TM

Kafka Admin/Ops

Support around Cassandra and Kafka running in EC2

Kafka growing

Kafka Admin, Ops, DevOps

Kafka Admin
Kafka Ops
Kafka DevOps
Production Systems

Kafka Topic Creation Important for Operations

- Topics are added and modified using the topic tool

```
#!/usr/bin/env bash

cd ~/kafka-training

kafka/bin/kafka-topics.sh \
    --create \
    --zookeeper localhost:2181 \
    --replication-factor 3 \
    --partitions 23 \
    --topic stock-prices \
    --config min.insync.replicas=2
```

- ***Replication factor*** - replicas count amount of Kafka Brokers needed
  - use replication factor of at least 3 (or 2)
    - survive outages, head-room for upgrades and maintenance -ability to bounce servers
- ***Partition count*** - how much topic log will get sharded
  - determines broker count - if you have a partition count of 3, but have 5 servers, 2 not host topic log
  - consumers parallelism - active consumer count in consumer group

Modifying Topics

- You can modify topic configuration
- You can add partitions
  - ***existing data partition don't change!***
  - Consumers semantics could break, data is not moved from existing partitions to new partitions
- You can use ***bin/kafka-topics.sh —alter*** to modify a topic
  - add partitions - ***you can't remove partitions!***
  - ***you can't change replication factor!***
  - modify config or ***delete*** it
- You can use ***bin/kafka-topics.sh —delete*** to delete a topic
  - Has to be enabled in Kafka Broker config - ***delete.topic.enable=true***

Review of Kafka Topic Tools

Create Topic

```
#!/usr/bin/env bash

cd ~/kafka-training

## Create a new Topic
kafka/bin/kafka-topics.sh \
    --create \
    --zookeeper localhost:2181 \
    --replication-factor 2 \
    --partitions 3 \
    --topic stock-prices \
    --config min.insync.replicas=1 \
    --config retention.ms=60000
```

Describe Topic

```
#!/usr/bin/env bash
cd ~/kafka-training

# Describe existing topic
kafka/bin/kafka-topics.sh \
    --describe \
    --topic stock-prices \
    --zookeeper localhost:2181
```

Delete Topic

```
#!/usr/bin/env bash
cd ~/kafka-training

# Delete Topic
kafka/bin/kafka-topics.sh \
    --delete \
    --zookeeper localhost:2181 \
    --topic stock-prices
```

Alter Topic

alter-topic.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

## Alter the topic
kafka/bin/kafka-topics.sh --alter \
    --zookeeper localhost:2181 \
    --partitions 13 \
    --topic stock-prices \
    --config min.insync.replicas=2 \
    --delete-config retention.ms

```

- Changes ***min.insync.replicas*** from 1 to 2
- Changes partition count (***partitions***) from 3 to 13
- Use ***—delete-config*** to delete ***retention.ms*** configuration

TM

Modifying Topics with Alter

```
$ bin/delete-topic.sh
Topic stock-prices is marked for deletion.

$ bin/create-topic.sh
Created topic "stock-prices".

$ bin/describe-topic.sh
Topic:stock-prices      PartitionCount:3        ReplicationFactor:2     Configs:retention.ms=
        Topic: stock-prices     Partition: 0    Leader: 1       Replicas: 1,2   Isr: 1,2
        Topic: stock-prices     Partition: 1    Leader: 2       Replicas: 2,0   Isr: 2,0
        Topic: stock-prices     Partition: 2    Leader: 0       Replicas: 0,1   Isr: 0,1

$ bin/alter-topic.sh
Adding partitions succeeded!

$ bin/describe-topic.sh
Topic:stock-prices      PartitionCount:13       ReplicationFactor:2     Configs:min.insync.re
        Topic: stock-prices     Partition: 0    Leader: 1       Replicas: 1,2   Isr: 1,2
        Topic: stock-prices     Partition: 1    Leader: 2       Replicas: 2,0   Isr: 2,0
        Topic: stock-prices     Partition: 2    Leader: 0       Replicas: 0,1   Isr: 0,1
      …
        Topic: stock-prices     Partition: 11   Leader: 0       Replicas: 0,1   Isr: 0,1
        Topic: stock-prices     Partition: 12   Leader: 1       Replicas: 1,0   Isr: 1,0
```

Kafka Broker Graceful Shutdown

- Kafka Clustering detects Kafka broker shutdown or failure
  - Elects new partition leaders
  - For maintenance shutdowns Kafka supports graceful shutdown
- Graceful shutdown optimizations - ***controlled.shutdown.enable=true***
  - Topic logs data synced to disk = faster log recovery on restart by avoiding log recovery and checksum validation
  - Partitions are migrated to other Kafka brokers
  - Clean, fast leadership transfers, reduces partitions unavailability
  - Controlled shutdown fails if replicas on broker do not have in-sync replicas on another server

Balancing Leadership

- When broker stops or ***crashes*** leadership moves to surviving brokers
  - crashed broker's partitions ***transfers*** to other replicas
  - If broker restarted becomes a ***follower*** for all its partitions
    - Recall only ***leaders*** read and write

