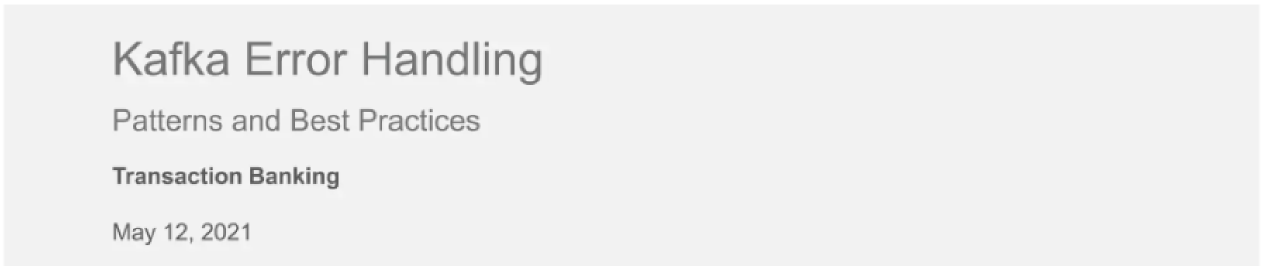


Kafka Error Handling

Patterns and Best Practices

Transaction Banking

May 12, 2021

Transaction Banking from Goldman Sachs is a high volume, latency sensitive digital banking platform offering. We have chosen an event driven architecture to build highly decoupled and independent microservices in a cloud native manner and are designed to meet the objectives of Security, Availability Latency and Scalability. Kafka was a natural choice – to decouple producers and consumers and to scale easily for high volume processing. However, there are certain aspects that require careful consideration – handling errors and partial failures, managing downtime of consumers, secure communication between brokers and producers / consumers. In this session, we will present the patterns and best practices that helped us build robust event driven applications. We will also present our solution approach that has been reused across multiple application domains. We hope that by sharing our experience, we can establish a reference implementation that application developers can benefit from.

Aruna Kalagnanam

Aruna Kalagnanam is a Tech Fellow in the Transaction Banking Group in Goldman Sachs, leading the development of Security solutions for the Cloud. Aruna has 20+ years of experience in developing distributed systems, application frameworks and APIs in firms like JP Morgan Chase, Walmart Labs and IBM.

Hemant Desale

Hemant Desale is a Vice President and Software Engineer in Transaction Banking Group in Goldman Sachs and enjoys working on distributed systems for banking and financial services.

Introduction

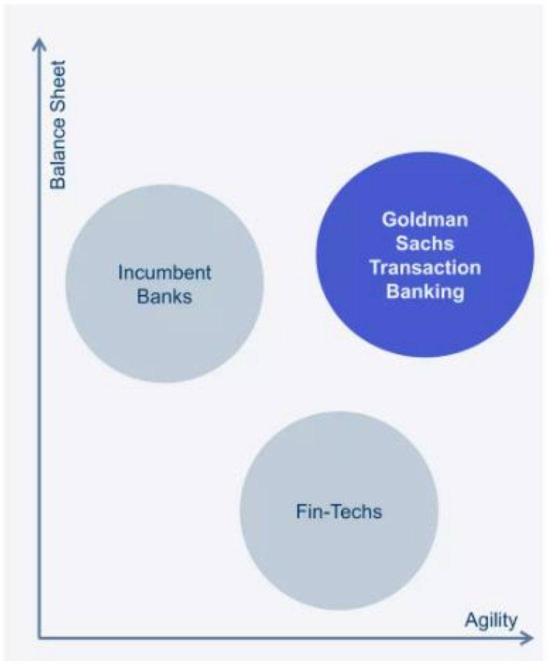
Transaction Banking

Transaction Banking Vision

Goldman Sachs has created a digital-first, built-from-scratch transaction bank. The firm is uniquely positioned to launch this product, leveraging over 150 years of financial and risk management expertise, yet unencumbered by legacy banking infrastructure.

