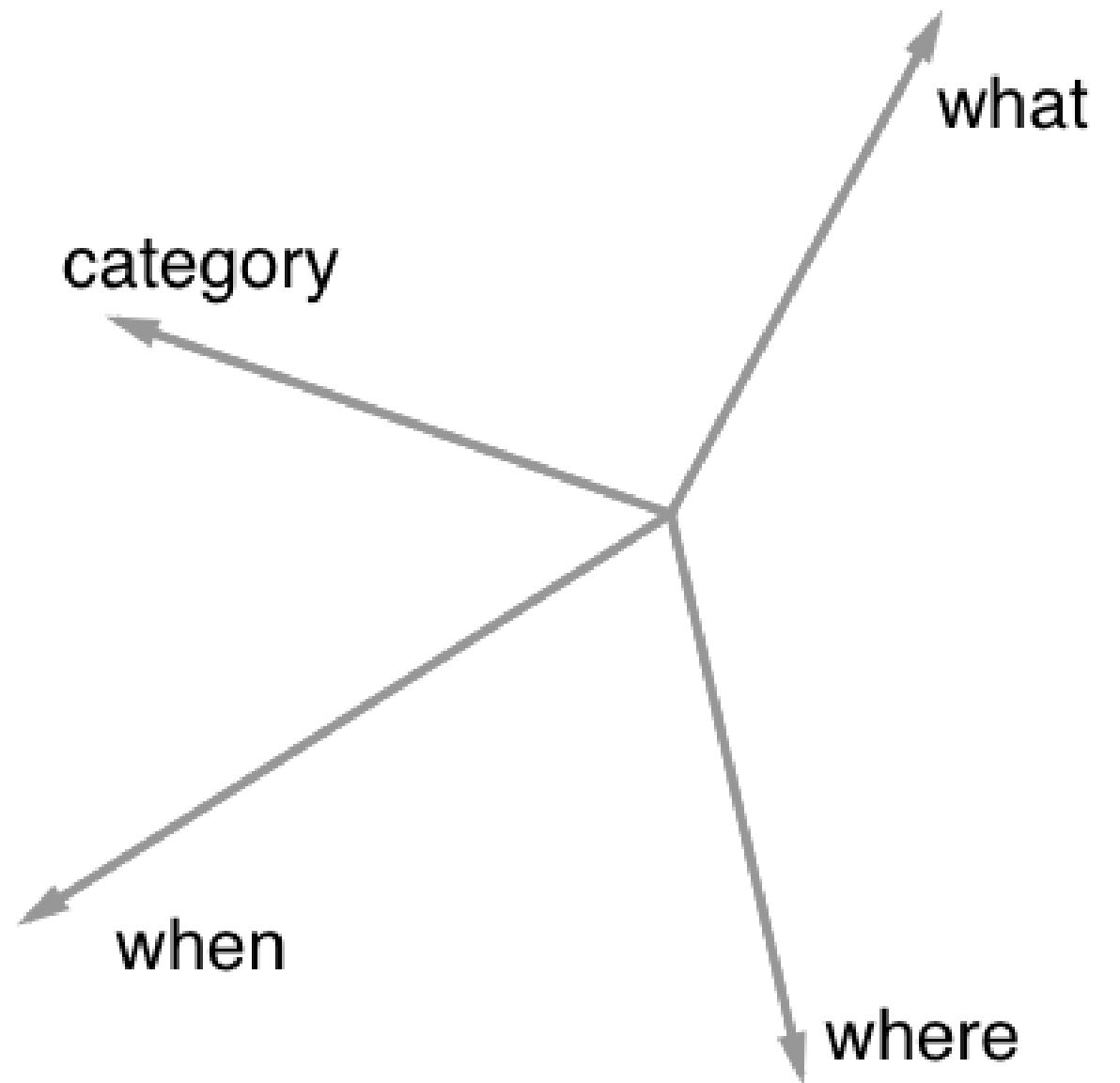
## Asking the right questions

- **What** happened?
- **When** did it happen?
- **Where** did it happen?
- **Category** of the problem
- **Do** we have a support dump?
- **What** did you do?



## Reducing the Problem Space

- Multi-dimensional problem space
- Dimensions: what, when, where, event category, etc.
- Pinpoint as many dimensions as possible





**Question:**

Which is the best way to troubleshoot consumer lag? Justify your answer.

## Further Reading

- Troubleshooting 201: Ask the Right Questions: <http://bit.ly/2zSu5eO>
- Problem Solving & Asking The Right Questions: <http://bit.ly/2zTv3Hi>

## c: Troubleshooting Toolkit

### Description

Tour of troubleshooting tools.

# Linux Performance Observability Tools

## Memory:

- free
- top, htop
- vmstat
- ...

## I/O:

- iostat
- iotop
- blktrace
- ...

## Network Stack:

- netstat
- netcat
- tcpdump
- iptraf
- ethtool
- ...

## Netshoot Container

- Contains many recommended tools
- Troubleshoot application container network:

```
$ docker run -it --net container:<container_name> nicolaka/netshoot
```

- Troubleshoot host network:

```
$ docker run -it --net host nicolaka/netshoot
```

# Kafka Command Line Tools

- `zookeeper-shell`
- `kafka-configs`
- `kafka-topics`
- `kafka-consumer-groups`
- `kafka-acls`
- `kafka-console-consumer`,  
`kafka-console-producer`
- `kafka-avro-console-consumer`,  
`kafka-avro-console-producer`
- `kafka-producer-perf-test`,  
`kafka-consumer-perf-test`

## The **kafkacat** Tool

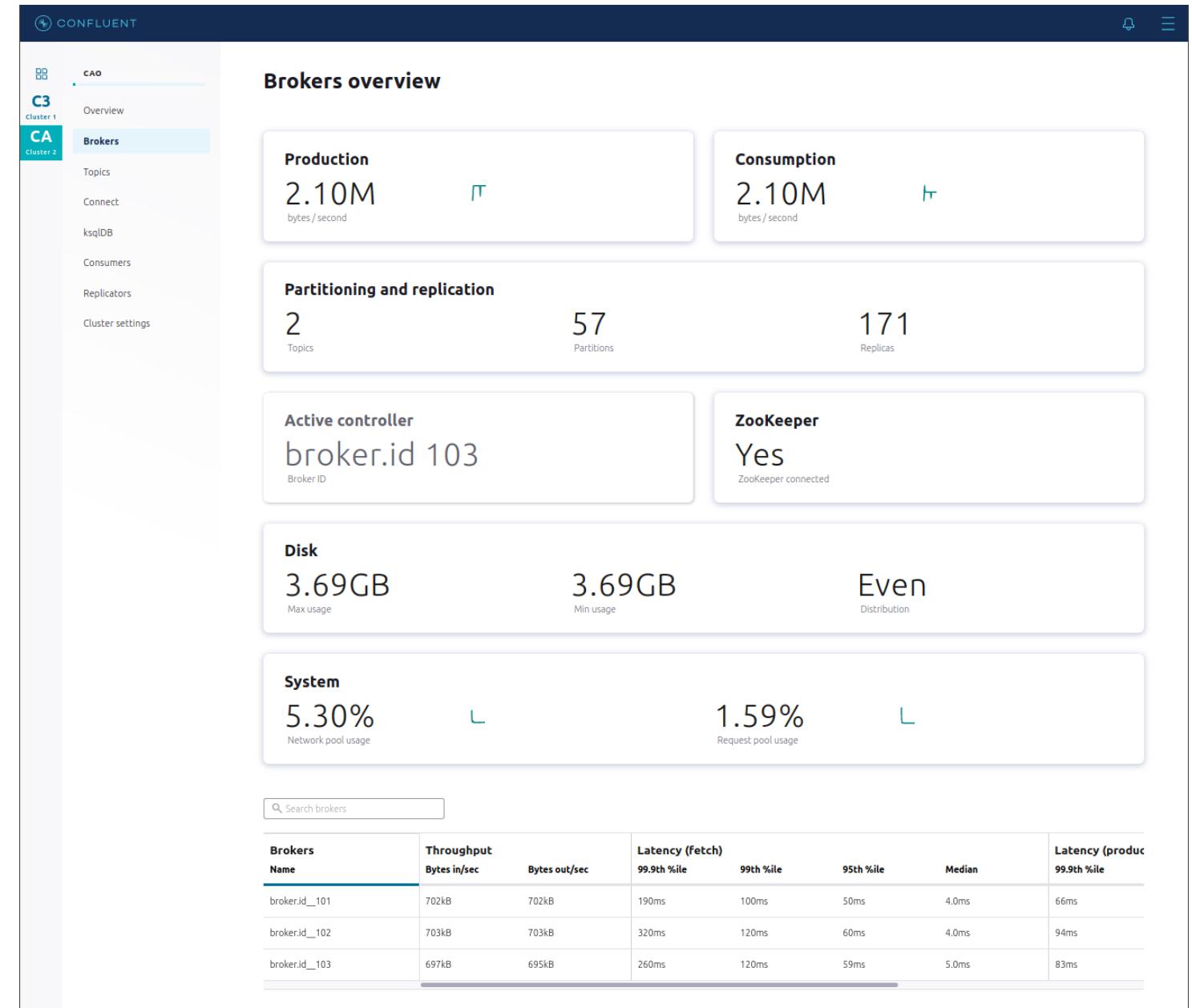
- Produce, consume and list topic and partition information
- **netcat** for Kafka

```
$ kcat \
  -b localhost:9092 \
  -t keyed_topic -C \
  -f 'Key: %k\nValue: %s\n'

Key: 1
Value: foo
Key: 2
Value: bar
```

# Confluent Control Center

- Broker health and configurations
- Topics and partitions
- Data flow & end-to-end latency
- Consumer groups
- Replication





**Question:**

Which is the best way to troubleshoot consumer lag? Justify your answer.

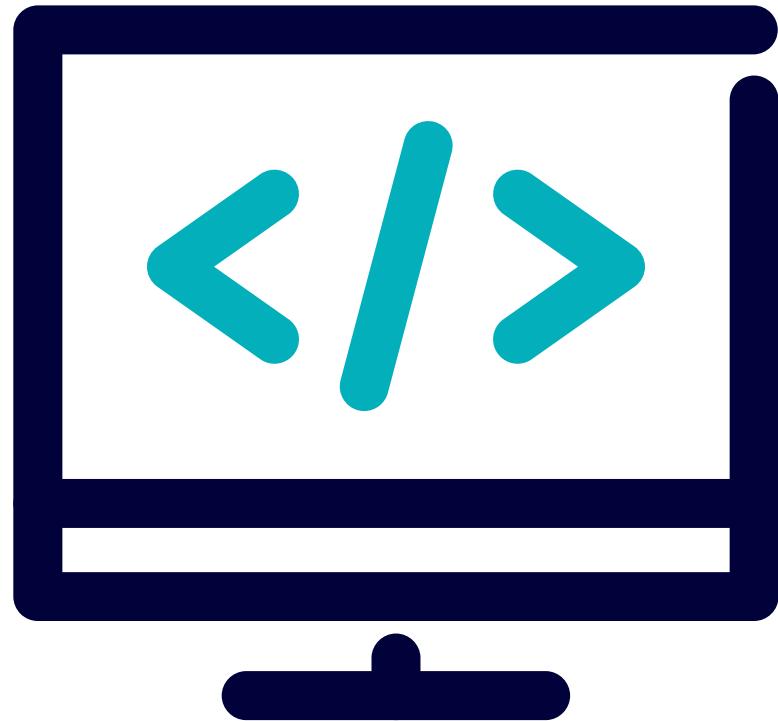
## Further Reading

- Linux Performance: <http://www.brendangregg.com/linuxperf.html>
- `netshoot` Container: <http://bit.ly/2z2xGn4>
- `kafkacat` Utility: <https://docs.confluent.io/current/app-development/kafkacat-usage.html>
- Performance Testing: <https://cwiki.apache.org/confluence/display/KAFKA/Performance+testing>

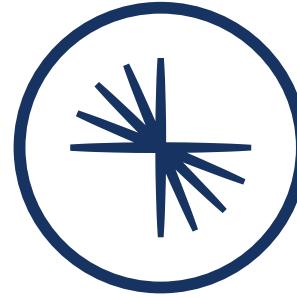
# Lab: Troubleshooting Toolbelt

Please work on **Lab 5a: Troubleshooting Toolbelt**

Refer to the Exercise Guide



# 06: Where are my System Log Files?



CONFLUENT  
Global Education

# Module Overview



This module contains one lesson:

1. Where are my System Log Files?

# a: Where are my System Log Files?

## Description

Overview of Kafka system logs and tips for analyzing them.

# Finding the relevant information

## Questions to ask:

- What is the event?
- What is the date/time of the unexpected event?
- Which component of the platform is affected?
- Which instances/nodes of the cluster are affected?



Log files can be very large

## Analyzing large log files

- Use `less` tool:

```
$ less /var/log/kafka/kafkaServer-0.current
```

or

```
$ docker-compose logs kafka | less
```

- Use `grep` tool:

```
$ docker-compose logs kafka | grep -i -E "(stopped|start(ing|ed))"  
$ docker-compose logs kafka | grep -i -E "started" | wc -l  
$ grep '<my pattern>' * -R
```

# Naming conventions

Component	Component Name	Subfolder
ZooKeeper	KAFKA	kafka
Kafka broker	KAFKA	kafka
Confluent Control Center	CONTROL_CENTER	confluent-control-center
Schema Registry	SCHEMA_REGISTRY	schema-registry
REST Proxy	KAFKA_REST	kafka-rest
Kafka Connect	CONNECT	kafka
ksqlDB Server	KSQl	ksql

## Logging Framework of Confluent Platform

- Kafka brokers and clients use the `slf4j` logging abstraction
- **Log4J** is the default logging framework used in all components
- **Log4J** is configured via properties files
- Default location of Log4J config file:

`/etc/<component-name>/log4j.properties`

- `log4j.properties` defines log target
  - STDOUT, STDERR
  - Files
- Also defines log level:

`TRACE, DEBUG, ..., ERROR, FATAL`

# Log4j Configuration in Production Systems

1. For key `log4j.rootLogger`, replace `stdout` with `kafkaAppender`
2. For key `log4j.appenders.X` use `RollingFileAppender`
3. Remove key `log4j.appenders.X.DatePattern`
4. Set `MaxFileSize=100MB`, `MaxBackupIndex=10`

```
log4j.rootLogger=INFO, kafkaAppender
...
log4j.appenders.stateChangeAppender=org.apache.log4j.RollingFileAppender
log4j.appenders.stateChangeAppender.File=${kafka.logs.dir}/state-change.log
log4j.appenders.stateChangeAppender.layout=org.apache.log4j.PatternLayout
log4j.appenders.stateChangeAppender.layout.ConversionPattern=[%d] %p %m (%c)%n
log4j.appenders.stateChangeAppender.Append=true
log4j.appenders.stateChangeAppender.MaxBackupIndex=10
log4j.appenders.stateChangeAppender.MaxFileSize=100MB
...
```

## Running in Containers

- Always log to STDOUT and STDERR
- Default location of Log4J config file:

```
/etc/<component>/log4j.properties
```

- Override with own config:

```
-Dlog4j.configuration=file:/path/to/log4j.properties
```

## What to collect?

- Log files of all Confluent Platform components
- System information:

```
echo $HOSTNAME  
cat /etc_host/*release  
cat /proc/version  
cat /proc/cpuinfo  
cat /proc/meminfo  
cat /proc/cgroups  
cat /proc/self/cgroup  
df -h  
mount  
vmstat 1 5  
iostat 1 5  
dmidecode
```

- When running in containers

```
$ docker version  
$ docker system info  
$ docker container stats --all --no-stream  
$ sudo cat /etc/docker/daemon.json  
$ journalctl -t dockerd --no-pager  
  
# from: cnfl.io/check-config  
$ ./check-config.sh
```

- Certificates

```
$ openssl x509 -in ca.pem -text  
$ openssl ec -in key.pem -text
```

## Brokers - Log4J Logs

Log Name	Description
server.log	Basic broker operations (segment rolls, replica fetching, etc.)
state-change.log	Updates received from and sent to controller
kafka-request.log	All information about all requests. Verbose, not great for live debugging. Use when no other options for root cause. Rogue clients, single request slowdowns, etc.
log-cleaner.log	Cleaner thread (related to compaction)
controller.log	Written to by active controller
kafka-authorizer.log	Secure connections only (set to DEBUG if you want successful connections)

## Broker - Logging Options

- Define alternative logging config file
  - running natively:

```
$ export KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:/path/to/tools-log4j.properties"
```

- running in container:

```
docker run -d \  
  --name=kafka \  
  --net=sample-net \  
  -e KAFKA_BROKER_ID=101 \  
  -e KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181 \  
  -e KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:9092 \  
  -e KAFKA_LOG4J_OPTS="-Dlog4j.configuration=file:/path/to/tools-log4j.properties" \  
  -e KAFKA_LOG4J_LOGGERS="kafka.controller=WARN,kafka.request.logger=DEBUG" \  
  -e KAFKA_LOG4J_ROOT_LOGLEVEL=WARN \  
  -e KAFKA_TOOLS_LOG4J_LOGLEVEL=ERROR \  
  confluentinc/cp-kafka:6.0.0-1-ubi8
```

## Set Log Levels Dynamically

- Need more data to troubleshoot!
- **Cannot restart** brokers in production
- Solution: **dynamically** set log level using `kafka-configs`
- Example: set a logger level to DEBUG for broker 101

```
$ kafka-configs \  
  --bootstrap-server kafka:9092 \  
  --alter \  
  --add-config "kafka.server.ReplicaManager=WARN,kafka.server.KafkaApis=DEBUG" \  
  --entity-type broker-logger \  
  --entity-name 101
```



Be sure to reset log level after successful debugging!

## Kafka Clients - Producer & Consumer

- Alternative log config file:

```
-Dlog4j.configuration=file:/path/to/log4j.properties
```

- Typical config file when running in container:

```
log4j.rootLogger=WARN, stderr
log4j.appender.stderr=org.apache.log4j.ConsoleAppender
log4j.appender.stderr.layout=org.apache.log4j.PatternLayout
log4j.appender.stderr.layout.ConversionPattern=[%d] %p %m (%c)%n
log4j.appender.stderr.Target=System.err
```

# ksqldb Server

Installation environment	How to access the ksqldb logs
Docker container	<pre>\$ docker logs &lt;container ID&gt;</pre> <p>— or —</p> <pre>\$ docker-compose logs ksqlldb-server</pre>
Confluent CLI	<pre>\$ bin/confluent logs ksqlldb-server</pre>
RPM/DEB	Log files in folder <code>/var/log/confluent</code>

- Examine logs and look for deserialization errors:

```
$ docker-compose logs ksqlldb-server | grep -i -E "deserialization error"
```

## Confluent Control Center

- Sample command to start Control Center:

```
$ docker run -d \
  --name=control-center --net=sample-net \
  --ulimit nofile=16384:16384 -p 9021:9021 \
  -e CONTROL_CENTER_BOOTSTRAP_SERVERS=kafka:9092 \
  -e CONTROL_CENTER_REPLICATION_FACTOR=1 \
  -e CONTROL_CENTER_MONITORING_INTERCEPTOR_TOPIC_PARTITIONS=1 \
  -e CONTROL_CENTER_INTERNAL_TOPICS_PARTITIONS=1 \
  -e CONTROL_CENTER_STREAMS_NUM_STREAM_THREADS=2 \
  -e CONTROL_CENTER_CONNECT_CLUSTER=http://localhost:8083 \
  -e CONTROL_CENTER_LOG4J_OPTS="-Dlog4j.configuration=file:/path/to/log4j.properties" \
  -e CONTROL_CENTER_LOG4J_ROOT_LOGLEVEL=DEBUG \
confluentinc/cp-control-center:6.0.0-1-ubi8
```

- Control Center is up and running?

```
$ docker-compose logs control-center | grep -i -E started
```

## librdkafka Clients (1)

- librdkafka supports logging
- Use property debug to enable contexts
  - Producer: debug=broker,topic,msg
  - Consumer: debug=consumer,cgrp,topic,fetch

## librdkafka Clients (2)

### Python

```
logger = logging.getLogger('producer')
logger.setLevel(logging.DEBUG)

p = confluent_kafka.Producer({'debug': 'all'},
                             logger=logger)
```

## librdkafka Clients (3)

### DOT.NET

```
var producerConfig = new ProducerConfig
{
    BootstrapServers = bootstrapServers,
    Debug = "all"
};
using (var producer =
    new ProducerBuilder<byte[], byte[]>(producerConfig)
        .SetLogHandler((_, m) => logCount += 1)
        .Build())
{ ... }
```



### Question:

Why is it so important to collect all logging information possible from a production system?

## Further Reading

- Apache Log4j 1.x: <https://logging.apache.org/log4j/2.x/manual/compatibility.html>
- How to configure logging for Kafka producers?  
<https://stackoverflow.com/questions/35773780/how-to-configure-logging-for-kafka-producers>
- Troubleshooting KSQL – Part 1: Why Isn't My KSQL Query Returning Data?:  
<https://www.confluent.io/blog/troubleshooting-ksql-part-1>

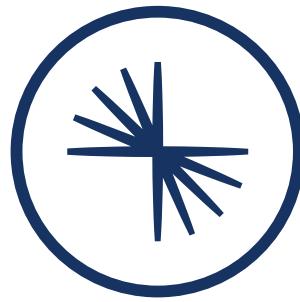
# Lab: Where are my Log Files?

Please work on **Lab 6a: Where are my Log Files?**

Refer to the Exercise Guide



# Branch 3: Troubleshooting & Tuning Central Services - Overview



CONFLUENT  
**Global Education**

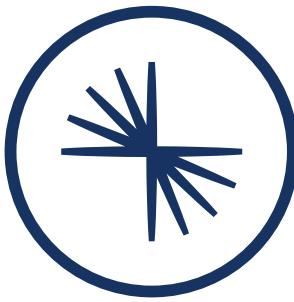
# Agenda



This is a branch of our CAO content on Troubleshooting & Tuning Central Services. It is broken down into the following modules:

7. Troubleshooting & Tuning ZooKeeper
8. Troubleshooting & Tuning Brokers
9. Troubleshooting & Tuning Schema Registry

# 07: Troubleshooting & Tuning ZooKeeper



CONFLUENT  
Global Education

# Module Overview



This module contains one lesson:

1. Troubleshooting Zookeeper

## a: Troubleshooting ZooKeeper

### Description

Overview of ZooKeeper's responsibilities and tips for managing ZooKeeper.