```
bin/kafka-preferred-replica-election.sh \
  —zookeeper host:port
```

- kaka-preferred-eleciton.sh will rebalance leadership, OR
- Kafka Broker Config: ***auto.leader.rebalance.enable=true***
  - auto-balance leaders on change

Kafka balancing across racks

- Kafka has rack awareness
  - spreads same partition replicas to different racks or AWS AZ (EC2 availability zones)
  - Survive single rack or single AZ outage
  - broker config: ***broker.rack=us-west-2a***
- During topic creation, rack constraint used to span replicas to as many racks as possible
  - min(#racks, replication-factor)
- Assignment of replicas to brokers ensures leaders count per broker same, regardless rack distribution if racks have equal number of brokers
  - if rack has fewer brokers, then each broker in rack will get more replicas
  - keep broker count the same in each rack or AZ

Checking Consumer Position

- Useful to see position of your consumers
  - Especially MirrorMaker consumers
- Tool to show consumer position
  - ***bin/kafka-consumer-groups.sh***
- Shows ***Topic*** and which ***Client*** (client id) and Consumer (consumer id) from consumer group is working with which Topic ***Partition***
  - GUID for Consumer ID based on ***client id plus GUID***
- ***Shows Lag between Consumer and Log***
- ***Shows Lag between Producer and what consumer can see (replicated vs non-replicated)***

TM

kafka-consumer-groups Describe

check-consumer-offsets.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

BOOTSTRAP_SERVERS="localhost:9092,localhost:9093"

kafka/bin/kafka-consumer-groups.sh --describe  \
    --bootstrap-server "$BOOTSTRAP_SERVERS"    \
    --group StockPriceConsumer
```

- Using —describe
- Specifies bootstrap server lists not ZooKeeper
- Specifies name of ConsumerGroup
- Will show lag, etc. for every consumer in group

TM

kafka-consumer-groups Describe Output

```
$ bin/check-consumer-offsets.sh

TOPIC          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG  HOST        CLIENT-ID
stock-prices   5          910             910             0    /10.0.1.11  green-2
stock-prices   4          611             611             0    /10.0.1.11  green-1
stock-prices   2          949             949             0    /10.0.1.11  blue-2
stock-prices   6          39              39              0    /10.0.1.11  red-0
stock-prices   8          13              13              0    /10.0.1.11  red-2
stock-prices   1          13              13              0    /10.0.1.11  blue-1
stock-prices   3          1534            1534            0    /10.0.1.11  green-0
stock-prices   7          -               0               -    /10.0.1.11  red-1
stock-prices   0          611             611             0    /10.0.1.11  blue-0
```

- Shows ***Topic*** and which ***Client*** from the consumer group is working with which Topic ***Partition*** - Note also shows GUID for Consumer ID (not shown)
- ***Current offset*** is what is visible to Consumer (replicated to ISRs)
- ***Log end*** shows what the leader of has written

kafka-consumer-groups Describe Output Lagging

```
$ bin/check-consumer-offsets.sh

TOPIC          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG  HOST        CLIENT-ID
stock-prices   1          524             524             0    /10.0.1.11  blue-1
stock-prices   8          380             524             144  /10.0.1.11  red-2
stock-prices   7          0               0               0    /10.0.1.11  red-1
stock-prices   3          2959            3067            108  /10.0.1.11  green-0
stock-prices   0          909             1122            213  /10.0.1.11  blue-0
stock-prices   6          1464            1572            108  /10.0.1.11  red-0
stock-prices   5          1277            1421            144  /10.0.1.11  green-2
stock-prices   4          934             1122            188  /10.0.1.11  green-1
stock-prices   2          2464            2993            529  /10.0.1.11  blue-2
```

- Notice Partition 8, the replication is behind Current Offset is behind Log End
- Notice how partition 3 has 6x as many records as Partition 1
  - Could be an example of a hot spot!
- Notice how Partition 7 has no records so red-2 is idle!