GS Financial Cloud

- 24/7 Availability via Cloud-Based Banking Platform
- Robust Suite of Liquidity and Payment Solutions
- Faster and Reimagined Client Onboarding
- Data-Driven Actionable Insights and Routing Engines
- Integration with Leading Treasury Workstations

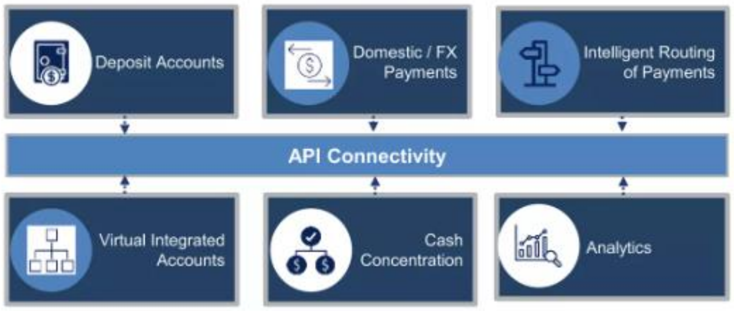


The platform

Transaction Banking

Transaction Banking Solution Overview

Goldman Sachs Transaction Banking (TxB) Product Development is Agile, Client-Driven, and Fully Configurable



Deposit Accounts

Deposit options with varying yield and liquidity, backed by GS Bank USA (S&P:A+/A-1, Fitch: A+/F1, Moody's: A1/P-1*)

Domestic and International Payments

Centralized solution with real-time payment tracking and no hidden fees

Virtual Integrated Accounts

Full payment capabilities with clearing recognized account numbers

Cash Concentration

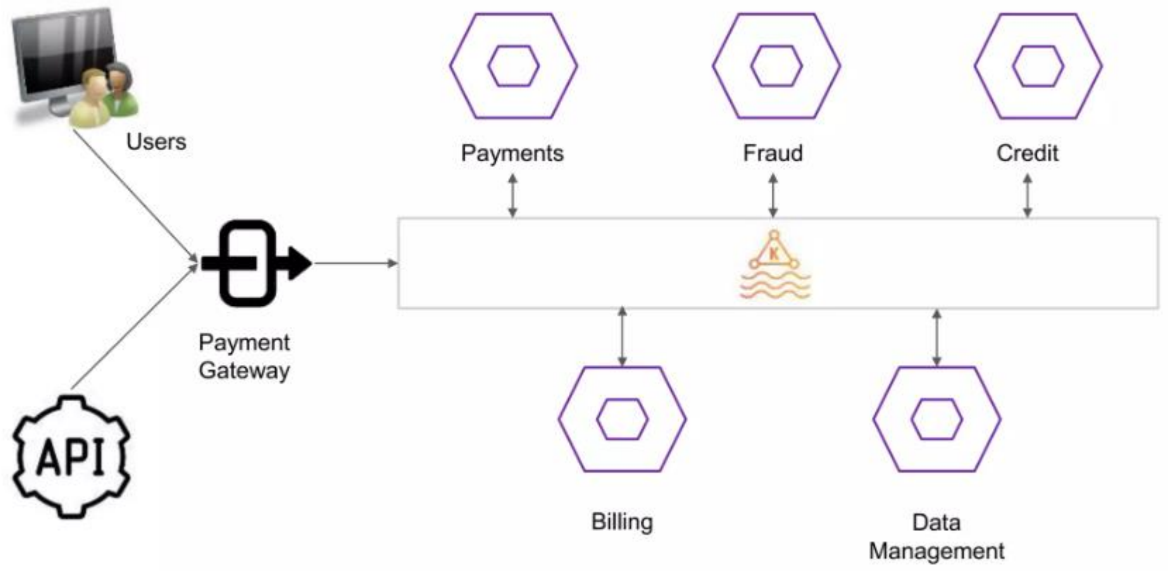
Intelligent, rule-base sweep for additional yield on excess cash