## ZooKeeper's Responsibilities

- Topic configuration
- Controller election
- Access Control Lists (ACLs)
- Client Quotas
- Cluster membership
- Info about leaders
- Info about ISRs
- Dynamic broker configurations

## Tuning ZooKeeper

- Dedicated ZooKeeper cluster
- Keep ZK instances close
- Dedicated disks for TX logs
- Dedicated server for each ZK instance
- Turn **swappiness** to `1` or disable it
- Servers in different AZs
- Enough RAM
- At least 512MB Heap (1GB recommended)
- 3 to 5 instances in Quorum
- Use static IP addresses



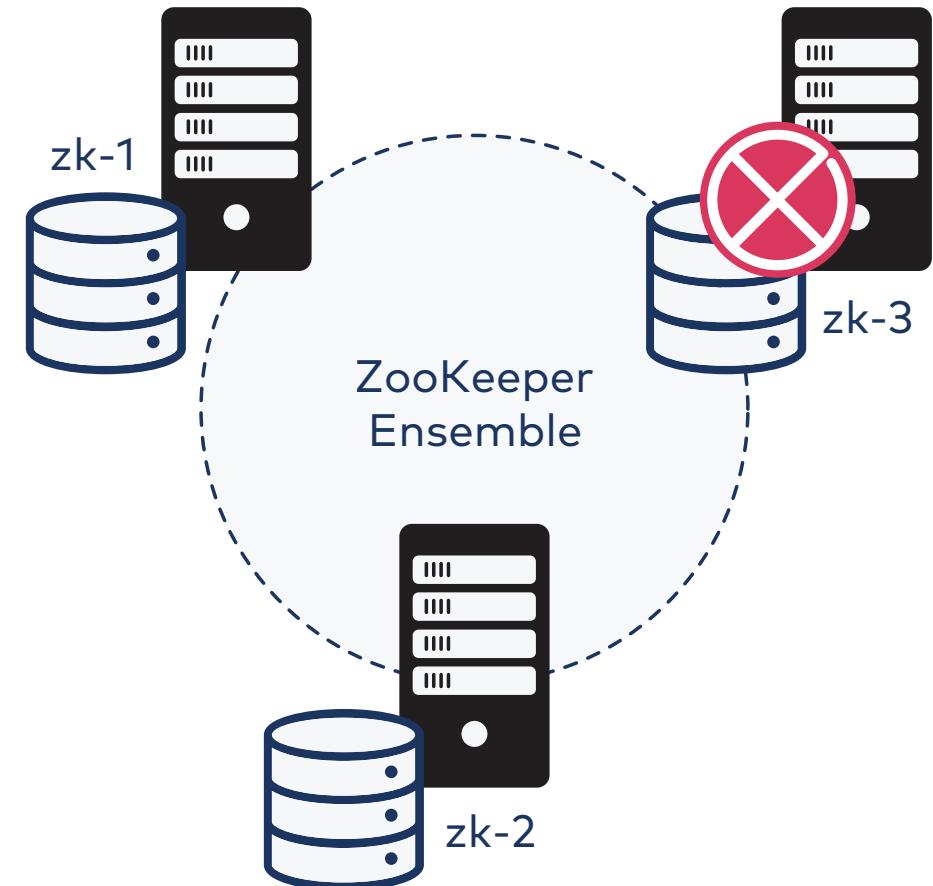
ZK ensemble down == Kafka limited!

## Basic Check

1. Find the state of each ZK instance
2. Verify that there is **only one** leader
3. Verify that a quorum exists
4. Verify all followers are in-sync

## Total Server Loss

1. Backup ZooKeeper data from the **leader**
2. Prepare a **new server** with the ZK binaries
3. Stop ZK process gracefully
4. On new server **start ZK process**
5. Optional: rolling restart of all ZK instances
6. Wait until all followers are **in-sync with leader**



# Read ZooKeeper Data

Why:

- Debug issues with ZK connectivity
- Locate unexpected ZK client

How:

## 1. Read Transaction Log:

```
java -cp zookeeper-3.4.13.jar:lib/log4j-1.2.16.jar:lib/slf4j-log4j12-1.6.1.jar:lib/slf4j-api-1.6.1.jar  
org.apache.zookeeper.server.LogFormatter version-2/log.xxx
```

## 2. Read Snapshot Logs:

```
java -cp zookeeper-3.4.13.jar:lib/log4j-1.2.16.jar:lib/slf4j-log4j12-1.6.1.jar:lib/slf4j-api-1.6.1.jar  
org.apache.zookeeper.server.SnapshotFormatter version-2/snapshot.xxx
```



### Question:

Why should all ZooKeeper instances be located geographically close to each other?

## Further Reading

- ZK Quorums: [https://zookeeper.apache.org/doc/current/zookeeperInternals.html#sc\\_quorum](https://zookeeper.apache.org/doc/current/zookeeperInternals.html#sc_quorum)
- ZK Leader Activation: [https://zookeeper.apache.org/doc/current/zookeeperInternals.html#sc\\_leaderElection](https://zookeeper.apache.org/doc/current/zookeeperInternals.html#sc_leaderElection)

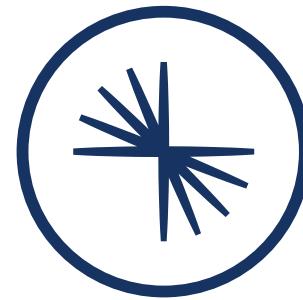
# Lab: Troubleshooting ZooKeeper

Please work on **Lab 7a: Troubleshooting ZooKeeper**

Refer to the Exercise Guide



# 08: Troubleshooting & Tuning Brokers



CONFLUENT  
**Global Education**

# Module Overview



This module contains seven lessons:

1. Troubleshooting Brokers
2. Replication Concerns
3. Tuning Brokers: General Concepts & Best Practices
4. Optimization of a Message's Life Cycle on a Broker
5. Misc. Matters
6. Confluent Auto Data Balancer
7. Message Delivery Guarantees

## a: Troubleshooting Brokers

### Description

Application of general troubleshooting to brokers. Common broker troubleshooting scenarios. Metrics. General recommendations for brokers.

## Troubleshooting Checklist

- What exactly is happening?
- Make sure symptoms are well described
- Establish a time line
- Have you changed anything?
- What monitoring do you have in place?
- Can you SSH into broker host?
- Locate logs
- Parse logs for errors



## Monitoring

- Monitoring is prerequisite to efficient troubleshooting
- Deviation from trends indicate problems
- Monitoring and alerts help to avoid/prevent problems

## Common Troubleshooting Scenarios

- Enabling TLS results in bad performance
- Common network problems: `java.io broken pipe`
- Unable to resolve hostname
- Timeout between ZK and broker
- VMs fight for shared resources
- No resource limits set for containers
- OOM on JVM
- AD integration and Kerberos
- Number of open files

## Broker Metrics for Troubleshooting (1)

- Kafka-specific metrics

```
kafka.network:type=RequestMetrics,name=TotalTimeMs,request={Produce|FetchConsumer|FetchFollower}
```

```
kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec
```

```
kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec
```

```
kafka.server:type=KafkaServer,name=TotalDiskReadBytes
```

```
kafka.server:type=KafkaServer,name=TotalDiskWriteBytes
```

```
kafka.server:type=SessionExpireListener,name=ZooKeeperExpiresPerSec
```

## Broker Metrics for Troubleshooting (2)

- Host metrics
  - Disk usage
  - CPU usage
  - Page cache reads ratio
  - Network bytes sent/received
- GC metrics (JVM)

`java.lang:type=GarbageCollector,name=G1 Young|Old Generation`

`CollectionCount`

`CollectionTime`

## Broker - Important Files

- Location is set by the `log.dirs` broker config setting
  - If `log.dirs` is not specified, the `log.dir` config setting value is used
    - Default value for `log.dir` is `/tmp/kafka-logs`

File	Description
<code>recovery-point-offset-checkpoint</code>	Last offset flushed to disk
<code>cleaner-offset-checkpoint</code>	Offset up to which the cleaner has cleaned (compacted topics only)
<code>replication-offset-checkpoint</code>	Last committed offset
<code>log-start-offset-checkpoint</code>	Internal broker log where Kafka tracks the starting offset of the topic partition
<code>leader-epoch-checkpoint</code>	Per partition. Contains rows with <b>epoch</b> and <b>offset</b> . Each row is a checkpoint for the latest recorded leader epoch and the leader's latest offset upon becoming leader.

## Is Broker In Sync?

- Broker maintains session with ZooKeeper
- If broker hosts a **Follower**, don't lag behind "too much"

# Synchronize Broker Configs

The screenshot shows the Confluent Cloud interface for managing Kafka clusters. On the left, a sidebar lists various cluster configurations: Overview, Brokers, Topics, Connect, ksqlDB, Consumers, Replicators, and Cluster settings. The 'Cluster settings' option is selected. The main area is titled 'Cluster settings' and shows the 'Brokers' tab. A checkbox labeled 'Hide settings set to default value' is checked. Below it are three buttons: 'Edit Settings' and 'Download', with 'Download' being highlighted by a red rectangle. The configuration sections include:

- GENERAL**:
  - broker.id: 3 different values\* (with broker.101, broker.102, and broker.103 listed)
- LISTENER**:
  - listeners: 3 different values\* (with broker.101, broker.102, and broker.103 listed, each with a DOCKER URL)
  - inter.broker.listener.name: DOCKER
  - advertised.listeners: 3 different values\* (with broker.101, broker.102, and broker.103 listed, each with a DOCKER URL)
- LOG**:
  - log.dirs: /var/lib/kafka/data
  - auto.create.topics.enable: true
  - delete.topic.enable: true

## Controller Issues

***The Controller is the "brain" of the cluster***

- Symptoms of Controller issues:
  - ISR updates are not happening
  - Replication stopped for no reason
- Causes of Controller issues:
  - Broker failure
  - Admin removed the `/controller` path in ZK

## Add more Storage to Broker

- Using a partition reassignment tool:
  - Confluent Auto Data Balancer
  - `kafka-reassign-partitions`
    1. Using one of the tools, move all partitions off the broker
    2. Add new disks or SSDs
    3. Stop broker daemon gracefully
    4. Modify `log.dir` in `server.properties` with new data folder(s)
    5. Start broker daemon
    6. Using one of the tools, add the original partitions back to broker

# Gracefully Shut Down Kafka Cluster

Prerequisite configuration settings:

- All brokers `controlled.shutdown.enable=true`
- Use defaults for `controlled.shutdown.max.retries` and `controlled.shutdown.retry.backoff.ms`

Process:

1. Stop all clients
2. Shutdown one broker at a time
3. Wait for completion; observe log:
  - ...Starting controlled shutdown...
  - ...Controlled shutdown succeeded...
  - ...shut down completed...

## Java Heap Dump for Support

- Automatically upon **OutOfMemoryError**

```
-XX:+HeapDumpOnOutOfMemoryError -XXHeapDumpPath=<file_name>
```

- Manually using:

- **jmap tool**

```
jmap -dump:format=b,file=<file_name> <pid>
```

- **jconsole**
  - **jvisualvm**

# JVM GC Logging

1. Location for log dumps
2. Number of logs
3. Size of log files (best ~5MB)

```
export KAFKA_OPTS="-Xloggc:<location>`date +%F_%H-%M-%S`-gc.log -XX:+PrintGCDetails  
-XX:+PrintGCDateStamps -XX:+PrintTenuringDistribution -XX:+PrintGCCause  
-XX:+UseGCLogFileRotation -XX:NumberOfGCLogFiles=<num> -XX:GCLogFileSize=<size>M"
```

4. Restart brokers

## Recommendations

- Add more brokers
- Make sure all brokers use (similar) SSDs
- All brokers have the same configuration
- Brokers are located in same region
- Fast network connections and dedicated NICs
- Limit rogue clients with quotas



## Question:

- Assuming you do not have access to Confluent Control Center. How would you make sure all brokers of your cluster are using the same configuration values?
- How many in-sync replicas should you minimally have in production to guarantee fault tolerance?

## Further Reading

- Kafka Replication:  
<https://cwiki.apache.org/confluence/display/KAFKA/Kafka+Replication>
- Hands-free Kafka Replication: A lesson in operational simplicity  
<https://www.confluent.io/blog/hands-free-kafka-replication-a-lesson-in-operational-simplicity/>

## b: Replication Concerns

### Description

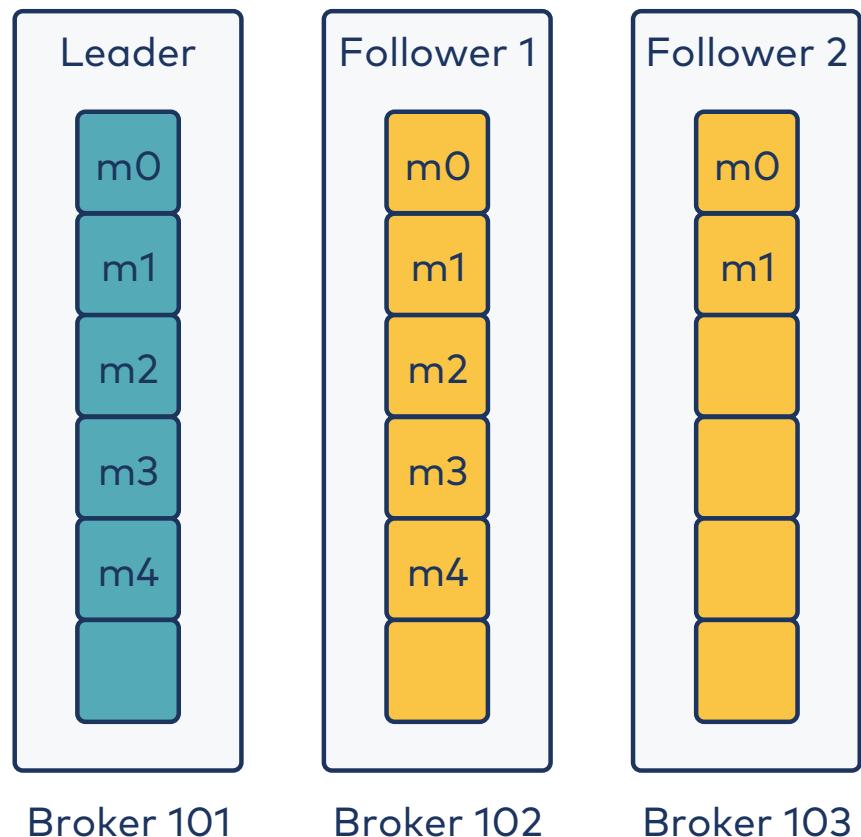
Review of replication, metrics related to replication, and tips for troubleshooting replication-related issues.

## Recap

- Messages are **only** committed if all ISRs have persisted it
- Consumers **only** see committed messages
- Producers can choose between:
  - latency: `acks=0 | 1`
  - durability: `acks=ALL`

## What is an out-of-sync replica?

- Follower can fall behind or crash
- Follower is "sufficiently" behind leader



## What to monitor for?

- ISR shrinks

`IsrShrinksPerSec`

`IsrExpandsPerSec`

`kafka-topics --describe ...`

- Under-replicated partitions

`UnderReplicatedPartitions`

`OfflinePartitionsCount`

- Unclean leader elections

`LeaderElectionRateAndTimeMs`

`UncleanLeaderElectionsPerSec`

## What happens if too few ISRs?

- Assumptions:
  - Producer uses `acks=all`
  - `min.insync.replicas` set (typically `=2`)
- Producer receives **error** when too few ISRs
  - `NOT_ENOUGH_REPLICAS`
  - `NOT_ENOUGH_REPLICAS_AFTER_APPEND`

## Why is my replica out of sync

- Slow replica
- Stuck replica
- Bootstrapping replica



# Lab: Troubleshooting Brokers

Please work on **Lab 8a: Troubleshooting Brokers**

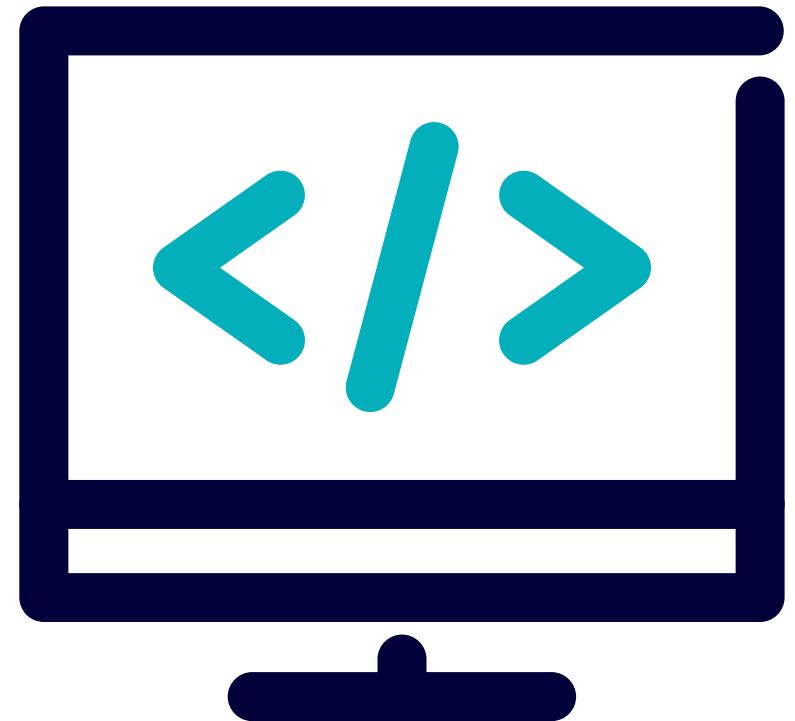
Refer to the Exercise Guide



# Lab: Not Enough ISRs

Please work on **Lab 8b: Not Enough ISRs**

Refer to the Exercise Guide



# c: Tuning Brokers: General Concepts & Best Practices

## Description

Broker monitoring, general guidelines, performance checks.

## Broker Best-Practices

- Tuning
  - Stick (mostly) with the defaults
  - Set default cluster retention as appropriate
  - Default partition count should be at least the number of brokers
- Monitoring
  - Watch the right things
  - Don't try to alert on everything
- Triage and Resolution
  - Solve problems, don't mask them

## Broker Monitoring

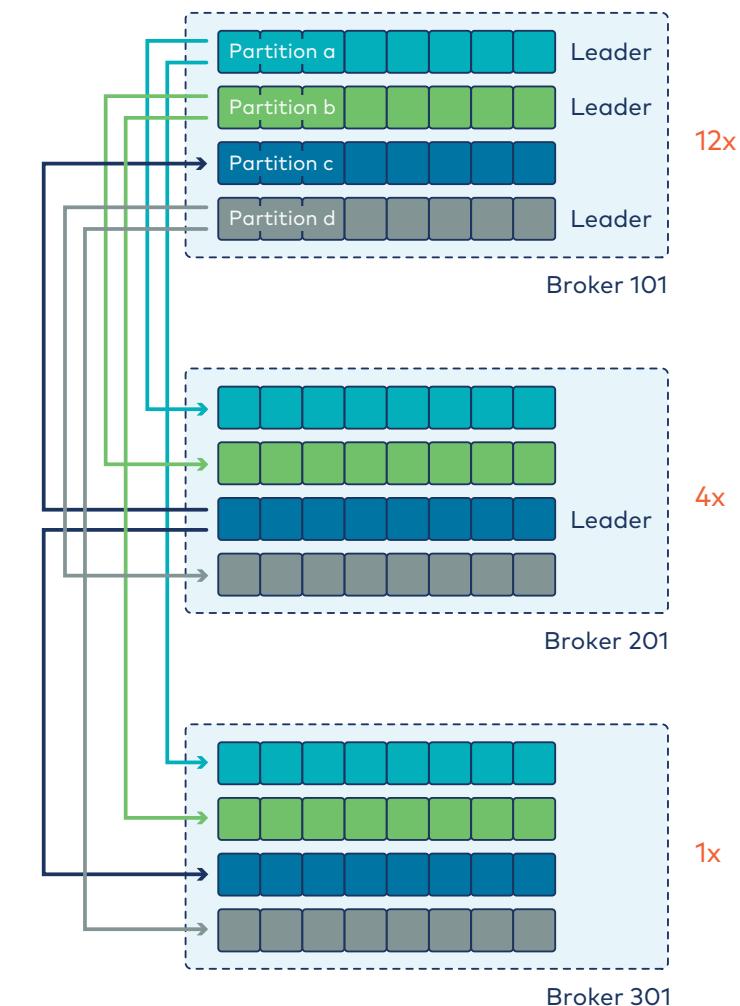
- Bytes in/out; Messages in
- Partitions
  - Count and leader count
  - Under replicated and offline
- Threads
  - Network pool, request pool
  - Max. dirty percent
- Requests
  - Rates and times - total, queue, local, and send

## Topic Monitoring

- Bytes in/out
- Messages in, Produce rate, Produce failure rate
- Fetch rate, fetch failure rate
- Partition bytes
- Quota throttling

## General Guidelines

- Evenly distribute partition leadership
- Provision sufficient memory
- Avoid version mismatches
- Mind your logging (Log4j)
- Compacted topics require extra resources
- Set User Limits



## Broker Performance Checks

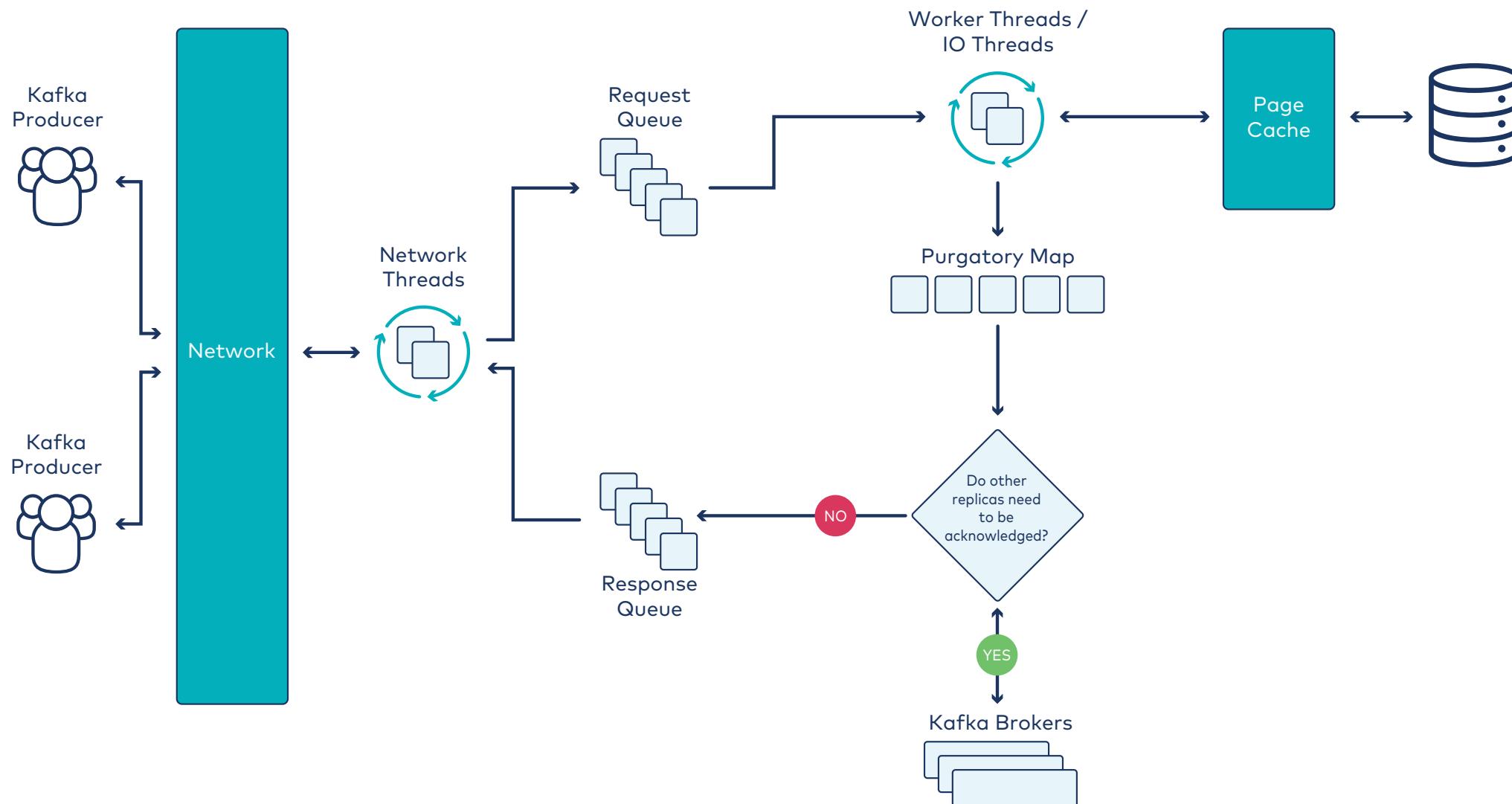
- Are all brokers in cluster working?
- Are network interfaces saturated?
- Is CPU utilization high?
- Do you have really big messages?