Managing Consumer Groups

- ConsumerGroupCommand - ***kafka-consumer-groups.sh***
  - you can also list, describe, or delete consumer groups
- Delete restriction -
  - Only works with older clients
  - No need for new client API because group is deleted automatically when last committed offset for group expires
  - If using older consumers that relied on ZooKeeper then you can use ***—delete***

TM

List Consumers

list-consumers.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

BOOTSTRAP_SERVERS="localhost:9092,localhost:9093"

kafka/bin/kafka-consumer-groups.sh --list  \
    --bootstrap-server "$BOOTSTRAP_SERVERS"

```

- Use —list to get a list of consumers

Expanding Kafka cluster

- Adding Kafka Brokers to cluster is simple
  - need unique broker id
  - new Kafka Brokers are not automatically assigned Topic partitions
  - You need to migrate partitions to it
- Migrating Topic Partitions is manually initiated
  - New Kafka Broker becomes followers of partitions
  - When it becomes ISR set member, then it gains leadership over partitions assigned to it
  - Once it becomes leader, existing replica will delete partition data if needed
- Kafka provides a partition reassignment tool

Kafka Partition Reassignment Tool

- partition can be moved across brokers
- avoid hotspots, balance load on brokers
- you have to look at load on Kafka Broker
  - use ***kafka-consumer-groups.sh***
  - other admin tools to find hotspots (top, KPIs, etc.)
  - balance as needed

Kafka Partition Reassignment Tool - Modes

- **GENERATE A PLAN —*generate***
  - Inputs: Topics List, and Kafka Broker List
  - Generates ***reassignment plan*** to move all topic partitions to new Kafka Brokers
- **EXECUTE A PLAN —execute**
  - Input: ***reassignment plan*** (--***reassignment-json-file***)
  - Action: Does partition reassignment using plan
- **CHECK STATUS OF EXECUTE PLAN —verify**
  - Shows status of —execute
  - Outputs: Completed Successfully, Failed or In-Progress

TM

Generate Partition Reassignment Plan

reassign-partitions-generate-plan.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

# Generate Reassignment Plan
kafka/bin/kafka-reassign-partitions.sh --generate \
    --broker-list 0,1,2,3 \
    --topics-to-move-json-file "$CONFIG/move-topics.json" \
    --zookeeper localhost:2181 > "$CONFIG/assignment-plan.json"
```

move-topics.json

```
{ "version": 1,
  "topics": [
    { "topic": "stocks" },
    { "topic": "stock-prices" }]}
```

- Added 4th Broker! Now we want it to have some partitions
- ***move-topics.json*** - list of topics to move in JSON format
- Generates assignment plan which needs to be edited

TM

Generated Partition Assignment Plan

assignment-plan.json

partitions replicas

```
1  {
2    "version": 1,
3    "partitions": [
4      {
5        "topic": "stocks",
6        "partition": 7,
7        "replicas": [
8          0,
9          1,
10         2
11       ]
12     },
13     {
14       "topic": "stocks",
15       "partition": 2,
16       "replicas": [
17         3,
18         2,
19         0
20       ]
```

- Assignment Plan
- List of Partitions
- List of Replicas
- Replicas might be moved to new Kafka Broker after plan executes
- Need to execute plan

TM

Execute Partition Reassignment Plan

reassign-partitions-execute-plan.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

# Execute reassignment plan
kafka/bin/kafka-reassign-partitions.sh --execute \
    --reassignment-json-file "$CONFIG/assignment-plan.json" \
    --throttle 100000 \
    --zookeeper localhost:2181
```

- Executes reassignment plan
- Use generated plan or use modified generated plan
- Set throttle rate (optional) so it does not all happen at once
  - reduces load on Kafka Brokers

Monitor Executing Partition Reassignment Plan

reassign-partitions-verify-plan.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

# Verify executing reassignment plan
kafka/bin/kafka-reassign-partitions.sh --verify \
    --reassignment-json-file "$CONFIG/assignment-plan.json" \
    --zookeeper localhost:2181
```

- Verify/Monitor reassignment plan
- Use generated plan or use modified generated plan that is already running
- Let’s you know when the plan is done

Decommissioning Kafka Brokers

- After we add a new broker,
  - add it to the —broker-list
  - Run generate plan
  - Execute plan
- To decommission Kafka Broker
  - Remove it from the —broker-list
  - Run generate plan, execute generate plan