Analytics

Real-time multi-bank reporting available via UI, file, API and message

Event driven architecture

Kafka as the messaging system

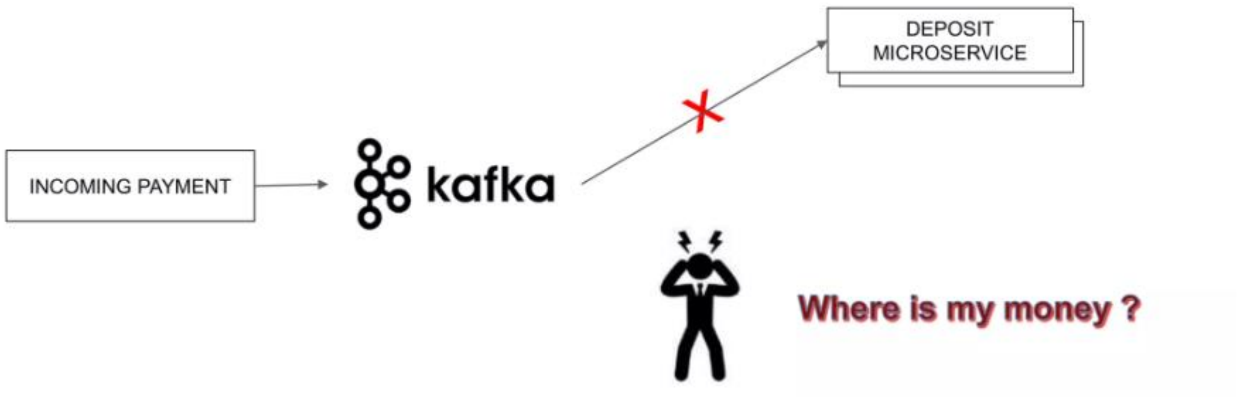


Error Handling

Patterns and Best Practices

Error handling

- ❑ Event driven microservices are decoupled, independent and scalable, however, introduce operational complexity due to message flow through discrete components
- ❑ Handling partial failures is challenging, but extremely important
- ❑ Best to consider error handling as part of design, this helps make the system more reliable and predictable



Transient errors

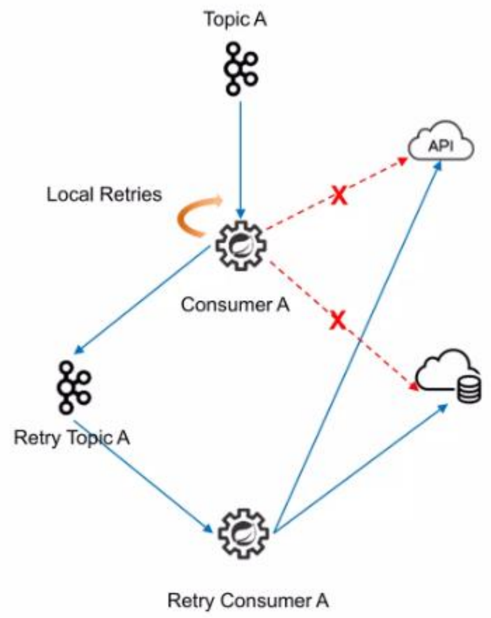
Retry-able errors

- ❑ Services should be prepared for handling transient errors
  - Network issues (they do occur!)
  - External API endpoints being down
  - Database failures

- ❑ Kafka consumers consume message off a topic, process it and hand it off to another service
- ❑ This maybe an external service
- ❑ If a request to a dependent service fails with a 503, how to handle it ?