## d: Optimization of a Message's Life Cycle on a Broker

### Description

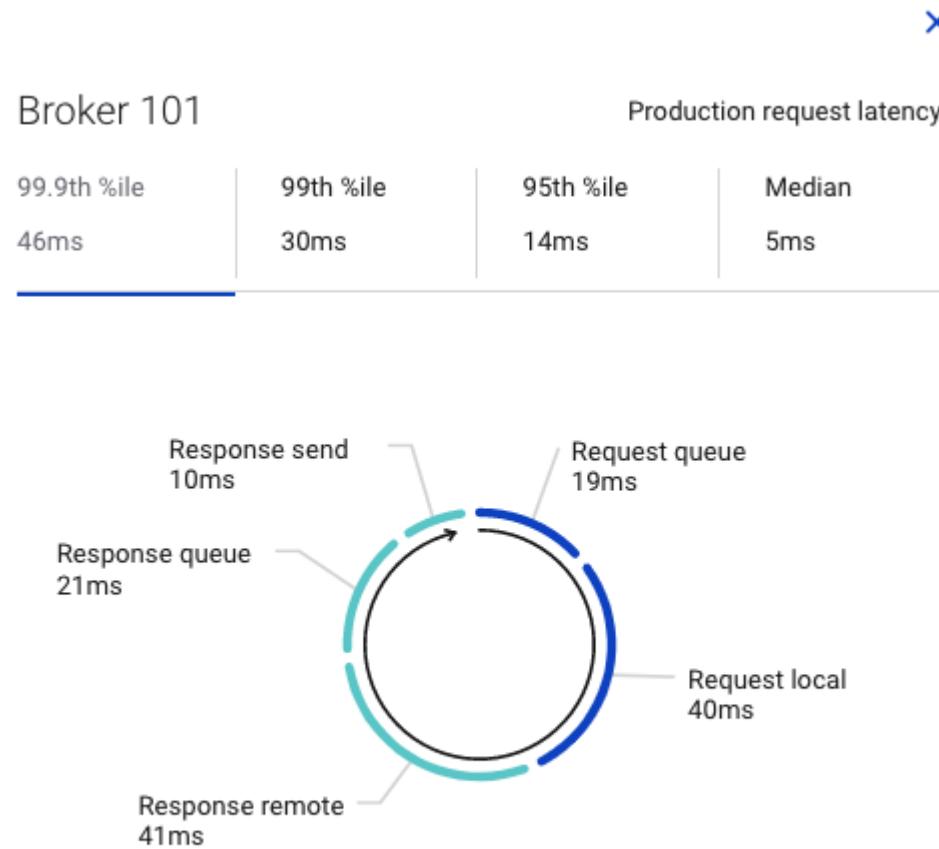
Review of the life cycle of requests on a broker. Metrics overall and at various stages.  
Tips for mitigating high latencies.

# Producer Request



# Producer Request in Control Center

JMX Metrics: `kafka.network:type=RequestMetrics,request=Produce,name=<name>`



where `<name>` is:

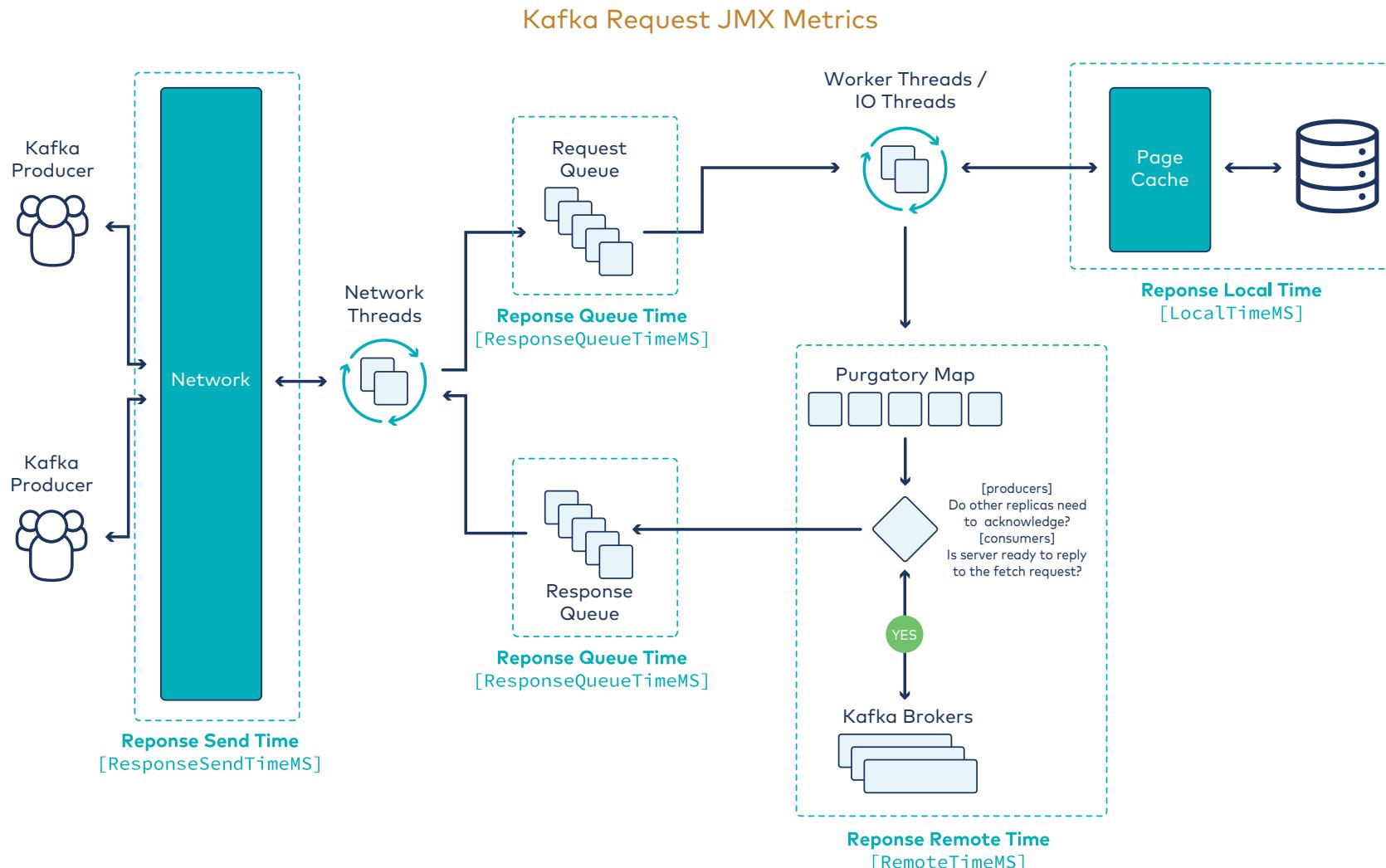
- `RequestQueueTimeMs`
- `LocalTimeMs`
- `RemoteTimeMs`
- `ResponseQueueTimeMs`
- `ResponseSendTimeMs`
- `TotalTimeMs`

## Producer Request - Latency Explained

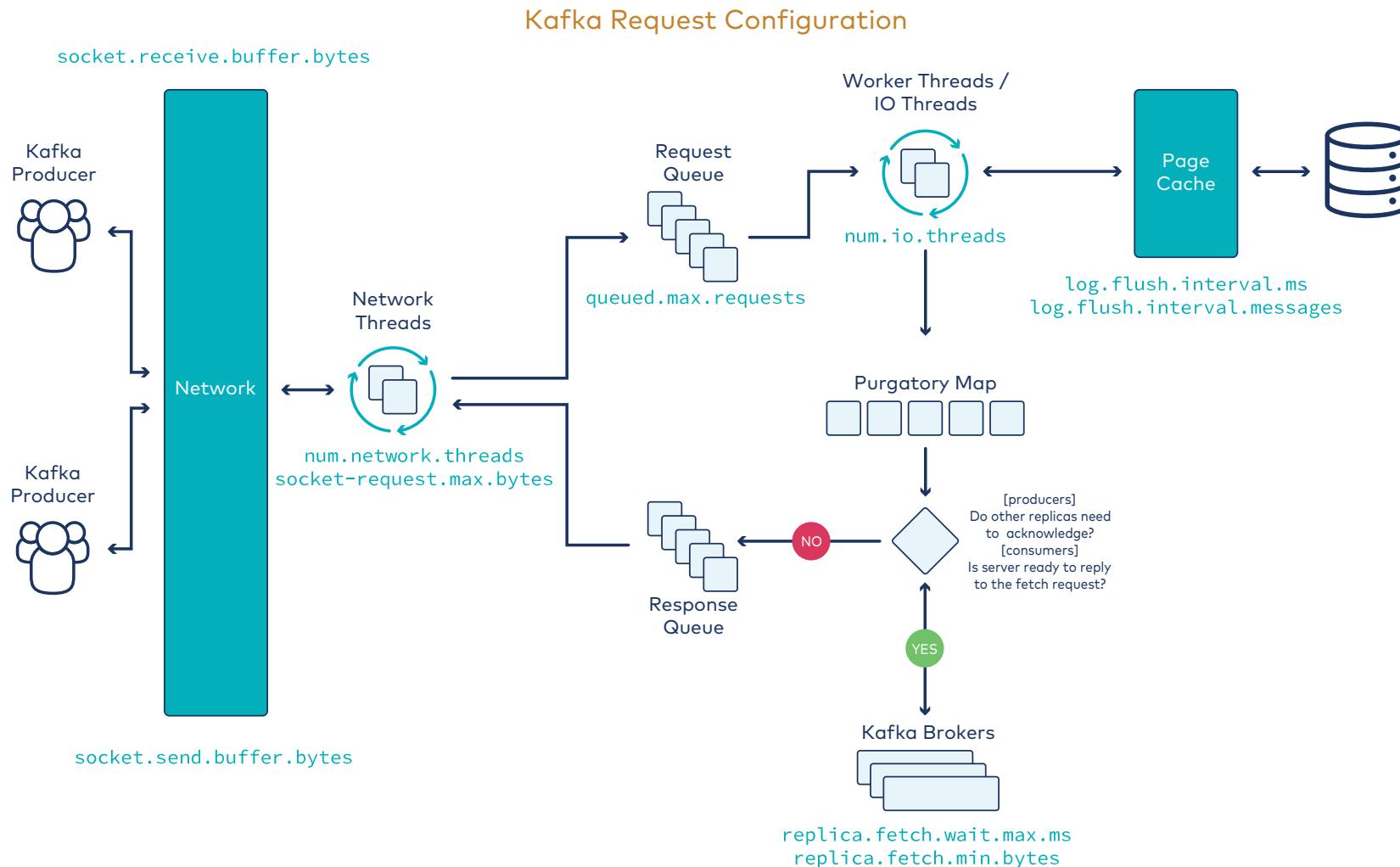
Break down `TotalTimeMs` further to see the entire request lifecycle:

Metric	Description
<code>RequestQueueTimeMs</code>	Time the request waits in the request queue
<code>ResponseSendTimeMs</code>	Time to send the response
<code>ResponseQueueTimeMs</code>	Time the request waits in the response queue
<code>LocalTimeMs</code>	Time the request is processed at the leader
<code>RemoteTimeMs</code>	Time the request waits for the follower

# Monitoring Requests on the Broker



# Configuring Requests on the Broker



# Mitigate High Latencies

Request queue	<ul style="list-style-type: none"><li>increase the pool of I/O threads (<code>num.io.threads</code>)</li></ul>
Request local	<ul style="list-style-type: none"><li>use dedicated, faster local disks (SSD)</li><li>tune <code>log.flush.interval.messages</code> &amp; <code>log.flush.interval.ms</code></li></ul>
Response remote	<ul style="list-style-type: none"><li>make sure brokers are co-located</li><li>use high bandwidth network (1 GbE, 10 GbE)</li><li>use dedicated NICs for brokers</li><li>assure all brokers nodes are equal</li><li>do you really need <code>acks=all</code>?</li></ul>
Response queue	<ul style="list-style-type: none"><li>use high bandwidth network (1 GbE, 10 GbE)</li><li>use dedicated NICs for brokers</li></ul>

## e: Miscellaneous Matters

### Description

Broker monitoring, memory usage, optimization.

# Monitoring

Monitor the following resources:

- Network throughput
- Disk I/O
- Disk space
- CPU usage

# Optimization per Service Agreement

## Optimize Latency

- `num.replica.fetchers=<value>`

## Optimize Durability

- `default.replication.factor=3`
- `auto.create.topics.enable=false`
- `min.insync.replicas=2`
- `unclean.leader.election.enable=false`
- `broker.rack=<rack ID>`
- `log.flush.interval.messages`
- `log.flush.interval.ms`

## Optimize Throughput

- **Nothing to do**

## Optimize Availability

- `unclean.leader.election.enable=true`
- `min.insync.replicas=1`
- `num.recovery.threads.per.data.dir`

## Memory Usage

- `replica.fetch.max.bytes`: 1MB allocated per replicated partition

## f: Confluent Auto Data Balancer

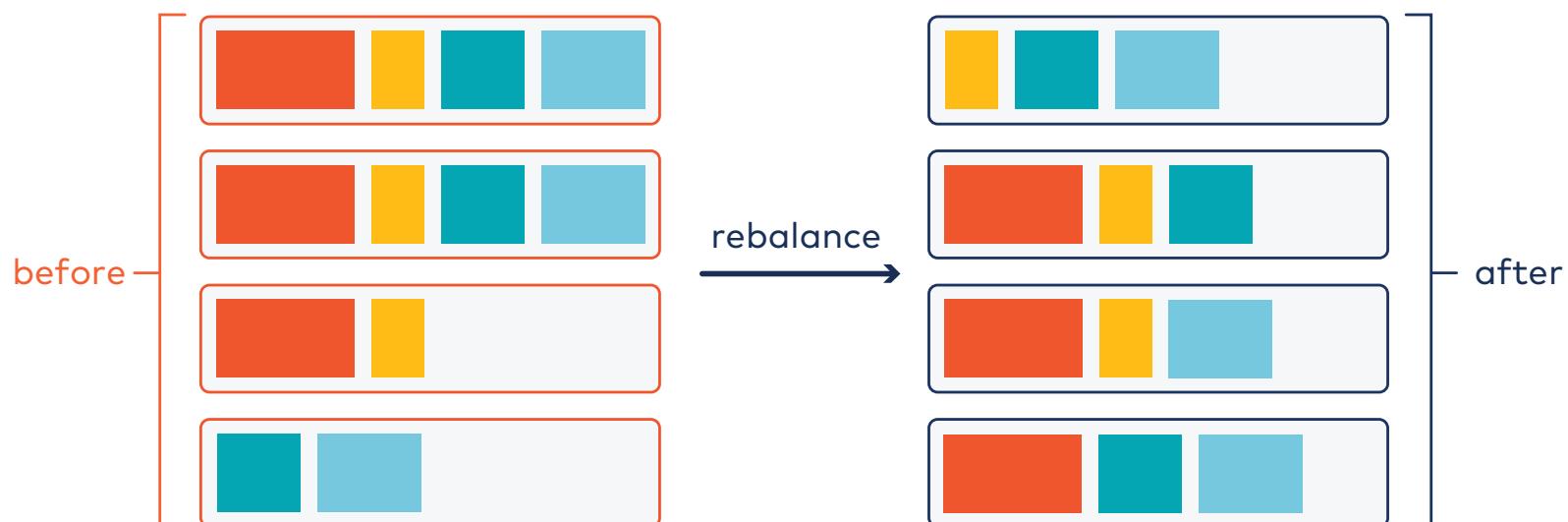
### Description

Overview of Auto Data Balancer.

# Confluent Auto Data Balancer (ADB)

Dynamically move partitions to optimize resource utilization and reliability

- Shift data to create an even workload across your cluster
- Easily add and remove nodes from your Kafka cluster
- Rack aware algorithm rebalances partitions across a cluster
- Traffic from balancer is throttled when data transfer occurs



# Confluent Auto Data Balancer (ADB)

```
$ confluent-rebalancer execute \
  --bootstrap-server kafka:9092 \
  --metrics-bootstrap-server kafka:9092 \
  --throttle 10000000 \
  --verbose
```

# Confluent Auto Data Balancer (ADB)

```
Computing the rebalance plan (this may take a while) ...
You are about to move 17 replica(s) for 14 partitions to 4 broker(s) with total size 827.2 MB.
The preferred leader for 14 partition(s) will be changed.
In total, the assignment for 15 partitions will be changed.
The minimum free volume space is set to 20.0%.
```

The following brokers will have less than **40%** of free volume space during the rebalance:

Broker	Current Size (MB)	Size During Rebalance (MB)	Free % During Rebalance	Size After Rebalance (MB)	Free % After Rebalance
0	413.6	620.4	30.1	519.6	30.5
2	620.4	723.8	30.1	520.8	30.5
3	0	517	30.1	520.8	30.5
1	1,034	1,034	30.1	519.6	30.5

Min/max stats **for** brokers (before -> after):

Type	Leader Count	Replica Count	Size (MB)
Min	12 (id: 3) -> 17 (id: 0)	37 (id: 3) -> 43 (id: 3)	0 (id: 3) -> 517 (id: 1)
Max	21 (id: 0) -> 17 (id: 0)	51 (id: 1) -> 45 (id: 0)	1,034 (id: 1) -> 517 (id: 3)

No racks are defined.

Broker stats (before -> after):

Broker	Leader Count	Replica Count	Size (MB)	Free Space (%)
0	21 -> 17	48 -> 45	413.6 -> 517	30.5 -> 30.5
1	20 -> 17	51 -> 44	1,034 -> 517	30.5 -> 30.5
2	15 -> 17	40 -> 44	620.4 -> 517	30.5 -> 30.5
3	12 -> 17	37 -> 43	0 -> 517	30.5 -> 30.5

Would you like to **continue?** (y/n):

Rebalance started, its status can be checked via the status command.

Warning: You must run the status or finish **command** periodically, **until** the rebalance completes, to ensure the throttle is removed. You can also alter the throttle by re-running the execute **command** passing a new value.

## g: Message Delivery Guarantees

### Description

Message delivery guarantees. High-level view of idempotence and transactions.

## Message Delivery Semantics

**At most once** - messages may be lost but are never redelivered.

**At least once** - messages are never lost but may be redelivered.

**Exactly once** - each message is delivered **once and only once**.

## Idempotent Producer

Idempotent Producer == Exactly once per Partition

- No data loss
- No duplicates
- In-order semantics

```
enable.idempotence=true
```

# Atomic Transactions

Simplified sample code...

```
1 producer.initTransactions();
2
3 try
4 {
5     producer.beginTransaction();
6     producer.send(...);
7     producer.send(...);
8     producer.send(...);
9     producer.commitTransaction();
10 }
11 catch(ProducerFencedException pfe)
12 {
13     producer.close();
14 }
15 catch(KafkaException ke)
16 {
17     producer.abortTransaction();
18 }
```

## Consumers and EOS

- Use `isolation.level=read_committed`
- Commit offsets with computed results in transaction



## Question:

- You are going to add new brokers to your Kafka cluster. What are actions you're going to execute to balance the load among all brokers evenly?
- Do you think EOS should be the default for Kafka? If yes, why? Why not?