TM

Generate Partition Reassignment Plan

reassign-partitions-generate-plan.sh

```
#!/usr/bin/env bash
CONFIG=`pwd`/config
cd ~/kafka-training

# Generate Reassignment Plan
kafka/bin/kafka-reassign-partitions.sh --generate \
    --broker-list 0,1,2 \
    --topics-to-move-json-file "$CONFIG/move-topics.json" \
    --zookeeper localhost:2181 > "$CONFIG/assignment-plan.json"
```

- Remove 4th Broker (3)! Now we want it reassign its partitions
- Generates assignment plan that moves partitions to 0,1,2

Setting quotas

- You can configure quotas for client-id and user using ***kafka-configs.sh***
- Clients receive an unlimited quota
- You can set custom quotas for
  - (user, client-id) pair
  - user
  - client-id

Setting quota for client-id, user Pair

quota.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

## Add limit for stock_analyst user running as clientId stockConsumer
kafka/bin/kafka-configs.sh --alter \
    --zookeeper localhost:2181 \
    --add-config 'producer_byte_rate=1024,consumer_byte_rate=2048' \
    --entity-type users \
    --entity-name stock_analyst \
    --entity-type clients \
    --entity-name stockConsumer
```

- User stock_analyst
- client id stockConsumer

Quota Configuration

- Order of precedence for quota configuration is:

1. /config/users/<user>/clients/<client-id>
2. /config/users/<user>/clients/<default>
3. /config/users/<user>
4. /config/users/<default>/clients/<client-id>
5. /config/users/<default>/clients/<default>
6. /config/users/<default>
7. /config/clients/<client-id>
8. /config/clients/<default>

TM

Default Quota for Users

quota-default-user.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

## Add limit to default user
kafka/bin/kafka-configs.sh --alter \
    --zookeeper localhost:2181 \
    --add-config 'producer_byte_rate=512,consumer_byte_rate=512' \
    --entity-type users --entity-default
```

- Sets default quota for users

Default Quota for Clients

quota-default-client.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

## Add limit to default client
kafka/bin/kafka-configs.sh --alter \
    --zookeeper localhost:2181 \
    --add-config 'producer_byte_rate=512,consumer_byte_rate=512' \
    --entity-type clients --entity-default
```

- Sets default quota for clients

TM

Describe a Quota

quota-describe.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

## Describe a quota
kafka/bin/kafka-configs.sh --describe \
    --zookeeper localhost:2181 \
    --entity-type users \
    --entity-name stock_analyst \
    --entity-type clients \
    --entity-name stockConsumer
```

- You can see what quotas are set for a user

TM

Describe a Quota Output

```
$ bin/quota-describe.sh

Configs for user-principal 'stock_analyst', client-id 'stockConsumer'
are producer_byte_rate=1024,consumer_byte_rate=2048
```

- Output from describe quota

Multi-Datacenters Deploys

- Kafka may need to spans multiple datacenters or AWS regions
- Recommended approach deploy local Kafka cluster per datacenter
  - application and services using Kafka should be in same datacenter
  - Use mirroring between clusters in different datacenters
- Reduces latency from Kafka to application and services using Kafka avoid working over WAN
- Centralizes mirroring between data centers so it can be monitored
- If applications needs a global view of all data from all clusters
  - Use mirroring to provide clusters data from each cluster into one aggregate cluster
  - Aggregate clusters used by applications that require full data set
- Suggestion for most use cases

If you need to cross WAN or DCs, ok

- Kafka batches and compresses records
  - Both producer and consumer can achieve high-throughput even over a high-latency connection
  - If needed increase the TCP socket buffer sizes for the producer, consumer, and broker
    - ***socket.send.buffer.bytes*** and ***socket.receive.buffer.bytes***
- Not a good idea to span DCs or regions
  - Really bad for ZooKeeper
  - More outages due to latency

TM

Important Client Configurations

- Producer configurations control
  - acks
  - compression
  - batch size
- Consumer Configuration
  - fetch size

TM

A Production Server Config

server-0.properties

```
## Increment by 1 for each broker
broker.id=0

# Kafka should have its own dedicated disk(s) or use SSD(s)
# To increase reads and writes, add more disks/log dirs JBOD.
log.dirs=./logs/kafka-0

## Log config
default.replication.factor=3
num.partitions=8

## Data must be replicated to at least two brokers
min.insync.replicas=2

## Don't allow un-managed topics for production
auto.create.topics.enable=false

## Run brokers spread over AZs or Racks
broker.rack=us-west2-a

## Number of concurrent requests allowed
queued.max.requests=1000

## Allow leaders to auto rebalance
auto.leader.rebalance.enable=true
```