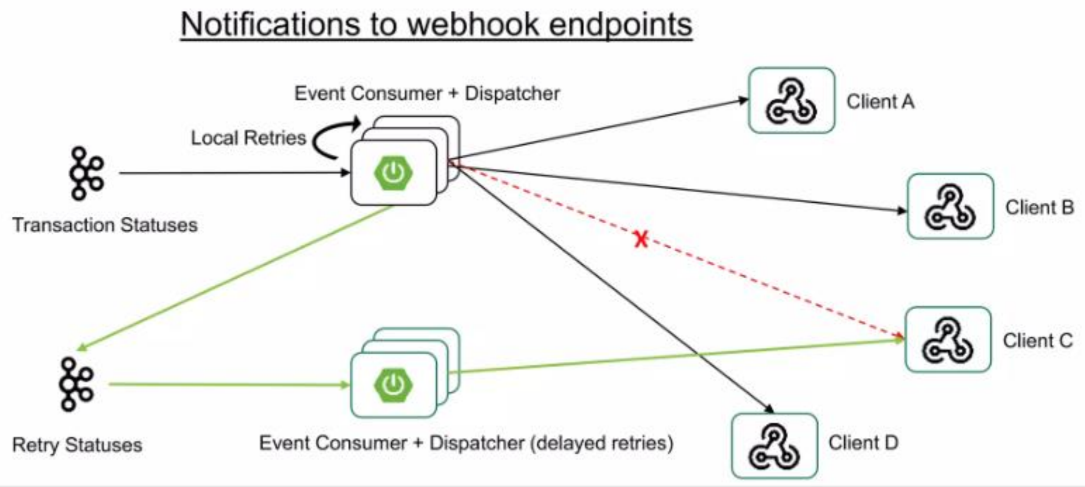
Handling transient errors

- ❑ Do not ignore the message and process the next one
- ❑ Implement a retry mechanism within the Kafka consumer
  - Retry after a fixed delay
  - Retry with an exponential back-off delay
- ❑ Setup retry topics for delayed processing
- ❑ If all retries are exhausted, configure a recovery callback mechanism
  - Logging the relevant information
  - Configuring a dead letter topic and alerting on the same
  - Tombstoning the information into a database



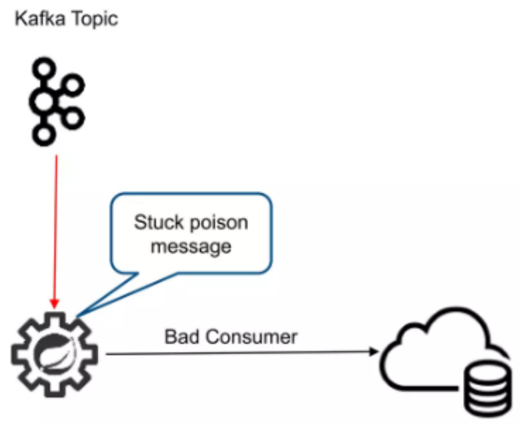
API, API, are you there ?

- ❑ Applications rely on 3rd party services for fulfilling transactions
- ❑ Or applications may have to push notifications to external endpoints, like webhooks
- ❑ Here is an example of how to handle downtime of such external dependencies in a Kafka driven event based system


Notifications to webhook endpoints
Event Consumer + Dispatcher
Local Retries
Transaction Statuses
Client A
Client B
Client C
Client D
Retry Statuses
Event Consumer + Dispatcher (delayed retries)

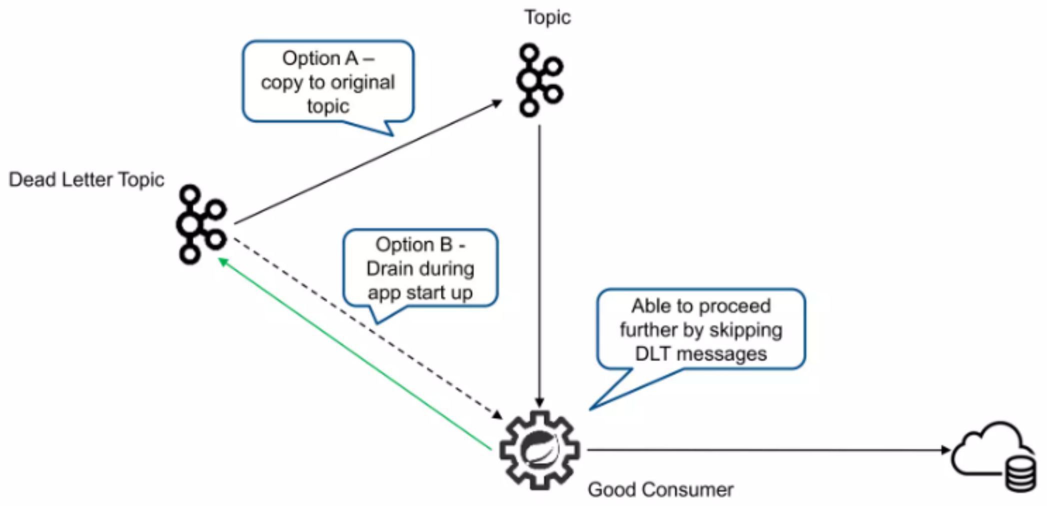
Non-Transient Errors
Bad Consumer

- ❑ Non-transient error : message fails with the same error no matter how many times retried
- ❑ Such errors usually need a fix, for example –
  - parsing errors
  - application bugs
  - schema compatibility issues