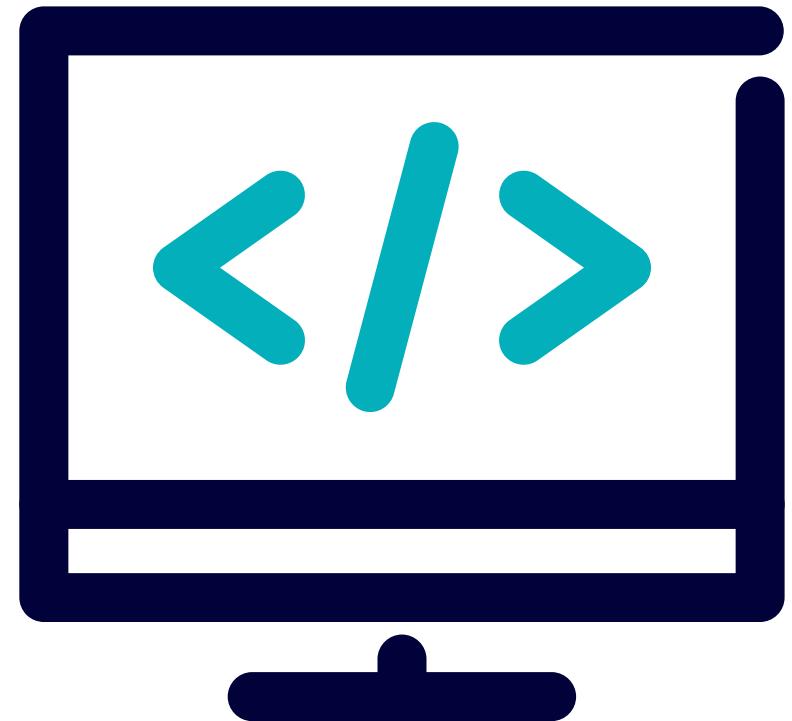
## Further Reading

- Apache Kafka, Purgatory, and Hierarchical Timing Wheels:  
<https://www.confluent.io/blog/apache-kafka-purgatory-hierarchical-timing-wheels/>
- Kafka Protocol Guide: <http://kafka.apache.org/protocol.html>
- Confluent Auto Data Balancer:  
<https://docs.confluent.io/current/kafka/rebalancer/rebalancer.html>
- Transactions in Apache Kafka: <https://www.confluent.io/blog/transactions-apache-kafka/>
- Exactly-once Semantics are Possible: Here's How Kafka Does it:  
<https://cnfl.io/kafka-eos>
- Introducing Exactly Once Semantics in Apache Kafka: <https://cnfl.io/intro-eos>
- Exactly-once Support in Apache Kafka:  
<https://medium.com/@jaykreps/exactly-once-support-in-apache-kafka-55e1fdd0a35f>

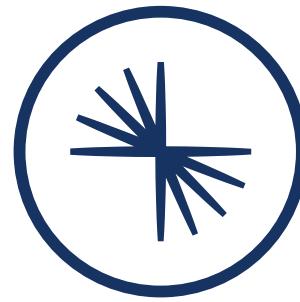
# Lab: Tuning Brokers

Please work on **Lab 8c: Tuning Brokers**

Refer to the Exercise Guide



# 09: Troubleshooting & Tuning Schema Registry



CONFLUENT  
Global Education

# Module Overview



This module contains one lesson:

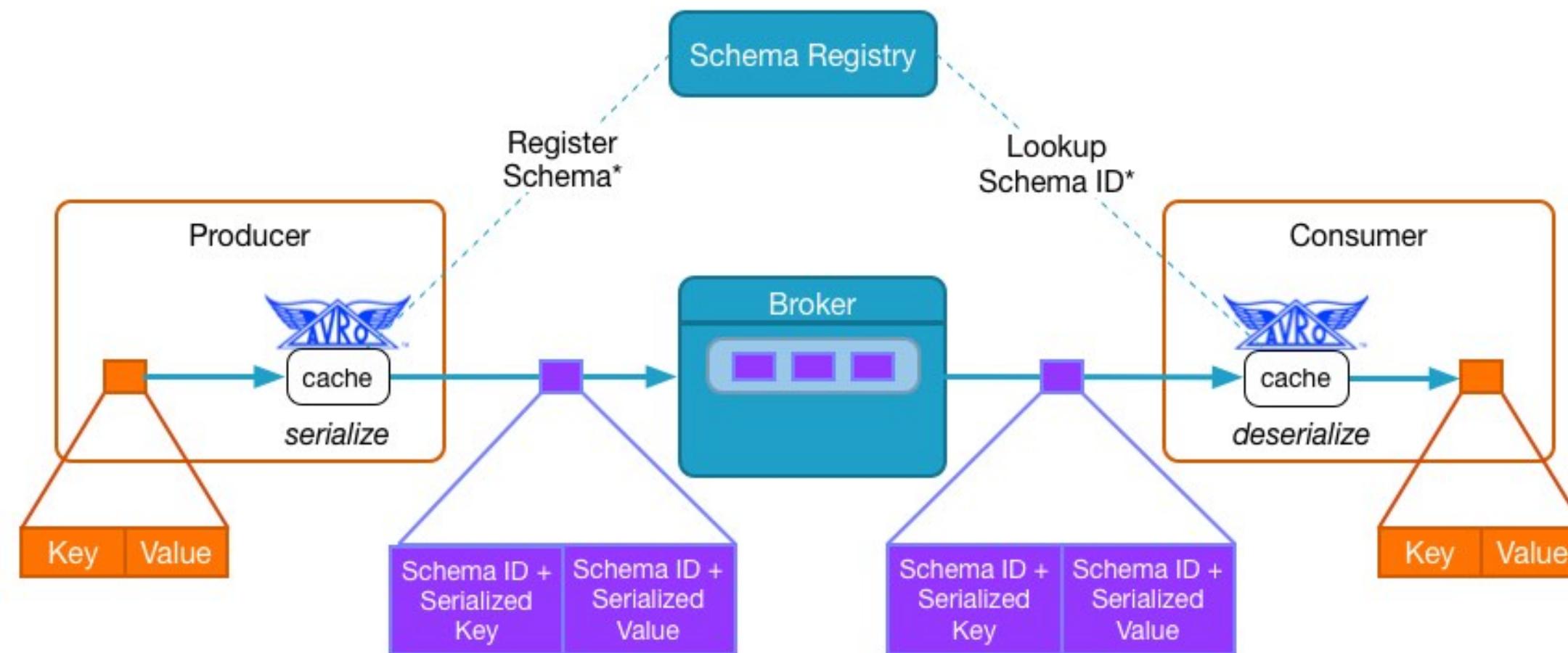
1. Troubleshooting Schema Registry

## a: Troubleshooting the Schema Registry

### Description

Review of schema registry and assorted issues that commonly arise with schema registry deployment.

# Confluent Schema Registry



## Common Issues - Bad Design

- Co-locating SR with Kafka Broker
- Multiple SRs in Company
- Wrong setup of SR in multi-DC
- No use of VIP

## Common Issues - Inconsistent Configurations

- Different names for schemas topic
- Mixing election modes
- Different settings between instances of SR
- Using same `host.name`

## Common Issues - Operational Mistakes

- SR is not secured
- Misconfigured credentials
- Manually creating schemas topic
- Deleting the schemas topic
- Not backing up the schemas topic
- Restarting SR before restoring schemas topic
- Not monitoring SR
- Wrong Java version for SR
- Poorly managed Kafka cluster

## Using Topic Schemas

The importance of shared Schema Registry:

- Tackling organizational challenges of data management
- Resilient data pipelines
- Safe schema evolution
- Storage and computation efficiency
- Data discovery
- Cost-efficient ecosystem
- Data policy enforcement

# Schema Compatibility Rules

- BACKWARD
- BACKWARD\_TRANSITIVE
- FORWARD
- FORWARD\_TRANSITIVE
- FULL
- FULL\_TRANSITIVE
- NONE

## Schema Evolution

- Cannot change field data type
  - Exception - data type widening is allowed

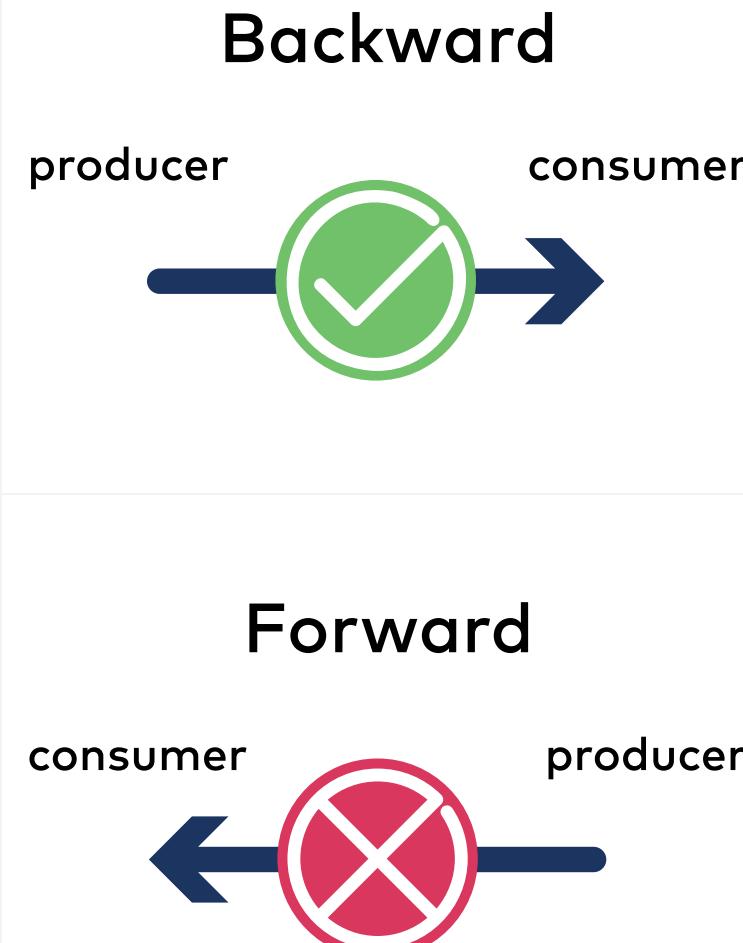
## Example

## Schema V1 fields

```
"fields":  
[  
  {  
    "name": "firstname"  
    "type": "string"  
  },  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": "-1"  
  }  
]
```

## Schema V2 fields

```
"fields":  
[  
  {  
    "name": "lastname",  
    "type": "string"  
  },  
  {  
    "name": "age",  
    "type": "int",  
    "default": "-1"  
  },  
  {  
    "name": "hobby",  
    "type": "string",  
    "default": ""  
  }  
]
```



# Test Compatibility of Schema

REST Endpoint:

```
POST /compatibility/subjects/<subject>/versions/<version>
```

Maven Plugin for Schema Registry:

Goal:

```
schema-registry:test-compatibility
```

```
<plugin>
  <groupId>io.confluent</groupId>
  <artifactId>kafka-schema-registry-maven-plugin</artifactId>
  <version>5.1.2</version>
  <configuration>
    <schemaRegistryUrls>
      <param>http://schema-registry:8081</param>
    </schemaRegistryUrls>
    <subjects>
      <product-key>src/main/avro/product-key.avsc</product-key>
      <product-value>src/main/avro/product-value.avsc</product-value>
    </subjects>
  </configuration>
  <goals>
    <goal>test-compatibility</goal>
  </goals>
</plugin>
```

## Subject Name Strategies

- **Subject:** scope where schemas can evolve in Schema Registry

Naming Strategies	Configurations
<ul style="list-style-type: none"><li>• <code>TopicNameStrategy</code> (default)</li><li>• <code>RecordNameStrategy</code></li><li>• <code>TopicRecordNameStrategy</code></li></ul>	<ul style="list-style-type: none"><li>• <code>key.subject.name.strategy</code></li><li>• <code>value.subject.name.strategy</code></li></ul>

- `TopicNameStrategy` example

Topic: `driver-positions`

Subjects: `driver-positions-key`  
`driver-positions-value`



### Question:

Which schema compatibility rule makes most sense for your particular use case(s)? Justify your answer.

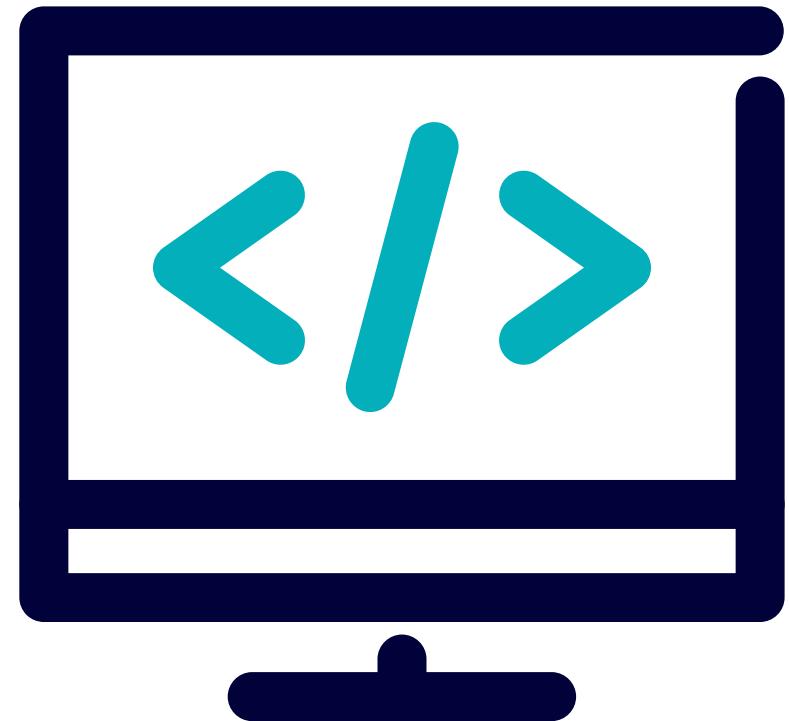
## Further Reading

- Schema Registry: <https://docs.confluent.io/current/schema-registry/docs/index.html>
- Decoupling Systems with Apache Kafka, Schema Registry and Avro:  
<https://www.confluent.io/blog/decoupling-systems-with-apache-kafka-schema-registry-and-avro/>
- Yes, Virginia, You Really Do Need a Schema Registry:  
<https://www.confluent.io/blog/schema-registry-kafka-stream-processing-yes-virginia-you-really-need-one/>
- Should You Put Several Event Types in the Same Kafka Topic?  
<https://www.confluent.io/blog/put-several-event-types-kafka-topic/>
- Schema Registry Maven Plugin:  
<https://docs.confluent.io/current/schema-registry/docs/develop/maven-plugin.html>

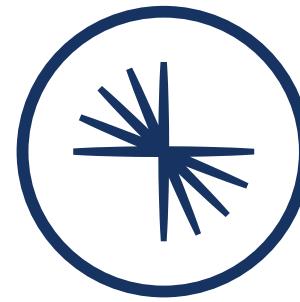
# Lab: Troubleshooting the Schema Registry

Please work on **Lab 9a: Troubleshooting the Schema Registry**

Refer to the Exercise Guide



## Branch 4: Troubleshooting & Tuning Clients - Overview



CONFLUENT  
**Global Education**

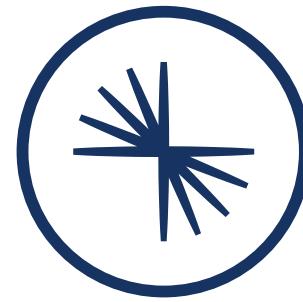
# Agenda



This is a branch of our CAO content on Troubleshooting & Tuning Clients. It is broken down into the following modules:

10. Troubleshooting & Tuning Producers
11. Troubleshooting & Tuning Consumers
12. Troubleshooting & Tuning Streams Apps
13. Troubleshooting & Tuning Kafka Connect

# 10: Troubleshooting & Tuning Producers



CONFLUENT  
**Global Education**

# Module Overview



This module contains two lessons:

1. Troubleshooting Producers
2. Tuning Producers

## a: Troubleshooting Producers

### Description

Producer metrics, common troubleshooting scenarios, librdkafka producer specifics.

# Producer Metrics for Troubleshooting

- Observe: `kafka.producer:type=producer-metrics,client-id=<client_id>`
  - Response Rate
  - Request Rate
  - Request latency avg
  - Outgoing byte rate
  - IO ratio & IO wait ratio
  - Record retry & error rate

# Common Producer Issues

## Issues:

- Cannot connect to Kafka
- Cannot write to topic
- Producer is very slow

## Troubleshooting

Use "kafkacat" to troubleshoot producer issues:

```
$ kcat -L -b kafka:9092  
  
$ cat iot_data.csv | kcat -P -p -1 -b kafka:9092 -t iot-data
```

## Troubleshoot librdkafka Clients

Context	Description	Verbosity
generic	anything generic enough not to fit the other contexts	sparse
broker	broker handling (protocol requests, queues)	medium
topic	topic and partition state changes and events	medium
queue	internal request and message queue events	low
msg	message transmission and parsing	<b>high</b>
protocol	Kafka protocol requests and responses	medium/high
cgrp	consumer group state machine	medium
security	SSL and SASL handshakes - on connect only	low
fetch	Consumer's fetcher state machine and fetch decisions	<b>high</b>
feature	Broker feature discovery - on connect only	medium
interceptor	interceptor handling and callbacks	low

## Troubleshoot librdkafka Clients

Context	Description	Verbosity
plugin	dynamic plugin loading	sparse
metadata	Topic and broker metadata updates	medium
all	enable all of the above contexts	<b>very high</b>

Common usages of the debug contexts:

- Troubleshooting common producer issues

Set `debug=broker,topic,msg`

- Troubleshooting security related issues

Set `debug=broker,security`



### Question:

How do you make sure that all messages produced are actually received by Kafka?

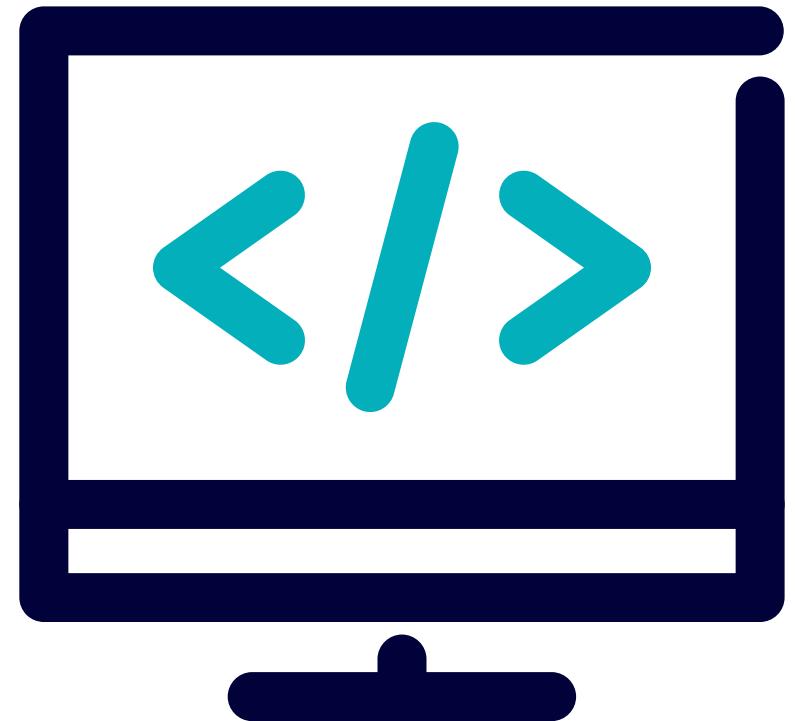
## Further Reading

- The (Kafka) Producer: <https://kafka.apache.org/documentation/#theproducer>
- librdkafka Wiki: <https://github.com/edenhill/librdkafka/wiki>

# Lab: Troubleshooting Producers

Please work on **Lab 10a: Troubleshooting Producers**

Refer to the Exercise Guide



## b: Tuning Producers

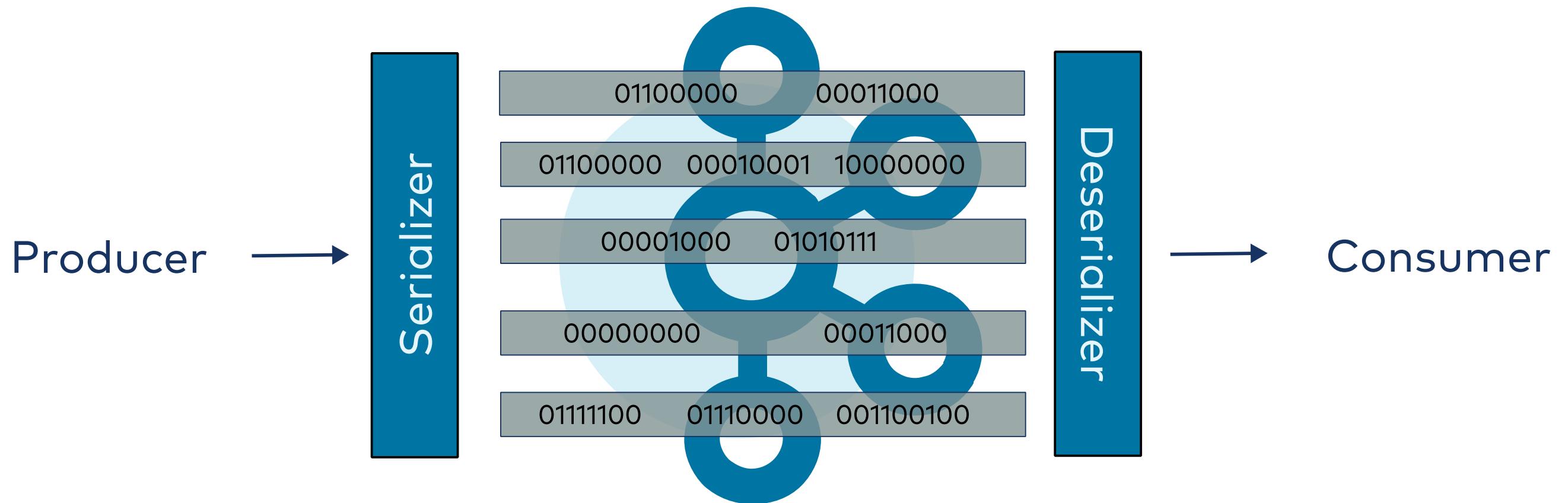
### Description

Tuning producer serialization, partitioning, and replication. Producer configs. Producer metrics. Compression. Producer tuning summary.

## Out of the box data formats

- String
- Short, Integer, Long
- Float, Double
- UUID
- JSON
- AVRO
- Protobuf
- Binary (`byte[]`, `ByteBuffer`)

# Serialization & Deserialization



## Custom SerDes for any data format

Can define custom SerDes...

1. Write **serializer**
2. Write **deserializer**
3. Write **serde**

## Why AVRO?

- Many tools to support it
- Direct mapping from/to JSON
- Much more compact than JSON
- Very fast
- Many language bindings
- Rich, extensible schema language
- Best in supporting schema evolution



Works with **Confluent Schema Registry**

## Number of Partitions

- Number of Partitions == directly proportional to scalability
- More consumer instances than partitions → waste
- Unless we use extra consumers as standby instances for fast failover

## Partitioning Strategy

- Default: Compute Hash of Record **Key**
- Select best partition strategy for given key
- Strategy should **evenly distribute** data among partitions

## Replication Quotas

- Use replication quotas to protect network bandwidth
- To be considered when using Confluent Rebalancer
- Replication quota is per broker:
  - Incoming quota: `leader.replication.throttled.rate`
  - Outgoing quota: `follower.replication.throttled.rate`



**Usually incoming = outgoing**

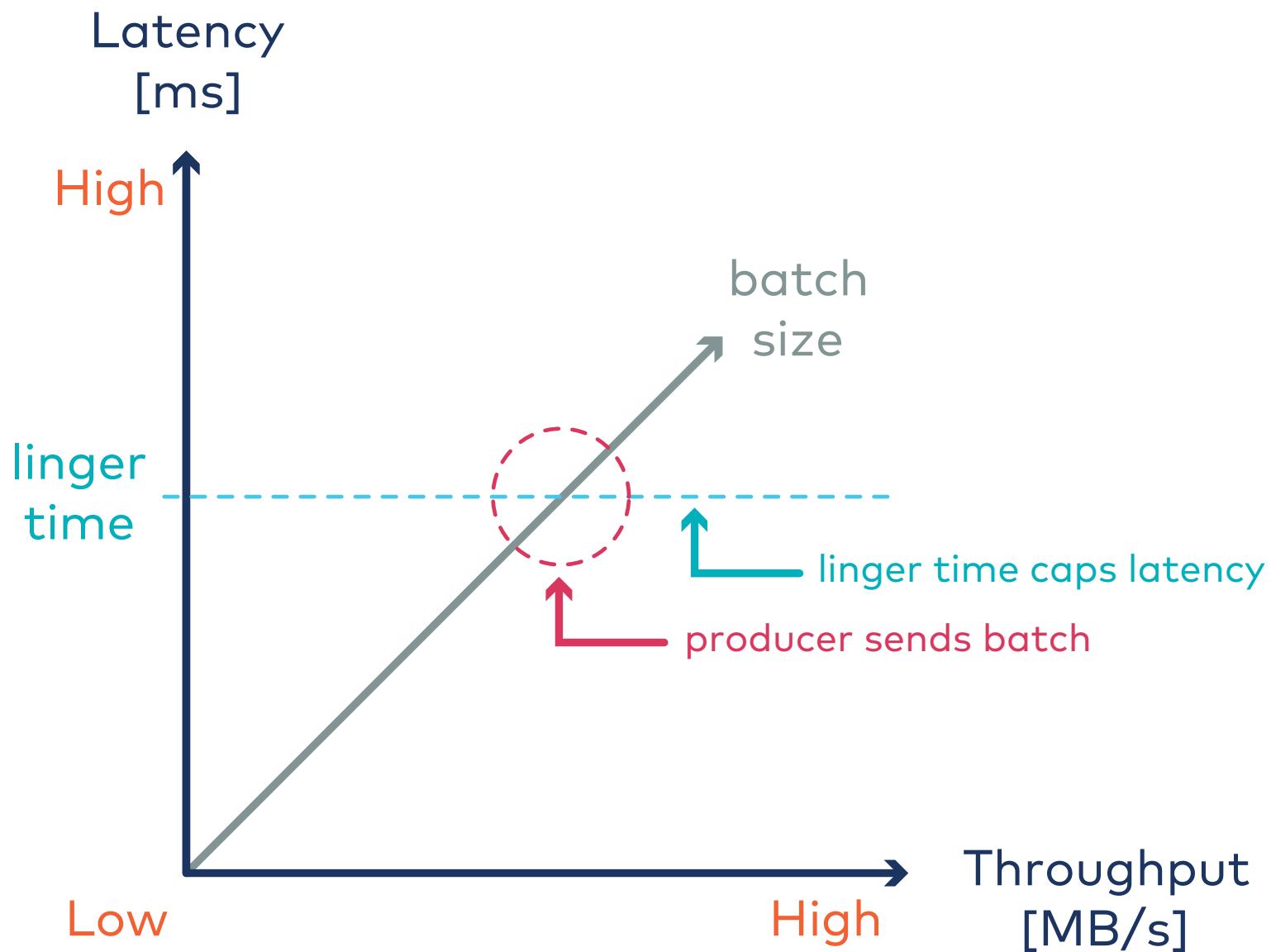
## Compacting Topics

- Kafka guarantees presence of last value for every key
- Total number of keys not known in advance
- Define hard retention policy to limit disk space usage

# Critical Configurations

- `batch.size`
- `linger.ms`
- `buffer.memory`
- `compression.type`
- `max.in.flight.requests.per.connection`
- `acks`

## Batch Size



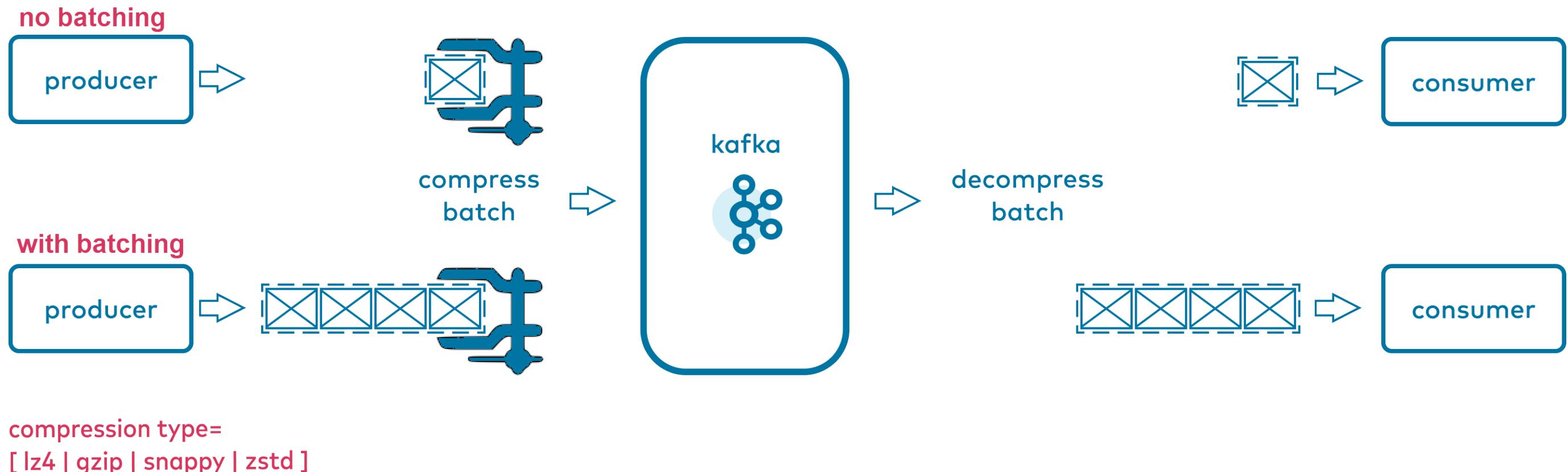
# Producer Metrics

Metric	Description
io-ratio	Fraction of time I/O thread spent doing I/O
io-wait-ratio	Fraction of time I/O thread spent waiting
1 - io-ratio - io-wait-ratio	User processing time
batch-size-avg	Average batch size
compression-rate-avg	Average compression rate

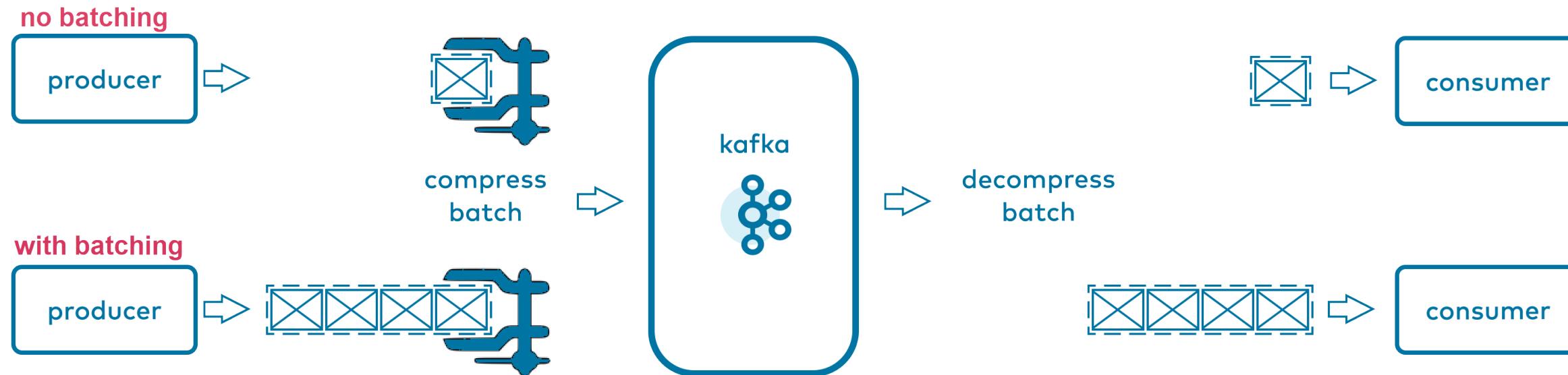
## Producer Metrics - Per Topic

Metric	Description
record-send-rate	The average number of records sent per second for a topic.
byte-rate	The average number of bytes sent per second for a topic.
record-error-rate	The average per-second number of record sends that resulted in errors for a topic.
record-retry-rate	The average per-second number of retried record sends for a topic.
compression-rate	The average compression rate of record batches for a topic.

# Producer Compression



# Broker Compression



compression type=  
[ lz4 | gzip | snappy | zstd ]

compression type=  
[ producer | lz4 | gzip | snappy | zstd ]

if `compression.type!=producer`  
then broker decompresses and recompresses batch!

## Compression Summary

- Compression is usually the **dominant** part of `producer.send()`
- Speed of different compression types differs **A LOT**
- Compression is in user thread → add more user threads if compression is slow

## Concurrent in-flight requests

- `max.in.flight.requests.per.connection>1` means pipelining
- In general, pipelining:
  - gives **better throughput**
  - may cause **out of order** delivery when retry occurs
  - **Excessive pipelining** → drop of throughput
    - lock contention
    - worse batching



**Idempotent producer** prevents out of order messages

## Tuning Scenarios (1/2)

### Optimize Throughput

- `batch.size`: increase to 100,000 - 200,000 (default 16,384)
- `linger.ms`: increase to 10 - 100 (default 0)
- `compression.type=lz4` (default none)
- `acks=1` (default 1)
- `buffer.memory`: increase if there are a lot of partitions (default 33,554,432)

### Optimize Latency

- `linger.ms=0` (default 0)
- `compression.type=None` (default none, i.e., no compression)
- `acks=1` (default 1)

## Tuning Scenarios (2/2)

### Optimize Durability

- `replication.factor=3` (topic override available)
- `acks=all` (default `1`)
- `enable.idempotence=true` (default `false`), to handle message duplication and ordering
- `max.in.flight.requests.per.connection=1` (default `5`), to prevent out of order messages when not using an idempotent producer

### Optimize Availability

- Not relevant

# Producer Performance Tool

## Sample Command:

```
$ kafka-producer-perf-test \
  --num-records 1000000 \
  --record-size 1000 \
  --topic sample-topic \
  --throughput 1000000 \
  --print-metrics \
  --producer-props bootstrap.servers=kafka:9092 \
    max.in.flight.requests.per.connection=1 \
    batch.size=100000 \
    compression.type=lz4
```

Above shell is wrapper around:

`org.apache.kafka.tools.ProducerPerformance`

## Tuning librdkafka based Clients

- Batch write requests
- Same optimizations as Java producer
- Important settings:
  - `batch.num.messages`
  - `queue.buffering.max.ms`
  - `compression.codec`
  - `request.required.acks`

## Tuning the Confluent REST Proxy

- Tune standard Java producer settings via REST Proxy's config file
- Batch write requests
- Reuse a session to push data
- Avoid large messages



### Question:

- What data formats are you envisioning to use in your Kafka powered streaming platform? What reasons did influence your choice?
- Envision a typical topic you are going to produce or use. How many partitions will you create? Justify your choice.
- Assuming you want to write a producer for highest possible throughput, where do you start to tune?

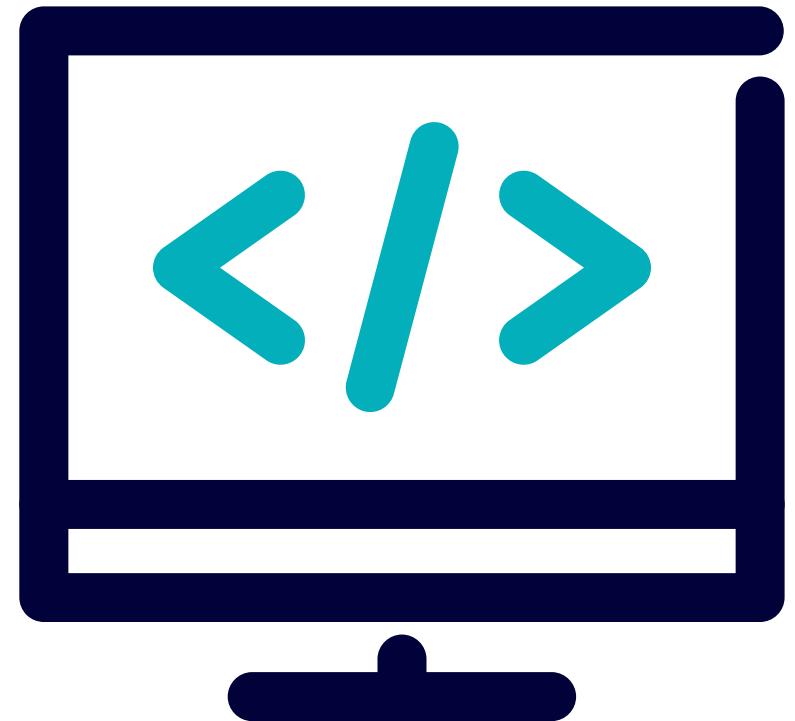
## Further Reading

- Decoupling Systems with Apache Kafka, Schema Registry and Avro: <https://cnfl.io/decoupling-systems>
- Optimizing Your Kafka Deployment: <http://cnfl.io/optimize-kafka-deployment>
- Introduction to Schemas in Apache Kafka with the Confluent Schema Registry: <https://bit.ly/2RvNkVr>
- Data Types and Serialization: <https://cnfl.io/data-types>
- The problem of managing schemas: <https://www.oreilly.com/ideas/the-problem-of-managing-schemas>
- How to choose the number of topics/partitions in a Kafka cluster? <https://cnfl.io/nbr-of-partitions>
- Reliability Guarantees in Kafka <https://cnfl.io/summit-reliability>
- Producer Perf. Tuning for Apache Kafka: <https://cnfl.io/slides-kafka-perf-tuning>
- Compression in Kafka now 34% faster: <https://cnfl.io/kafka-compression-faster>

# Lab: Tuning Producers

Please work on **Lab 10b: Tuning Producers**

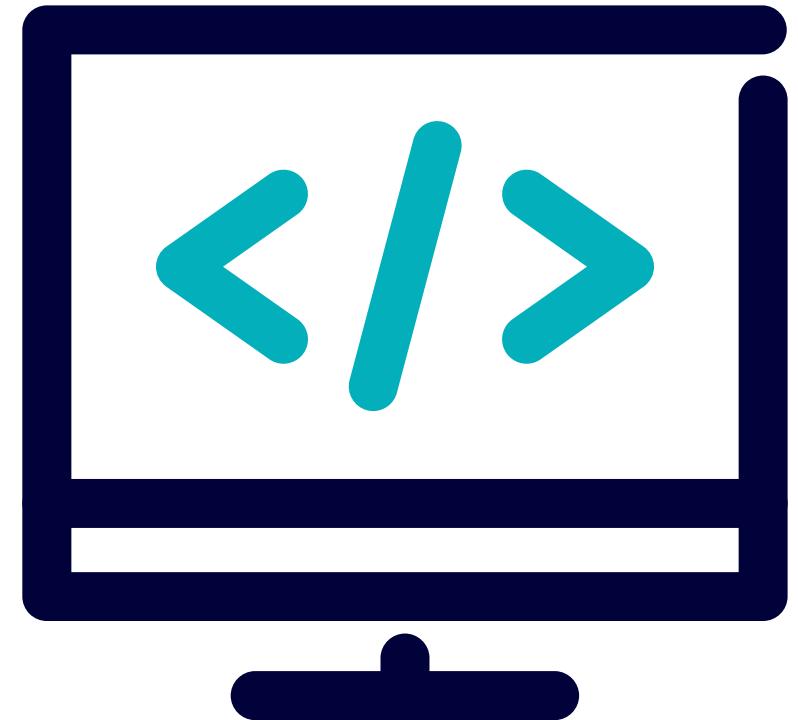
Refer to the Exercise Guide



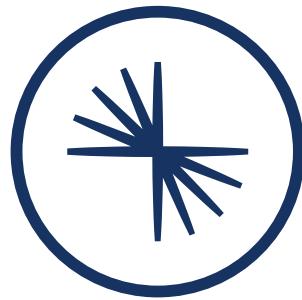
# Lab: Tuning Producers - Selecting the Best Partition Strategy

Please work on **Lab 10c: Tuning Producers - Selecting the Best Partition Strategy**

Refer to the Exercise Guide



# 11: Troubleshooting & Tuning Consumers



CONFLUENT  
**Global Education**

# Module Overview



This module contains two lessons:

1. Troubleshooting Consumers
2. Tuning Consumers

## a: Troubleshooting Consumers

### Description

Consumer lag troubleshooting. Reading and resetting offsets. Rebalancing.

# Troubleshoot Consumer Lag

Three phases:

1. Understand structure of environment
2. Determine cluster health
3. Test consumption

## Troubleshoot Consumer Lag: (1) Understand structure of environment

1. How many **brokers**?
2. Which **topic** is consumed?
3. Understand **details** of topic

## Troubleshoot Consumer Lag: (2) Determine cluster health

1. Are there under-replicated topics?
2. What is the name of the consumer group?
3. How many consumers?
4. What's their consumption status?
5. Note partitions where lag is increasing
6. Confirm current broker configuration
7. Check CPU load on brokers
8. Are JMX metrics enabled?

## Troubleshoot Consumer Lag: (3) Test consumption

1. Use `kafka-console-consumer` to consume from each affected partition

# Reset Consumer Group Offset

## Motivation:

- Reprocess old records
  - Does not require a code change

## WHY?

- Bug in consumer code is fixed
- New and better consumer algorithms/models

## Example:

- Reset to first offset since 01 January 2019, 00:00:00 hrs UTC

```
$ kafka-consumer-groups --reset-offsets \
  --group <Group ID> \
  --bootstrap-server kafka:9092 \
  --to-datetime 2019-01-01T00:00:00.000
```

## Read Consumer Group Offsets

```
$ kafka-consumer-groups --bootstrap-server kafka:9092 --describe --group my-group
```

GROUP	TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST
CLIENT-ID							
my-group	my-topic	0	2536818	2542828	6010	consumer-1...	/172.28.0.9
consumer-1							
my-group	my-topic	1	2519420	2525328	5908	consumer-1...	/172.28.0.9
consumer-1							
my-group	my-topic	4	2528207	2533934	5727	consumer-3...	/172.28.0.11
consumer-3							
my-group	my-topic	5	2510668	2515775	5107	consumer-3...	/172.28.0.11
consumer-3							
my-group	my-topic	2	2541226	2546270	5044	consumer-2...	/172.28.0.10
consumer-2							
my-group	my-topic	3	2509518	2515056	5538	consumer-2...	/172.28.0.10
consumer-2							

## Read \_\_consumer\_offsets Topic (1)

If broker is online:

```
$ kafka-console-consumer \
--consumer.config consumer.properties \
--from-beginning \
--topic __consumer_offsets \
--bootstrap-server kafka:9092 \
--formatter \
"kafka.coordinator.group.GroupMetadataManager\$/OffsetsMessageFormatter"
```

or

```
...
--formatter \
"kafka.coordinator.group.GroupMetadataManager\$/GroupMetadataMessageFormatter"
```

## Read \_\_consumer\_offsets Topic (2)

If broker is offline:

```
$ kafka-dump-log \
--files /var/lib/kafka/data/__consumer_offsets-<X>.log \
--offsets-decoder \
--print-data-log
```

# Consumer Group Rebalancing Issues

- Frequent rebalancing:

Metric: `join-rate`, `sync-rate`

- Long rebalancing times:

Metrics:

- `join-time-avg`
- `join-time-max`
- `sync-time-avg`
- `sync-time-max`

# Other Important Metrics for Troubleshooting

- Consumer lag: `records-lag-max`
- Consumer throughput:
  - `fetch-rate`
  - `fetch-latency-avg`
  - `fetch-latency-max`
  - `records-per-request-avg`
  - `bytes-consumed-rate`



### Question:

You realized that one of your consumer instances in consumer group `Foo` processed the data from partitions 1, 4 and 7 of topic `topic-1` incorrectly. You decide to start the processing of this consumer instance over from scratch. How can you do that?