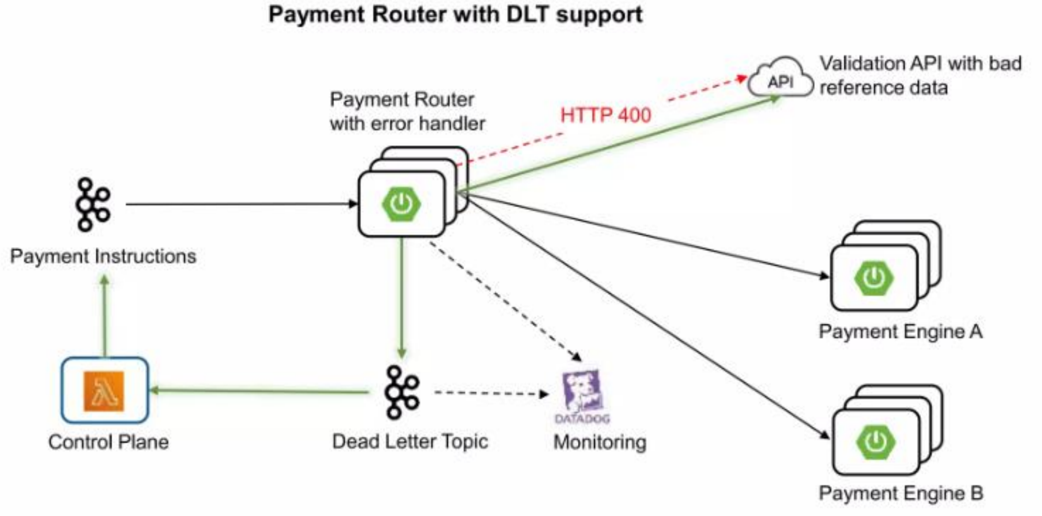


Non-Transient Errors
Good Consumer

- ❑ Define Non-retryable exception class hierarchy for such errors
- ❑ Push failed messages to Dead Letter Topics before skipping the message
- ❑ If message ordering needs to be maintained then fail all co-related messages


Topic
Option A – copy to original topic
Dead Letter Topic
Option B - Drain during app start up
Able to proceed further by skipping DLT messages
Good Consumer

Non-Transient Errors
Example



❑ Extended SeekToCurrentErrorHandler from spring-kafka to push messages to DLT

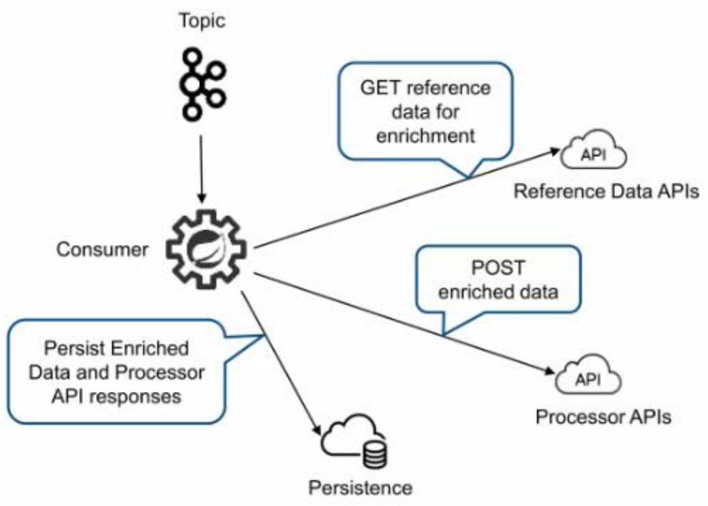
❑ Used Control Plane (internally uses AWS lambda) to copy messages back to original topic

❑ Used DataDog for DLT monitoring

Complex Message Processing
Error Handling Issues

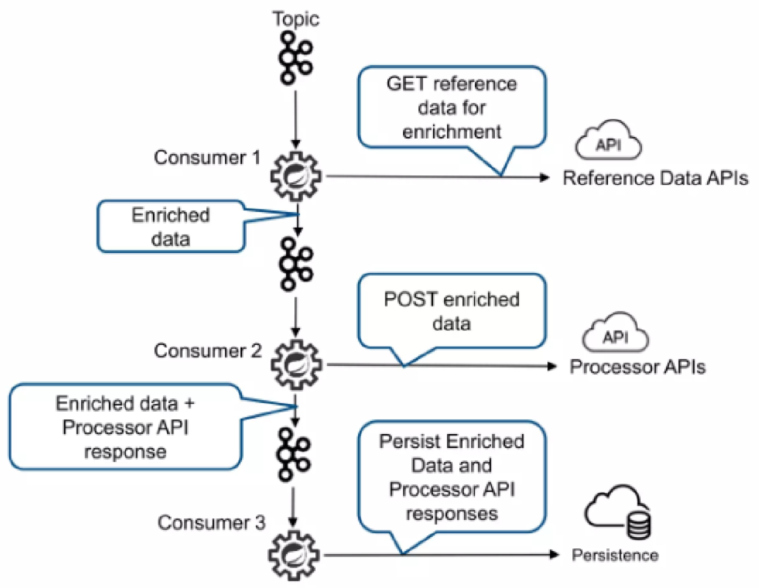
When message processing is ***complex and not atomic***