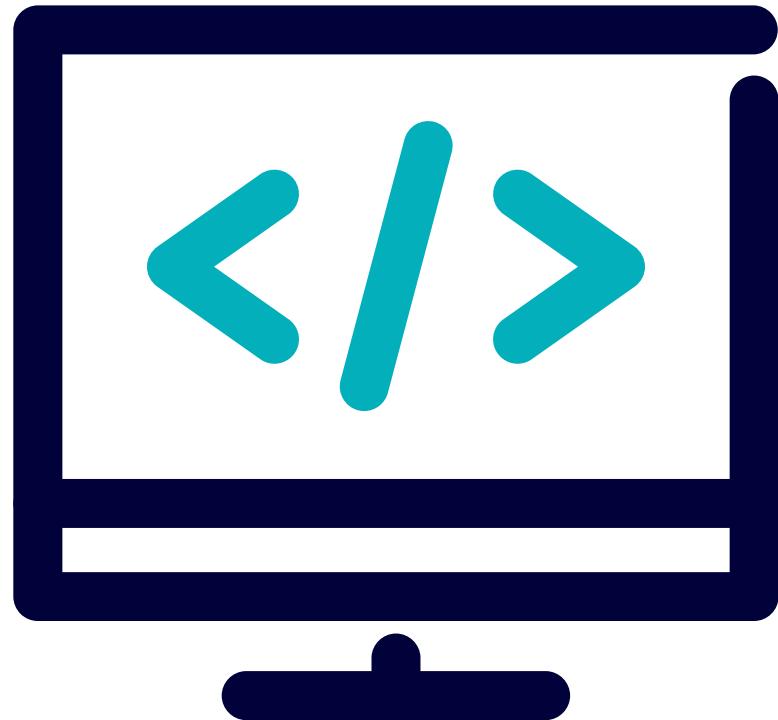
## Further Reading

- KIP-122: Add Reset Consumer Group Offset tooling:  
<https://cwiki.apache.org/confluence/display/KAFKA/KIP-122%3A+Add+Reset+Consumer+Group+Offsets+tooling>
- Managing Consumer Groups:  
[https://kafka.apache.org/documentation/#basic\\_ops\\_consumer\\_group](https://kafka.apache.org/documentation/#basic_ops_consumer_group)

# Lab: Troubleshooting Consumers

Please work on **Lab 11a: Troubleshooting Consumers**

Refer to the Exercise Guide



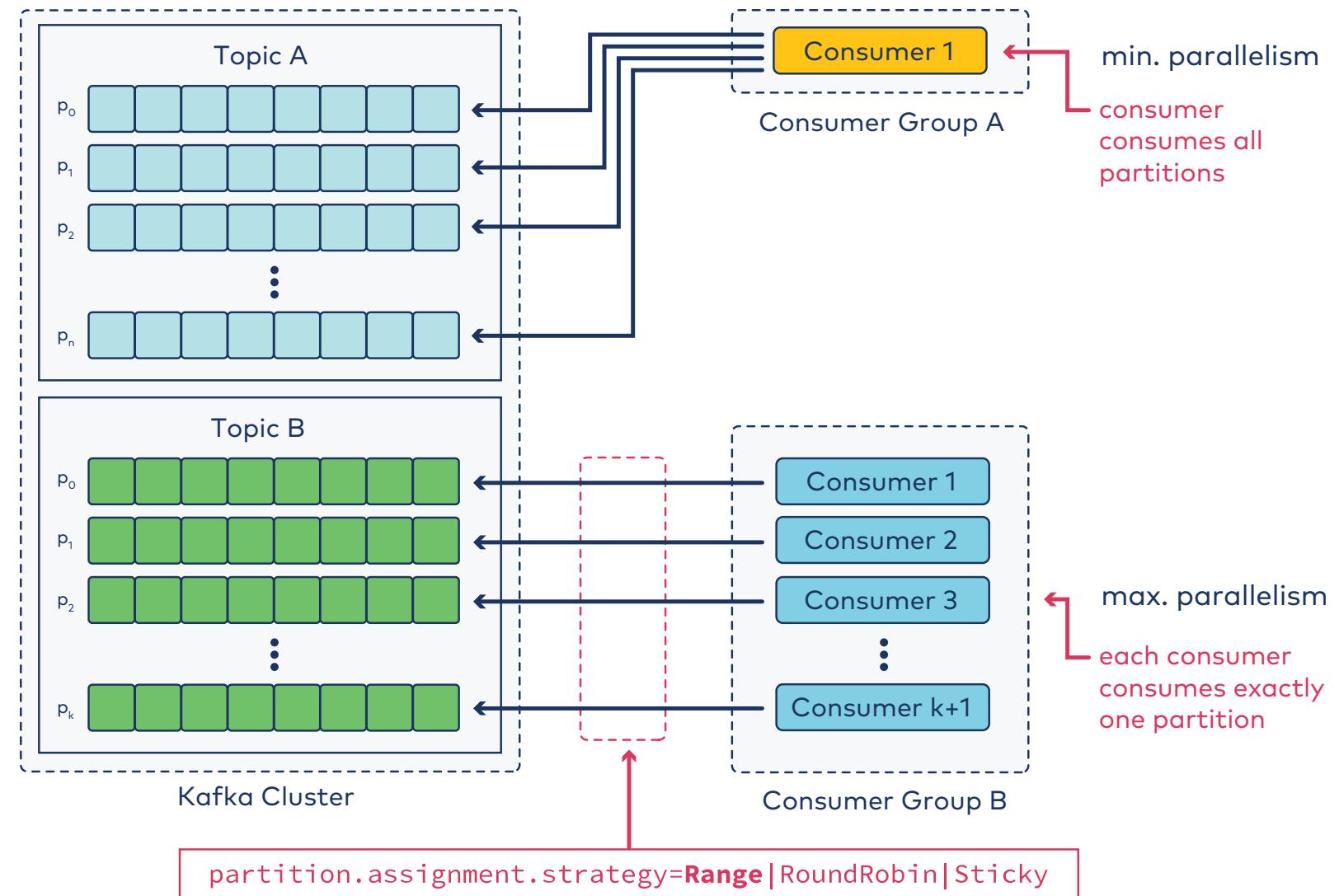
## b: Tuning Consumers

### Description

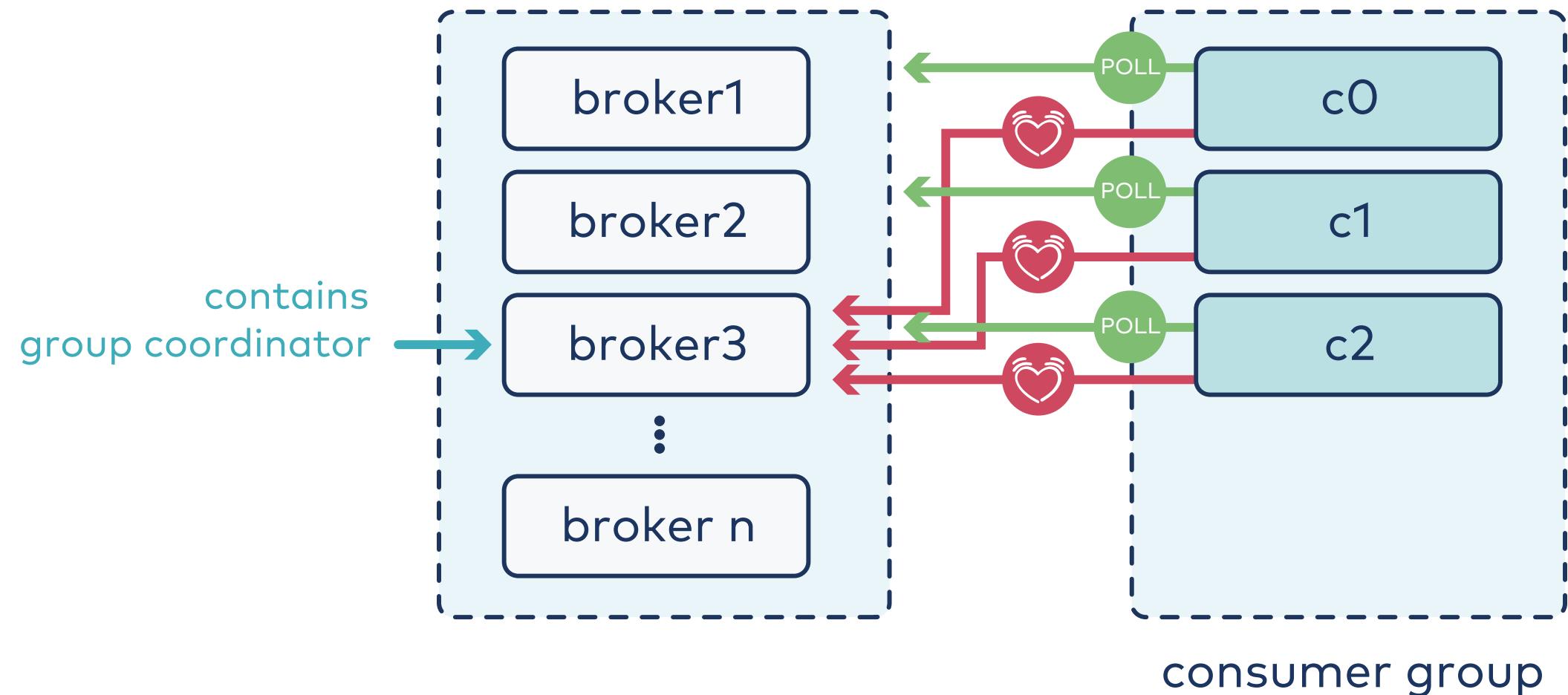
Consumer/partition parallelism. Fault detection and rebalancing. Fetch requests.

Details specific to librdkafka consumers.

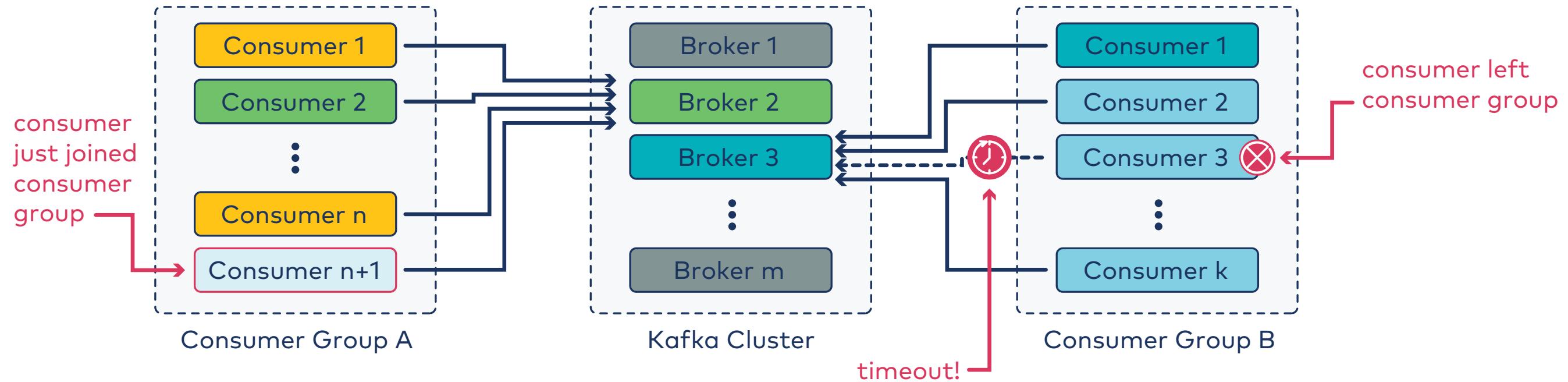
# Maximize Parallelism



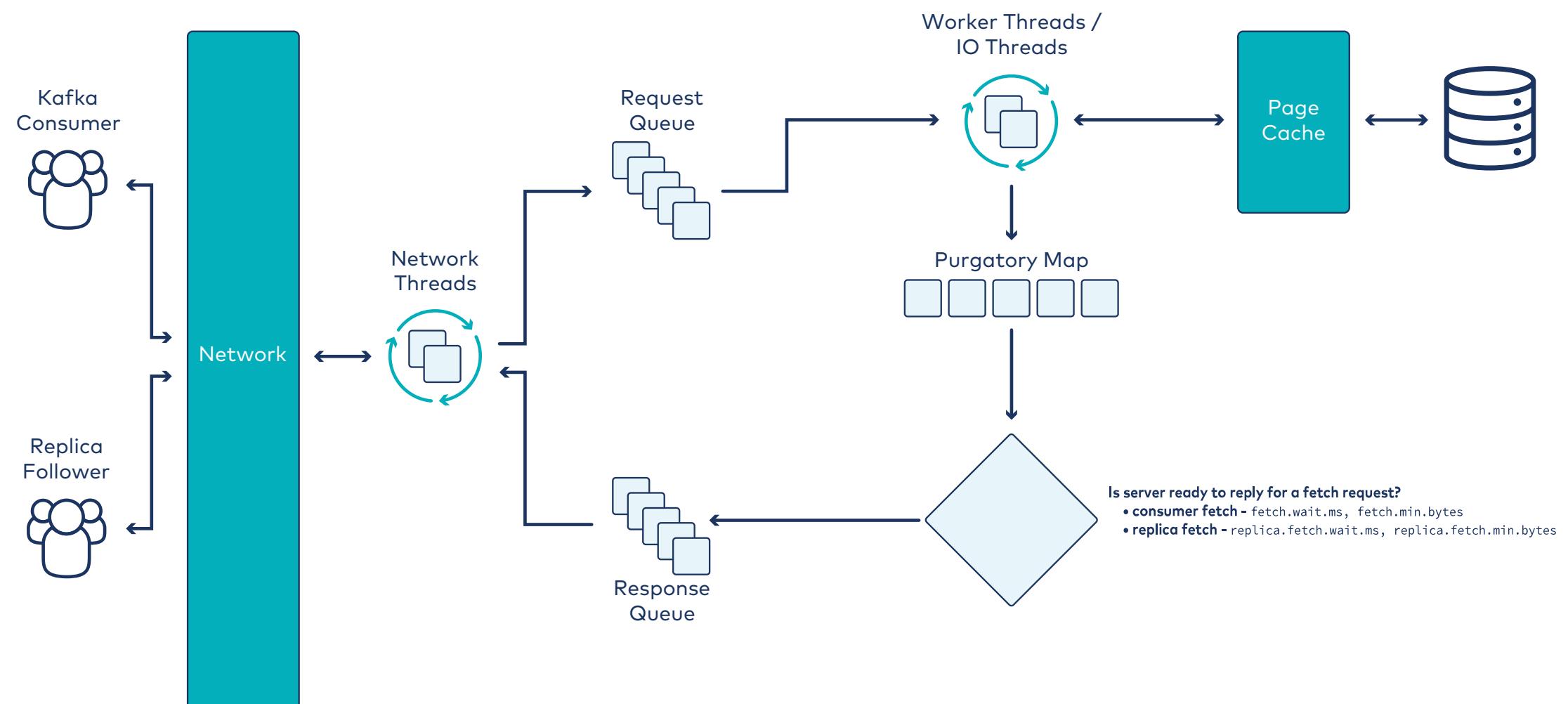
# Consumer Liveness



# Joining or Leaving a Consumer Group



# Fetch Request on a Broker



# Tuning Consumer Fetch Requests

## `fetch.min.bytes` vs. `fetch.max.wait.ms`

- What if topic does not have a lot of data?
- Reduce load on broker by letting fetch requests wait a bit for data
- Add latency to increase throughput
- Caution: do not fetch more than you can process!

## Commits Take Time

- Commit less frequently
- Commit asynchronously

## Monitoring

When tuning, observe:

- records-lag-max
- fetch-rate
- fetch-latency
- records-per-request, bytes-per-request

## Tuning via Partition Assignment Strategy

### Options:

- Range (default)
  - might not distribute workload evenly
- RoundRobin
  - uses all consumer resources evenly
- Sticky
  - preserves assignments if possible and reasonable

### Consumer property:

```
partition.assignment.strategy=Range|RoundRobin|Sticky
```

## Tuning librdkafka based Clients

- Same optimizations as Java consumer
- Important Properties:

`fetch.wait.max.ms`

`fetch.min.bytes`

- Commit less frequently
- Commit asynchronously
- Avoid large messages

## Tuning Clients that use REST Proxy

- Reuse a session to pull data
- Avoid large messages
- Tune `fetch.min.bytes` & `consumer.request.timeout.ms`

## Tuning Scenarios

- Optimize Latency

`fetch.min.bytes=1` (default 1)

- Optimize Durability

`enable.auto.commit=false` (default true)

`isolation.level=read_committed` (when using EOS transactions)

- Optimize Availability

`session.timeout.ms` (set as low as feasible, default 45 s.)



### Question:

Assuming your consumers need minimal latency, what are the first things you address? Where do you tune?

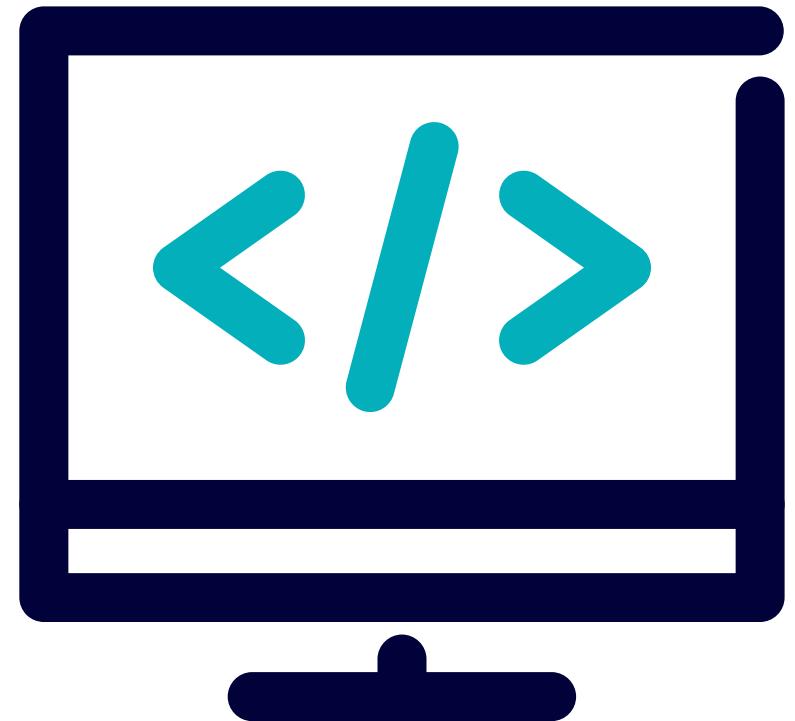
## Further Reading

- Running Kafka at Scale:  
<https://www.slideshare.net/gwenshap/kafka-at-scale-facebook-israel>
- Apache Kafka, Purgatory, and Hierarchical Timing Wheels:  
<https://www.confluent.io/blog/apache-kafka-purgatory-hierarchical-timing-wheels/>
- Kafka Protocol Guide:  
<http://kafka.apache.org/protocol.html>
- REST Proxy, Important Configuration Options:  
<https://docs.confluent.io/current/kafka-rest/docs/deployment.html#important-configuration-options>
- **librdkafka** Repository on GitHub: <https://github.com/edenhill/librdkafka>

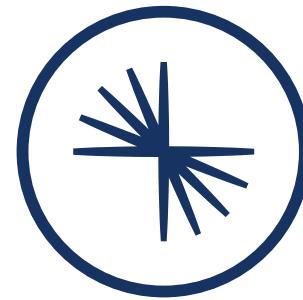
# Lab: Tuning Consumers

Please work on **Lab 11b: Tuning Consumers**

Refer to the Exercise Guide



# 12: Troubleshooting & Tuning Streams Apps



CONFLUENT  
**Global Education**

# Module Overview



This module contains two lessons:

1. Troubleshooting Streams Apps
2. Tuning Streams Apps

# a: Troubleshooting Kafka Streams & ksqlDB Apps

## Description

Common streams apps problems. Metrics. Health checks.

## Low Throughput - Kafka Streams

### Potential factors:

- low frequency input topic(s)
- available network bandwidth too low
- number of stream threads too low, or
- not enough consumer instances
- low number of partitions

### Stateful Apps:

- not enough memory
- slow state store IO
- slow connection to Kafka

## Low Throughput - ksqlDB Apps

### Potential factors:

- Same as for Kafka Streams apps, plus:
- Too many queries
- Growing consumer lag
- Slow network
- Kafka cluster underpowered

## Troubleshooting ksqlDB Queries - No data

- No data in the source topic
- No new data arriving in the topic
- ksqlDB consuming from a later offset than for which there is data
- Wrong filter/predicate
- Deserialization errors



## What's happening under the covers? (1)

```
ksql> DESCRIBE EXTENDED GOOD_RATINGS;  
[...]  
Local runtime statistics  
-----  
messages-per-sec: 1.10 total-messages: 2898 last-message: 9/17/18 1:48:47 PM UTC  
failed-messages: 0 failed-messages-per-sec: 0 last-failed: n/a  
(Statistics of the local ksqlDB server interaction with the Kafka topic GOOD_RATINGS)  
ksql>
```

```
ksql> SHOW QUERIES;
```

Query ID	Kafka Topic	Query String
CSAS_GOOD_IOS_RATINGS_0	GOOD_IOS_RATINGS	CREATE STREAM GOOD_IOS_RATINGS AS SELECT * FROM...

## What's happening under the covers? (2)

```
ksql> EXPLAIN CSAS_GOOD_IOS_RATINGS_0;
```

```
[...]
```

Execution plan

```
-----
> [ SINK ] Schema: [ROWTIME : BIGINT, ROWKEY : VARCHAR, RATING_ID : BIGINT, USER_ID : BIGINT, STARS : INT, ROUTE_ID : BIGINT, RATING_TIME : BIGINT, CHANNEL : VARCHAR, MESSAGE : VARCHAR].
    > [ PROJECT ] Schema: [ROWTIME : BIGINT, ROWKEY : VARCHAR, RATING_ID : BIGINT, USER_ID : BIGINT, STARS : INT, ROUTE_ID : BIGINT, RATING_TIME : BIGINT, CHANNEL : VARCHAR, MESSAGE : VARCHAR].
        > [ FILTER ] Schema: [RATINGS.ROWTIME : BIGINT, RATINGS.ROWKEY : VARCHAR, RATINGS.RATING_ID : BIGINT, RATINGS.USER_ID : BIGINT, RATINGS.STARS : INT, RATINGS.ROUTE_ID : BIGINT, RATINGS.RATING_TIME : BIGINT, RATINGS.CHANNEL : VARCHAR, RATINGS.MESSAGE : VARCHAR].
            > [ SOURCE ] Schema: [RATINGS.ROWTIME : BIGINT, RATINGS.ROWKEY : VARCHAR, RATINGS.RATING_ID : BIGINT, RATINGS.USER_ID : BIGINT, RATINGS.STARS : INT, RATINGS.ROUTE_ID : BIGINT, RATINGS.RATING_TIME : BIGINT, RATINGS.CHANNEL : VARCHAR, RATINGS.MESSAGE : VARCHAR].
```

# ksqldb Troubleshooting with JMX

- Use `jconsole` or `jmxterm`
- Use **Prometheus** and **Grafana**



## ksqldb Health Checks - REST Endpoint

```
$ http://ksqldb-server:8088/info
* Trying 127.0.0.1...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8088 (#0)
> GET /info HTTP/1.1
> Host: localhost:8088
> User-Agent: curl/7.54.0
> Accept: /
>
< HTTP/1.1 200 OK
< Date: Fri, 23 Nov 2018 11:44:00 GMT
< Content-Type: application/vnd.ksql.v1+json
< Transfer-Encoding: chunked
< Server: Jetty(9.4.11.v20180605)
<
* Connection #0 to host localhost left intact
{"KsqlServerInfo":{"version":"5.1.2","kafkaClusterId":"8MvZ8QmPRYeluBTY0izk_Q","ksqlServiceId":"

```

# Troubleshooting RocksDB

Issue	Solution options
RocksDB performance appears low	<ul style="list-style-type: none"><li>• Use <b>SSDs</b></li><li>• Add <b>Cores</b></li></ul>
RocksDB's file sizes appear larger than expected	<ul style="list-style-type: none"><li>• Use Linux <b>du</b> tool</li><li>• Check for <b>Write Amplification</b></li></ul>
Application memory utilization seems high	<ul style="list-style-type: none"><li>• Mind <b>number of stores</b> in topology</li><li>• Lower RocksDB <b>memory usage</b></li></ul>



### Question:

Why does a stateful Kafka Streams application need much more memory than a stateless application working on the same input topic?

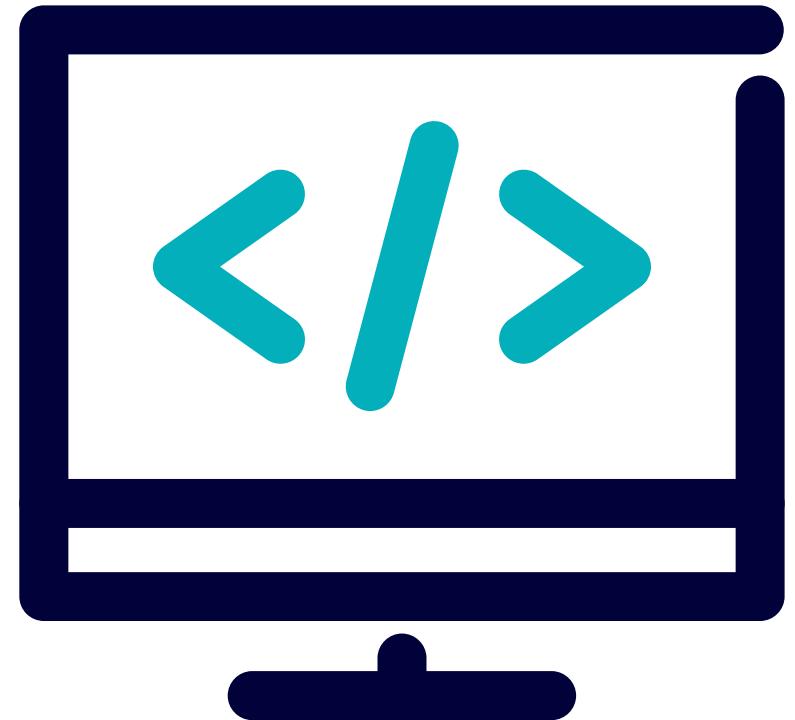
## Further Reading

- Elastic Scaling of Your Application: <https://bit.ly/2I1hOVA>
- Memory Management: <https://bit.ly/2M1phXa>
- KSQL capacity planning: <https://bit.ly/2x6KnAC>
- Troubleshooting KSQL – Part 1: Why Isn't My KSQL Query Returning Data?:  
<https://www.confluent.io/blog/troubleshooting-ksql-part-1>
- Troubleshooting KSQL – Part 2: What's Happening Under the Covers?:  
<https://www.confluent.io/blog/troubleshooting-ksql-part-2>
- Data Wrangling with Apache Kafka and KSQL:  
<https://www.confluent.io/blog/data-wrangling-apache-kafka-ksql>

# Lab: Troubleshooting Kafka Streams & ksqlDB Apps

Please work on **Lab 12a: Troubleshooting Kafka Streams & ksqlDB Apps**

Refer to the Exercise Guide



## b: Tuning Kafka Streams & ksqlDB Apps

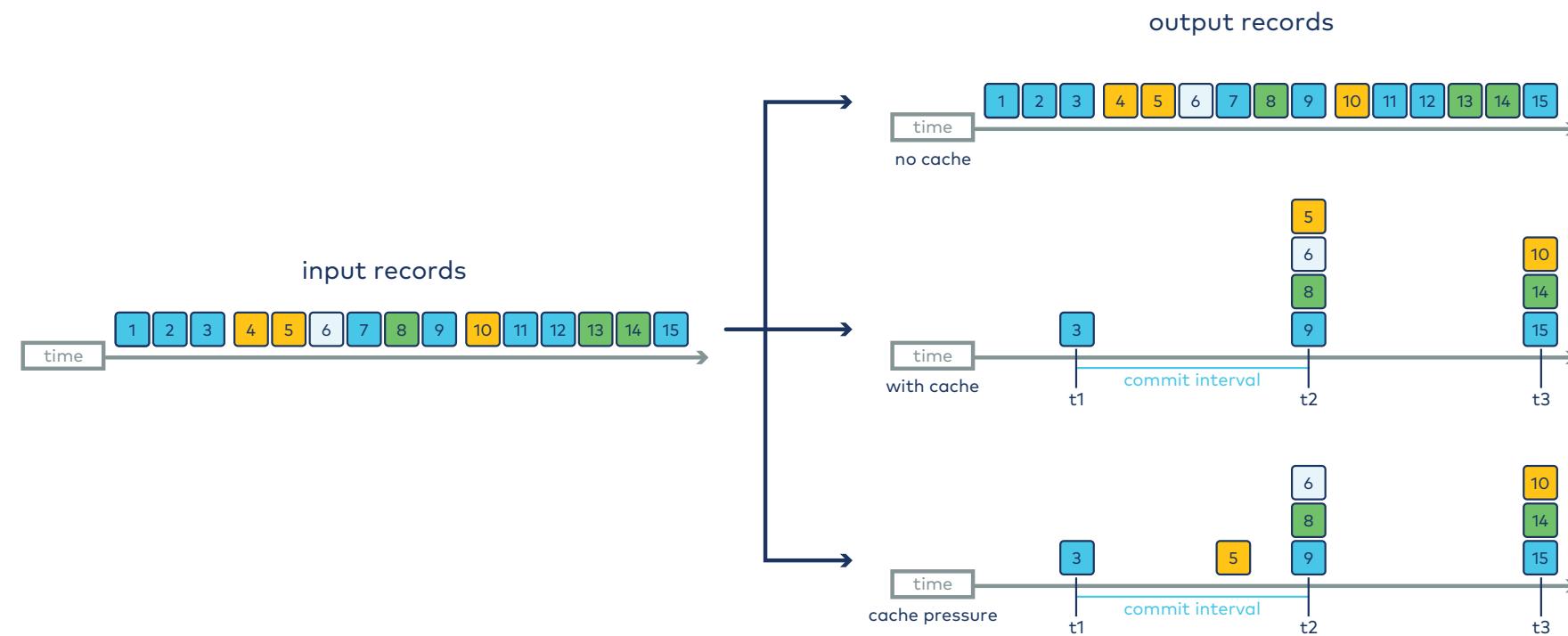
### Description

Streams apps scaling, memory management, fault tolerance. Best practices.

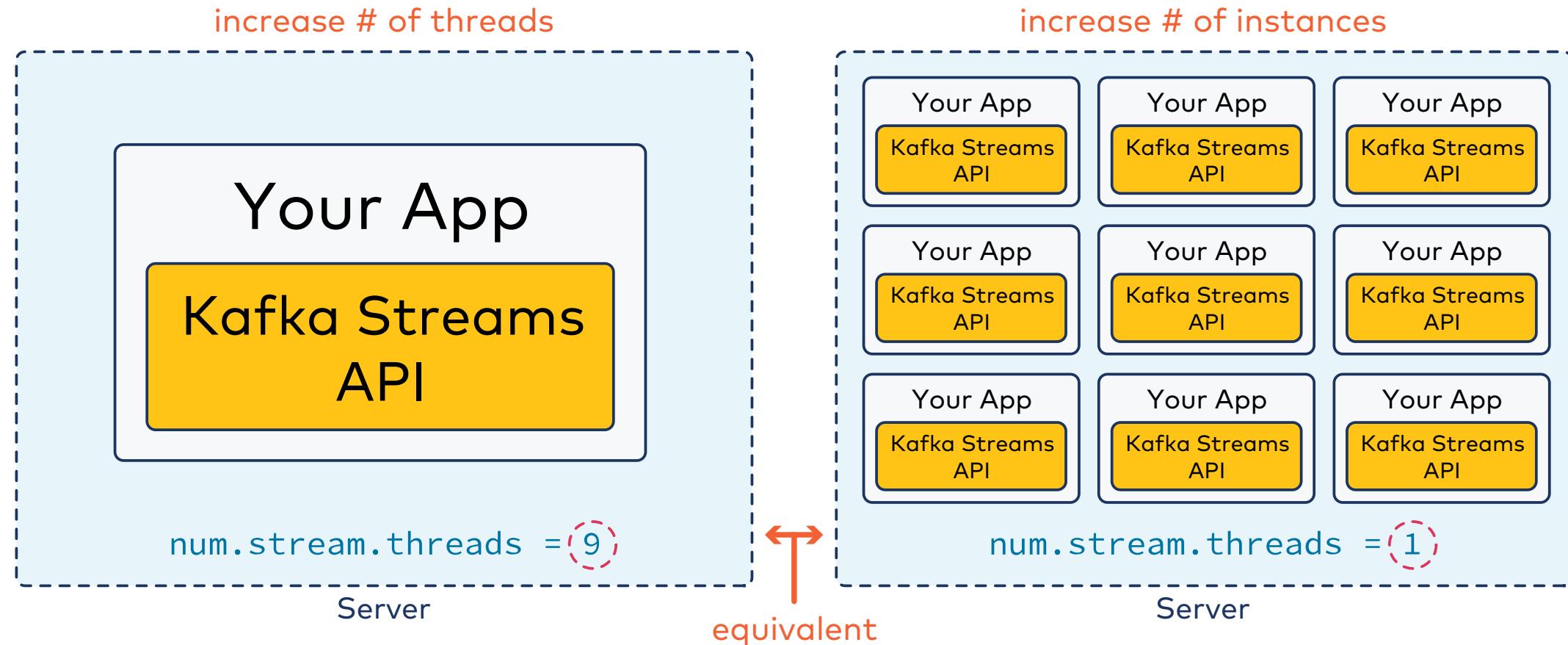
## How many Application Instances?

- Number of instances  $\leq$  number of topic partitions
- Distribute & balance data (topics)
- Distribute processing workload

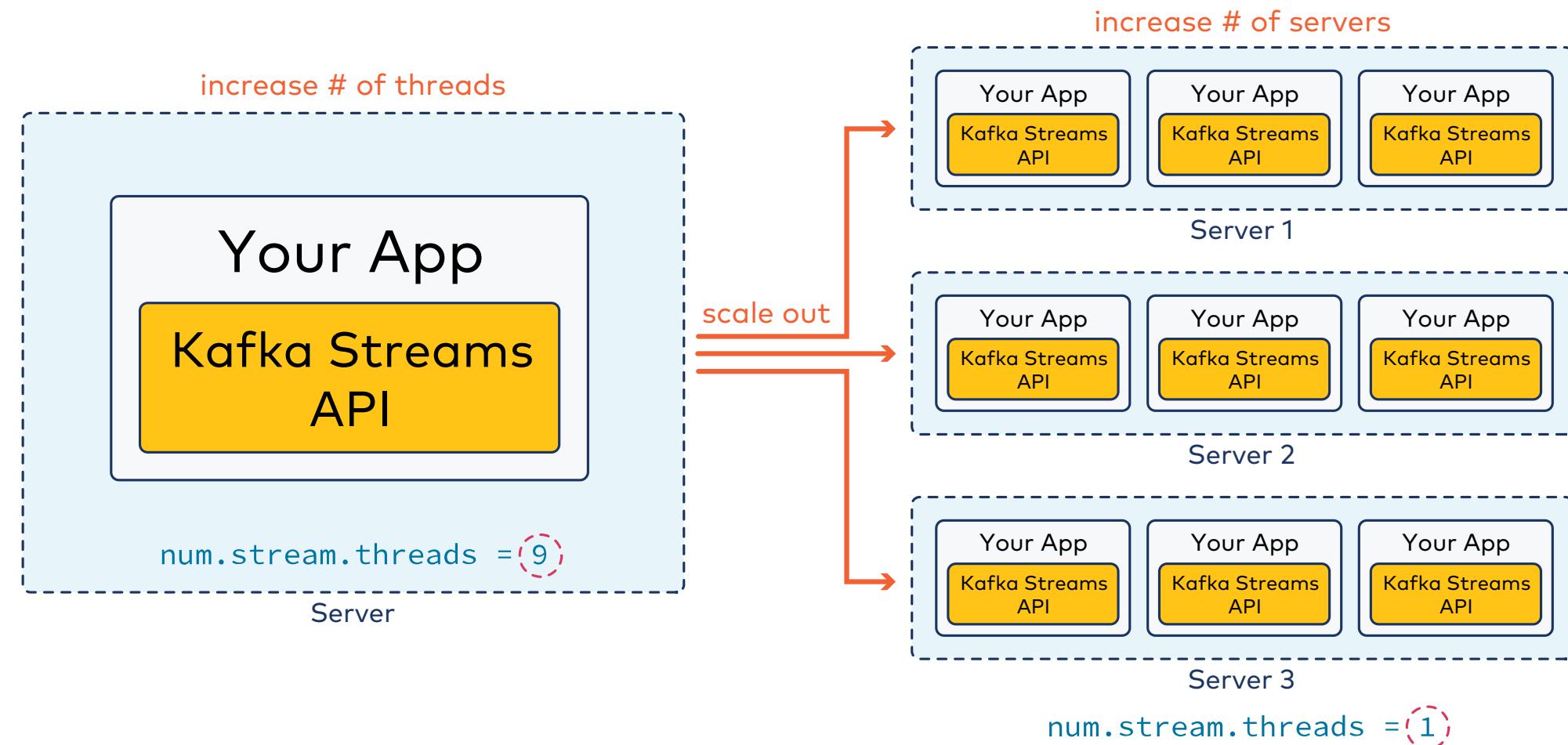
# Memory Management



# Task Placement - Scale Up



# Task Placement - Scale Out



## Stateless versus Stateful

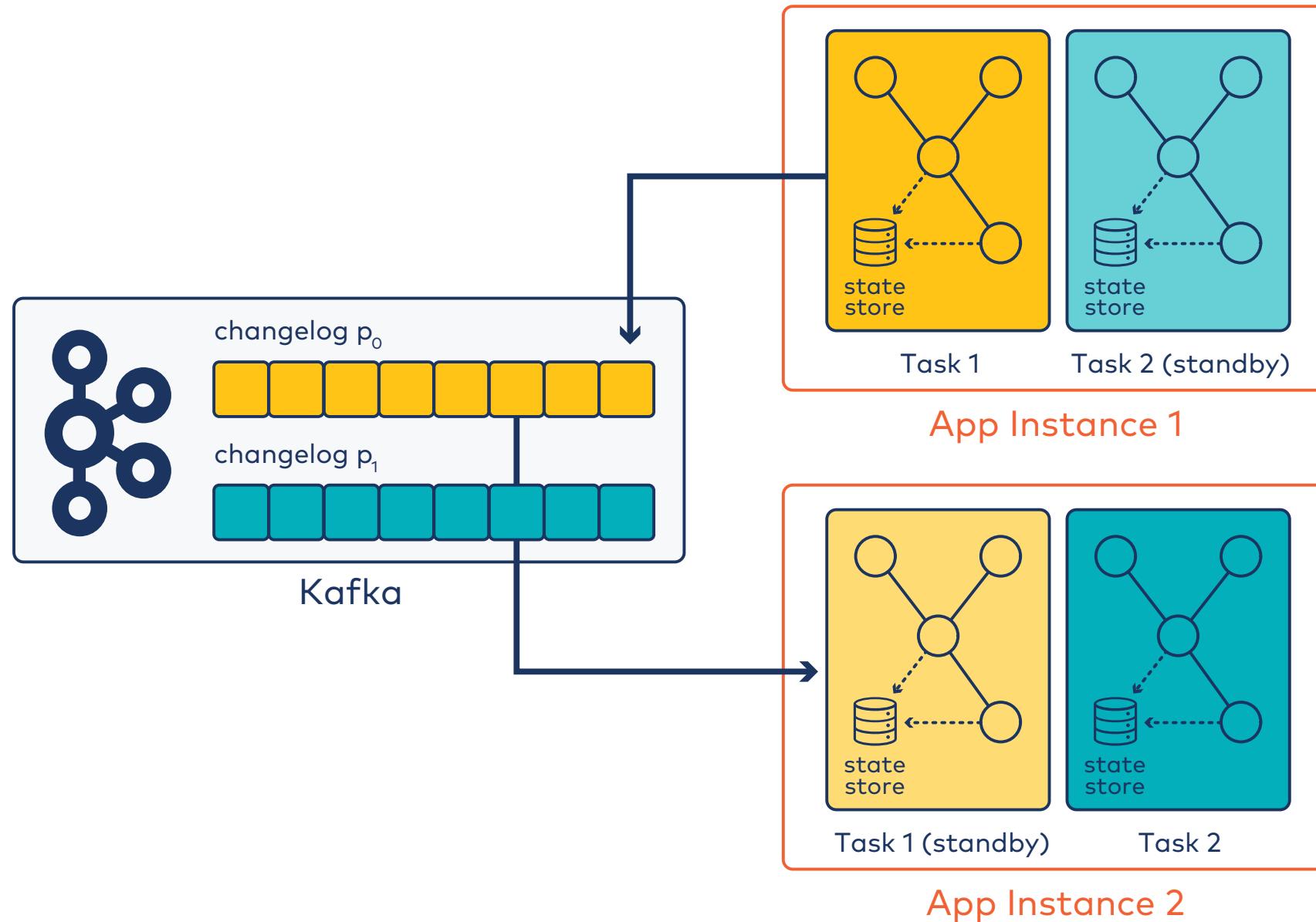
### Stateless Applications:

- CPU & Network are key

### Stateful Applications:

- **Memory:** High Performance
- **Local Disks:** Fast Queries
- **Kafka:** Fault Tolerance
- **Standby Replicas:** High Availability

# Standby Replicas



## Standby Replicas

- Kafka Streams: `num.standby.replicas=1`
- ksqlDB Server: `ksql.streams.num.standby.replicas=1`



`n` **standby replicas** require `n+1` Kafka Streams instances or ksqlDB servers

## ksqlDB Best Practices & Patterns

- Test Queries before moving to Prod
- Select the ksqlDB server mode based upon your requirements
  - Interactive mode
  - Headless mode
- Avoid big multi-purpose cluster
- Create ksqlDB cluster per App or per Team

## ksqldb Tuning

- Monitor **Consumer lag**
- Decrease threads for cheap queries
- Add resources to increase throughput
- Add standby instances for faster fail-over

# Recommended ksqlDB Production Settings

```
ksql.streams.producer.delivery.timeout.ms=2147483647      # Integer.MAX_VALUE  
ksql.streams.producer.max.block.ms=9223372036854775807    # Long.MAX_VALUE  
ksql.streams.replication.factor=3  
ksql.streams.producer.acks=all  
ksql.streams.topic.min.insync.replicas=2  
ksql.streams.state.dir=/some/non-temporary-storage-path/  
ksql.streams.num.standby.replicas=1
```



### Question:

What are the pros and cons of scaling up versus scaling out a Kafka Streams application?