- It is not always possible to rollback everything
- Message replays become difficult because we don't know where to resume


Topic
GET reference data for enrichment
API
Reference Data APIs
Consumer
POST enriched data
Persist Enriched Data and Processor API responses
API
Processor APIs
Persistence

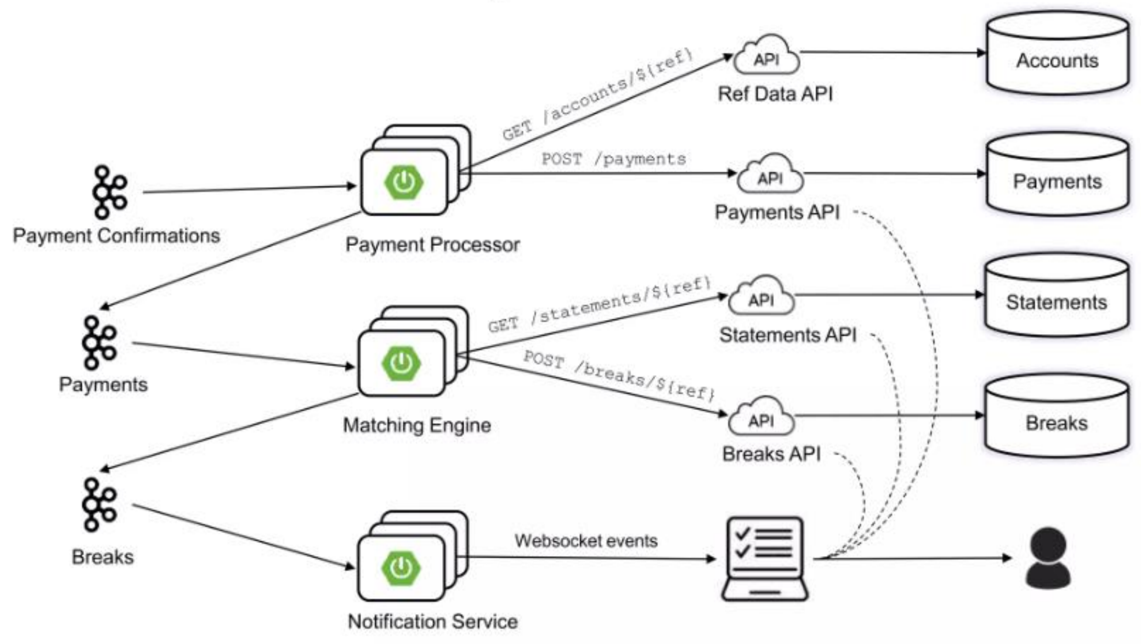
Complex Message Processing
Possible Solution

- ❑ Split message processing into small atomic operations and events
- ❑ Make operations idempotent so retries do not cause dupes


Topic
GET reference data for enrichment
API
Consumer 1
Reference Data APIs
Enriched data
POST enriched data
API
Consumer 2
Processor APIs
Enriched data + Processor API response
Persist Enriched Data and Processor API responses
Consumer 3
Persistence

Complex Message Processing
Example

Match Payments with Statement Items



Tips

From our experience

- ❑ If you are a Java shop like us then spring-kafka could be useful
  - Easy to extend to achieve your bespoke error handling requirements
  - Out of the box support for chaining transactions with other resources like DB

- ❑ Consistent set of configuration across multiple Kafka clients helps to achieve consistency in error handling.

- ❑ Some key configs
  - ***enable.auto.commit = false***
  - ***max.in.flight.requests.per.connection = 1***
  - ***acks = all***
  - ***min.insync.replicas > 1***