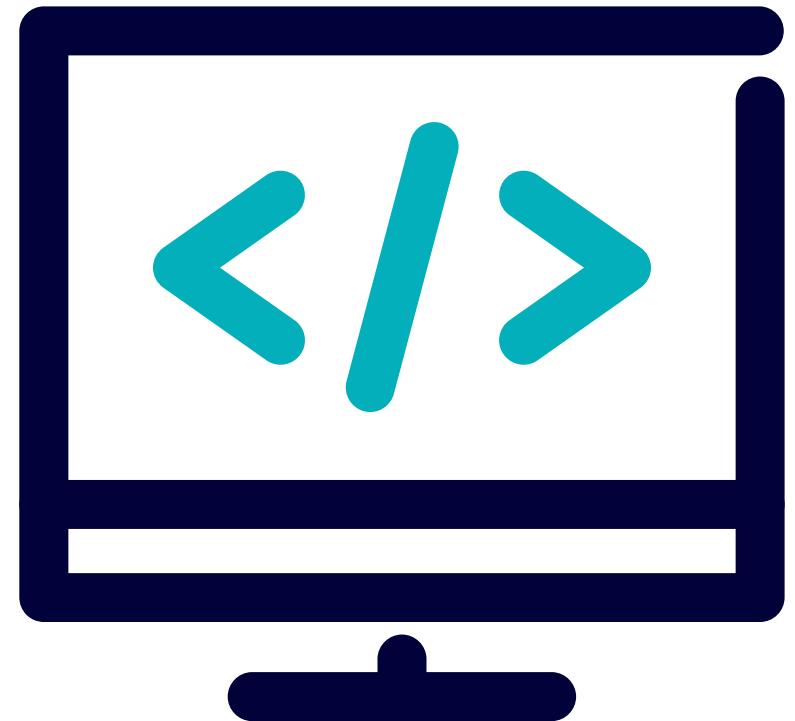
## Further Reading

- Kafka Streams Architecture: <https://kafka.apache.org/21/documentationstreams/architecture>
- Achieving High Availability With Stateful Kafka Streams Applications: <https://tech.transferwise.com/achieving-high-availability-with-kafka-streams/>
- Troubleshooting KSQL – Part 1: Why Isn't My KSQL Query Returning Data?: <https://www.confluent.io/blog/troubleshooting-ksql-part-1>
- Capacity Planning & Sizing: <https://docs.confluent.io/currentstreams/sizing.html>
- KSQL Capacity Planning: <https://docs.confluent.io/currentksql/docs/capacity-planning.html>

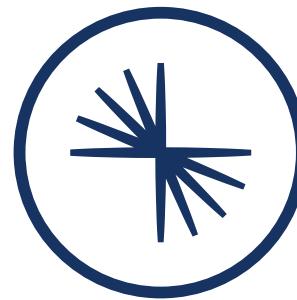
# Lab: Tuning Kafka Streams & ksqlDB Apps

Please work on **Lab 12b: Tuning Kafka Streams & ksqlDB Apps**

Refer to the Exercise Guide



# 13: Troubleshooting & Tuning Kafka Connect



CONFLUENT  
**Global Education**

# Module Overview



This module contains two lessons:

1. Troubleshooting Connect
2. Tuning Connect

## a: Troubleshooting Kafka Connect

### Description

Connect error handling framework. Other Connect issues.

## Validating Connector Configuration

```
PUT /connector-plugins/hdfs-sink-connector/config/validate HTTP/1.1
```

```
Host: connect.example.com
```

```
Accept: application/json
```

```
{  
    "connector.class": "io.confluent.connect.hdfs.HdfsSinkConnector",  
    "tasks.max": "10",  
    "topics": "test-topic",  
    "hdfs.url": "hdfs://fakehost:9000",  
    "hadoop.conf.dir": "/opt/hadoop/conf",  
    "hadoop.home": "/opt/hadoop",  
    "flush.size": "100",  
    "rotate.interval.ms": "1000"  
}
```

Look out for errors:

```
jq '.configs[]|select (.value.errors[]!=null) | .value'
```

# Fail Fast Scenario

## Why?

- **Poisoned messages**  
i.e., cannot be processed
- Source/target system unavailable

## How?

### Configuration settings:

```
# disable retries on failure (default 0)
errors.retry.timeout=0

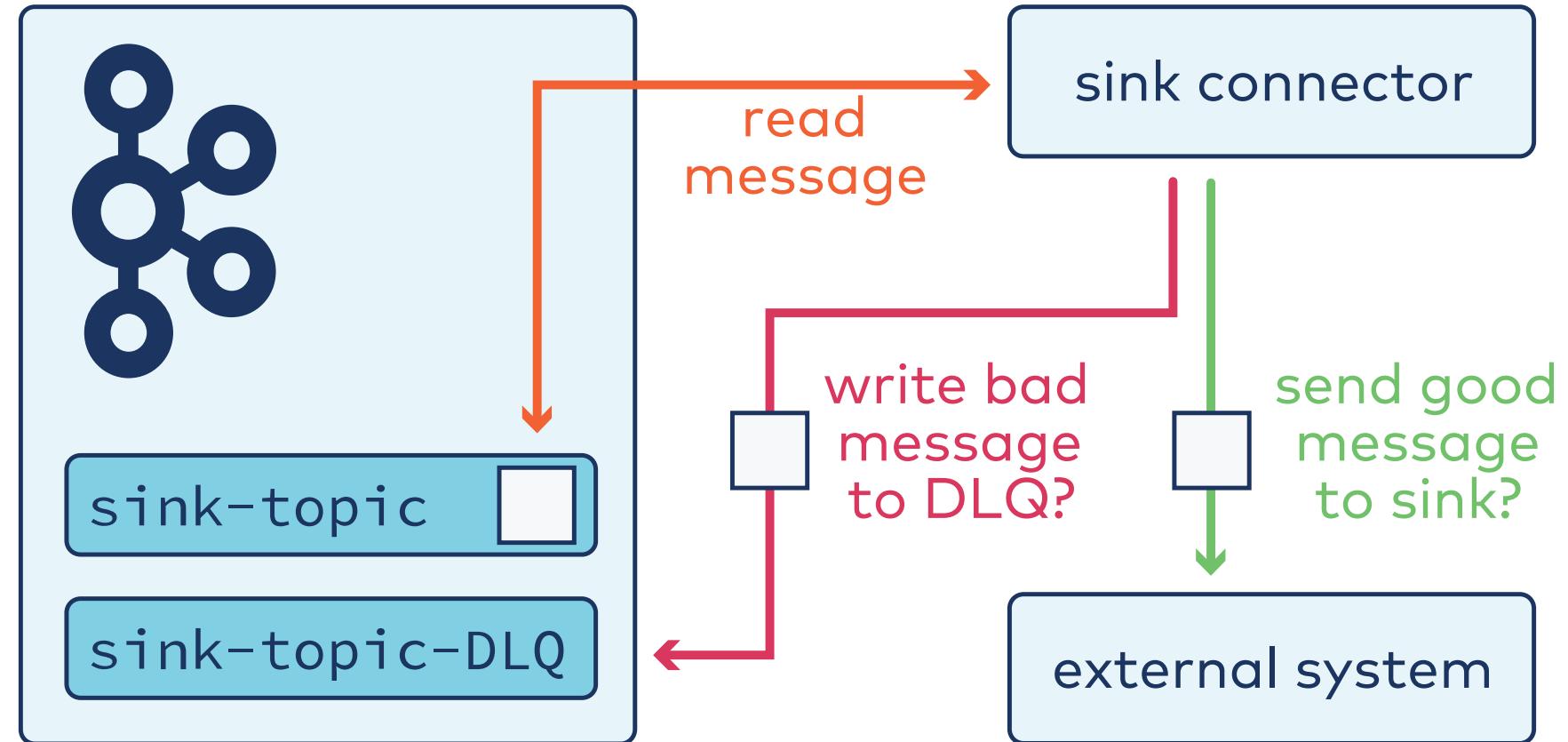
# do not log the error and their contexts
errors.log.enable=false

# do not record errors in a
# dead letter queue topic
errors.deadletterqueue.topic.name=""

# Fail on first error
errors.tolerance=none
```

# Dead Letter Queue

- **Problem:** Writing message to external system fails
- **Solution:**
  - Rather than giving up, produce this message to a special Kafka topic  
→ Called the **dead letter queue (DLQ)**
  - Can inspect those messages separately and decide how to handle



The DLQ is for sink connectors only.

## Error Management Options

To configure error management, configure the **connector** settings:

Name	Default	Source Connectors?	Sink Connectors?
errors.retry.timeout	0	yes	yes
errors.retry.delay.max.ms	1 min	yes	yes
errors.tolerance	-	yes	yes
errors.deadletterqueue.topic.name	""	no	yes
errors.log.enable	false	yes	yes
errors.log.include.messages	false	yes	yes

## Recommended Error Management Config

Here's an example of configuring error management:

```
# retry for at most 10 minutes waiting up
# to 30 seconds between consecutive failures
errors.retry.timeout=600000
errors.retry.delay.max.ms=30000

# log error context along with application logs
# but do not include configs and messages
errors.log.enable=true
errors.log.include.messages=false

# produce error context into the Kafka topic
errors.deadletterqueue.topic.name=my-connector-errors

# Tolerate all errors.
errors.tolerance=all
```

## Monitoring for Errors

Metric/Attribute Name	Description
<code>total-record-failures</code>	Total number of failures seen by this task.
<code>total-record-errors</code>	Total number of errors seen by this task.
<code>total-records-skipped</code>	Total number of records skipped by this task.
<code>total-retries</code>	Total number of retries made by this task.
<code>total-errors-logged</code>	The number of messages that was logged into either the dead letter queue or with Log4j.
<code>deadletterqueue-produce-requests</code>	Number of produce requests to the dead letter queue.
<code>deadletterqueue-produce-failures</code>	Number of records which failed to produce correctly to the dead letter queue.
<code>last-error-timestamp</code>	The timestamp when the last error occurred in this task.

The MBean to look for is:

```
kafka.connect:type=task-error-metrics,connector=([-.\w]+),task=([-.\w]+)
```

## Monitoring Workers and Hosts

- Monitor Workers with REST interface
- Monitor Hosts where Workers run
  - CPU utilization
  - Garbage collection pause duration
  - Heap usage
  - Physical memory usage

## Add Connector Context to Worker Logs

- Include `%X{connector.context}` in the log4j layout
  - Adds connector-specific and task-specific information to the log message
  - Makes it easier to identify log messages that apply to a specific connector
- To add this parameter, update the log layout configuration as follows:

```
#log4j.appender.stdout.layout.ConversionPattern=[%d] %p %X{connector.context}%m (%c:%L)%n
```

## Dynamically Adjust Connect Worker Log Levels

The Connect Worker `/admin/loggers` endpoint supports the following operations:

- Get a list of all named loggers
- Get the log level of a specific logger
- Set the log level of a specific logger

Log level modifications:

- will not be persisted across worker restarts
- will only affect the worker whose endpoint received this REST request

## Dynamically Adjust Connect Log Levels - Examples

- Get a list of all named loggers

```
curl -s http://connect:8083/admin/loggers/ | jq
{
  "root": {
    "level": "INFO"
  }
}
```

- Set the log level of a specific logger

```
curl -s -X PUT -H "Content-Type:application/json" \
>     http://connect:8083/admin/loggers/org.apache.kafka.connect.runtime.WorkerSourceTask \
>     -d '{"level": "TRACE"}' | jq '.'
[
  "org.apache.kafka.connect.runtime.WorkerSourceTask"
]
```



### Question:

How can you best discover that your Kafka Connect connectors are having an issue?

## Further Reading

- KIP-298: Error handling in Connect:  
<https://cnfl.io/kip-298>
- Connect Concepts:  
<https://docs.confluent.io/current/connect/concepts.html>
- Connect Worker Configs:  
<https://docs.confluent.io/current/connect/references/allconfigs.html>
- Install Connector Manually:  
<https://docs.confluent.io/current/connect/managing/install.html#install-connector-manually>

## b: Tuning Kafka Connect

### Description

Connect best practices.

## Monitoring Connect

- Monitoring helps in sizing & troubleshooting
- Workers have embedded producer and/or consumer

Metrics	MBean Name
Connector Metrics	<code>kafka.connect:type=connector-metrics,connector=&lt;connector-ID&gt;</code>
Task Metrics (Common, Source & Sink)	<code>kafka.connect:type=task-metrics,connector=&lt;connector-ID&gt;,task=&lt;task-ID&gt;</code> <code>kafka.connect:type=source-task-metrics,connector=&lt;connector-ID&gt;,task=&lt;task-ID&gt;</code> <code>kafka.connect:type=sink-task-metrics,connector=&lt;connector-ID&gt;,task=&lt;task-ID&gt;</code>
Worker Metrics	<code>kafka.connect:type=connect-worker-metrics</code>
Worker Rebalance Metrics	<code>kafka.connect:type=connect-worker-rebalance-metrics</code>

## Filter and Transform Data

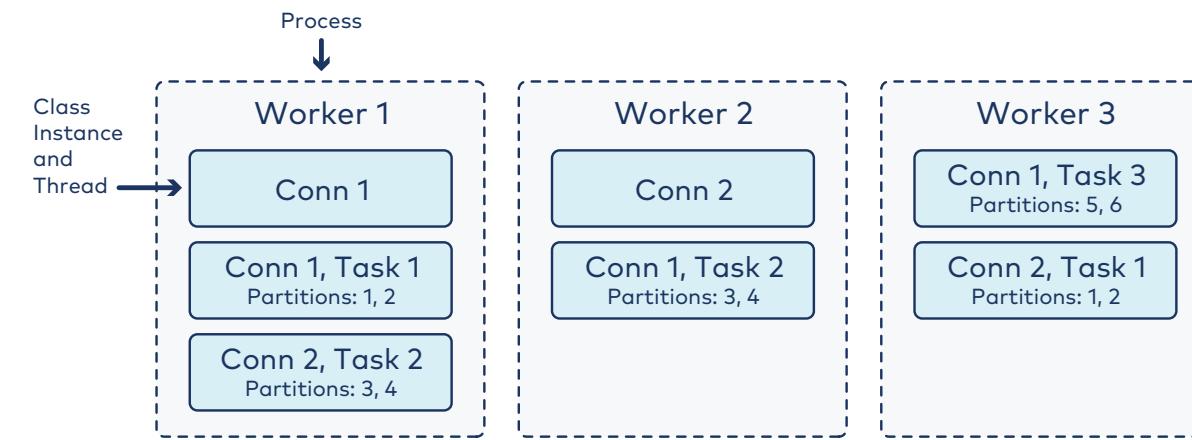
- Create/extract key from data field
- Add metadata to Kafka message
- Field Masking & whitelist/blacklist fields
- Route messages with
  - Regular expressions
  - Timestamp



Can chain many transformations!

# Scale your Connect cluster - Distributed Workers

- Scalability & fault tolerance
- All workers have same `group.id`
- Workers coordinate to distribute connectors and tasks across all worker instances
- Workload is auto-rebalanced upon failure





### Question:

You want to import a huge table from your MySQL DB to Kafka. The table has about 2 dozen fields and you only really need about 5 of them. How do you proceed?

## Further Reading

- The Simplest Useful Kafka Connect Data Pipeline In The World ... or thereabouts—Part 1-3:

<https://cnfl.io/connect-pipeline-1>

<https://cnfl.io/connect-pipeline-2>

<https://cnfl.io/connect-pipeline-3>

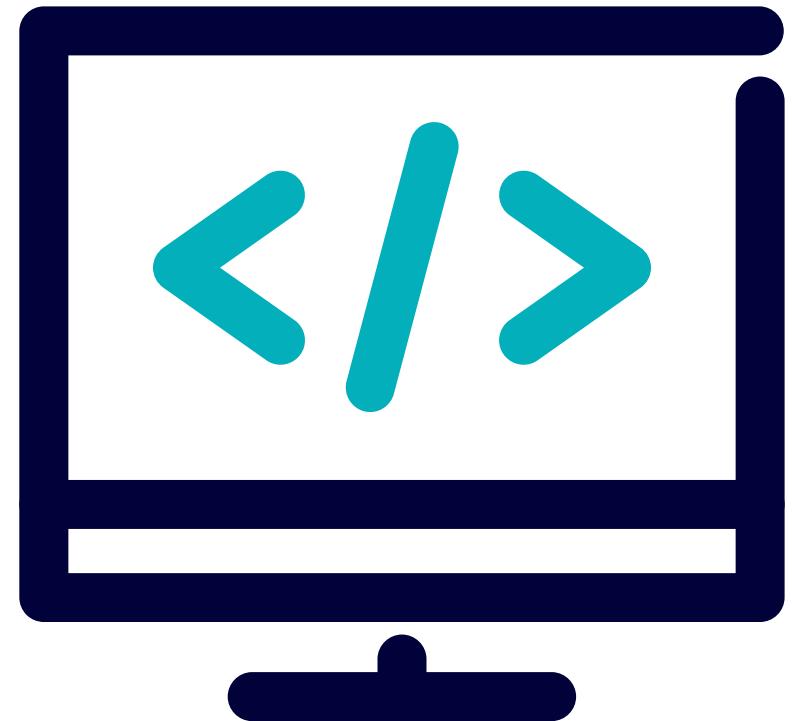
- Connect transformations:

<https://docs.confluent.io/current/connect/transforms/index.html>

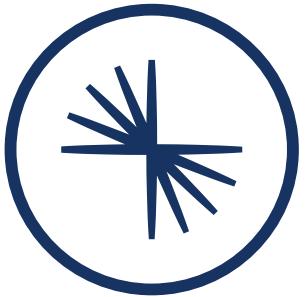
# Lab: Tuning Kafka Connect

Please work on **Lab 23a: Tuning Kafka Connect**

Refer to the Exercise Guide



# Conclusion



CONFLUENT  
**Global Education**

# Course Contents



Now that you have completed this course, you should have the skills to:

- Formulate the Apache Kafka® Confluent Platform specific needs of your company
- Monitor all essential aspects of your Confluent Platform
- Tune the Confluent Platform according to your specific needs
- Provide first level production support for your Confluent Platform

## Other Confluent Training Courses

- Confluent Developer Skills for Building Apache Kafka®
- Confluent Stream Processing Using Apache Kafka® Streams & ksqlDB
- Apache Kafka® Administration By Confluent
- Managing Data in Motion with Confluent Cloud



For more details, see <https://confluent.io/training>

# Confluent Certified Developer for Apache Kafka

**Duration:** 90 minutes

**Qualifications:** Solid understanding of Apache Kafka and Confluent products, and 6-to-9 months hands-on experience

**Availability:** Live, online, 24-hours a day!

**Cost:** \$150

**Register online:** [www.confluent.io/certification](http://www.confluent.io/certification)



# Confluent Certified Administrator for Apache Kafka

**Duration:** 90 minutes

**Qualifications:** Solid work foundation in Confluent products and 6-to-9 months hands-on experience

**Availability:** Live, online, 24-hours per day!

**Cost:** \$150

**Register online:** [www.confluent.io/certification](http://www.confluent.io/certification)



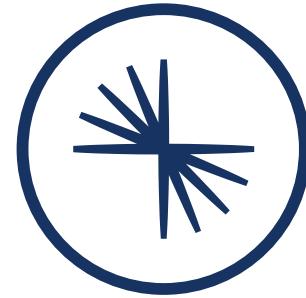
# We Appreciate Your Feedback!



Please complete the course survey now.

# Thank You!

## Appendix: Additional Problems



CONFLUENT  
**Global Education**

## Overview

This section contains a few additional problems to reinforce concepts in this course. These problems were originally written as warm-up problems for instructor-led training for this course. Your instructor may or may not choose to incorporate some or all of these problems in class; you may find them to provide additional enrichment in any case. Some other problems originally created as warm-up problems have been adapted into activities in the content of this version of this course.

## Problem A: Reviewing Message Sending and Broker Arrival

Suppose you have a new producer. Suppose, further, `linger.ms` has been increased from its default to 5 minutes.

1. You send a message and it gets partitioned to partition  $p_5$ . What all happens from the moment that assignment is determined until the message makes it to the page cache of the broker containing  $p_5$ ?
2. Suppose, now, an overall total of 31.9 MB of messages has been produced by this producer and assigned to various partitions. No broker yet knows anything about any of these messages. You send a message of size 0.5 MB. Does it fail? Explain why or why not.

## Problem B: Replication Review

Suppose you have 5 brokers. We will concern ourselves with a partition  $p_1$ .

1. Suppose replication factor for  $p_1$  is 3. Illustrate a possible scenario.
2. Suppose replication factor for  $p_1$  is instead 7. Illustrate a possible scenario.
3. Finally, regardless of anything in the previous parts, we have the following situation for our replicas:

$p_{1,L}$  contains messages  $m_0$ ,  $m_1$ , and  $m_2$  and the last write went to this replica first

$p_{1,F0}$  contains messages  $m_0$  and  $m_1$

$p_{1,F1}$  contains messages  $m_0$ ,  $m_1$ , and  $m_2$

Suppose the broker containing  $p_{1,L}$  fails. What will happen? Explain.

## Problem C: Consumer Offsets and Consumer Lag

Suppose partition  $p_0$  has messages at offsets 0 through 20 and consumer  $c_3$  is assigned to  $p_0$ . Say you are monitoring this consumer using CCC or the CLI.

- a. What are valid values of  $c_3$ 's consumer offset in  $p_0$ ?
- b. What would be reported as the "log end offset" for this consumer and partition?
- c. Suppose the offset is 12. What is the consumer lag?
- d. Where would you go in CCC to find lag? What would you look for as evidence that lag is increasing?
- e. Suppose your consumer application is reading messages containing mobile food orders and printing order slips for a restaurant's kitchen. How would you interpret your answer to (c)? What would we ideally want lag to be in this case?
- f. Give a different use case where we wouldn't care about lag so much?

## Problem D: Monitoring Disk Usage

Say you're doing the right thing and monitoring broker disk usage.

- a. You notice a disk on broker  $b_{101}$  is at 95% capacity. You decide that might be dangerous and decide to do something about that. Is this good? What could be problematic?
- b. How can you prevent such issues? What's Confluent's recommended best practice?

## Problem E: Assessing Discrepancies in Settings

- A colleague insists he changed how long a log segment could be the active log segment before it rolls to a max of 2 hours.
- But another colleague is reporting that she has seen some log segments for topic  $t_7$ , partition  $p_{12}$  are on broker 103 have been the active log segment with timestamps 4 and 5 hours in the past.

Another colleague wants answers and explanations. What do you tell them?

# Problem F: Troubleshooting Producers and Consumers

## Question 1

How do you make sure that all messages produced are actually received by Kafka?  
(Hint: think about some important settings to inspect/set.)

## Question 2

You realized that one of your consumer instances in consumer group `Foo` processed the data from partitions 1, 4, and 7 of `topic-1` incorrectly. You realize this is unacceptable...

- a. Conceptually, what should you do and why?
- b. Can you tell how to do this with a command or code, either at a high level or specifically?

## Question 3

After noticing problems with consumption, you find out that some consumers get significantly more messages than others. This behavior is not temporary; there is a

consistent imbalance, even after rebalancing. Some consumers go long periods of time consuming zero messages.

- a. What could be some causes of this problem?
- b. What are some possible solutions?