TM

Java GC config

```
-Xmx6g
-Xms6g
-XX:MetaspaceSize=96m
-XX:+UseG1GC
-XX:MaxGCPauseMillis=20
-XX:InitiatingHeapOccupancyPercent=35
-XX:G1HeapRegionSize=16M
-XX:MinMetaspaceFreeRatio=50
-XX:MaxMetaspaceFreeRatio=80
```

- Use G1 GC
- Heap Space should be 25% to 35% of available space for server
- Leave 50% for OS, Remember Kafka uses OS page cache
- Other tweaks for GC to limit overhead

LinkedIn cluster

- One of LinkedIn's busiest clusters has:
  - 60 Kafka brokers
  - 50,000 partitions
  - Replication factor 2
  - Does 800k messages/sec in
  - 300 MB/sec inbound (writes/producers)
  - 1 GB/sec+ outbound (reads/consumers)
- 21 ms pause for 90% GC
- Less than 1 young GC per second

Hardware and OS

- Dual quad-core Intel Xeon machines with 24GB of memory or higher
  - for production mission critical system
  - 24 GB total but only 25% of that for JVM (6 GB)
- Kafka Broker needs memory to buffer active readers and writers
  - to buffer for 30 seconds and memory needed is write_throughput*30
- Disk throughput is important
  - 8x7200 rpm SATA drives
  - Disk throughput is often performance bottleneck
  - JBOD - more disks is better

OS

- Kafka production usually runs on Linux
- Ensure you have enough file descriptors
  - Kafka uses file descriptors for log segments and open connections
  - (number_of_partitions)*(partition_size/segment_size) + number_of_producer_connections + number_of_consumer_connections
  - Start with 100,000 or more file descriptors
- Max socket buffer size:
  - increased to enable high-performance data transfer between data centers
- Use JBOD instead of RAID, RAID ok, JBOD better
- Check flusher threads and PDF Flush but defaults should be ok
- Prefer filesystem XFS (largeio, nobarrier), EXT4 ok too (data=writeback, commit=num_secs, nobh, delalloc)

Monitoring

- Kafka uses Yammer Metrics
  - metrics reporting for Kafka Broke, Consumers and Producers
  - Reports stats using pluggable stats reporters
- Metrics exposed via JMX
- You can see what metrics are available with jconsole

Kafka Broker Metrics -1 of 3

DESCRIPTION	JMX MBEAN NAME
Message in rate	kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec
Byte in rate	kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec
Request rate	kafka.network:type=RequestMetrics,name=RequestsPerSec,request={Produce|FetchConsumer|FetchFollower}
Byte out rate	kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec
Log flush rate and time	kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs
Time request waits in request queue	kafka.network:type=RequestMetrics,name=RequestQueueTimeMs,request={Produce|FetchConsumer|FetchFollower}
Time request is processed at leader	kafka.network:type=RequestMetrics,name=LocalTimeMs,request={Produce|FetchConsumer|FetchFollower}
Messages count consumer lags behind producer	kafka.consumer:type=consumer-fetch-manager-metrics,client-id={client-id} Attribute: records-lag-max

TM

Kafka Broker Metrics - 2 of 3

Under replicated Count partitions	kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions	0
Is controller active on broker?	kafka.controller:type=KafkaController,name=ActiveControllerCount	Only 1 Kafka Broker is controller and has 1. All else should have 0.
Leader election rate	kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs	>0 if failures
Unclean leader election rate	kafka.controller:type=ControllerStats,name=UncleanLeaderElectionsPerSec	0
Partition counts	kafka.server:type=ReplicaManager,name=PartitionCount	mostly even across brokers
Leader replica counts	kafka.server:type=ReplicaManager,name=LeaderCount	mostly even across brokers
ISR shrink rate	kafka.server:type=ReplicaManager,name=IsrShrinksPerSec	If a broker dies, ISR shrinks for some partitions. ISR expands when brokers come back.
ISR expansion rate	kafka.server:type=ReplicaManager,name=IsrExpandsPerSec	Opposite of ISR shrink rate

Kafka Broker Metrics - 3 of 3

Max follower lag	kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=Replica	lag usually proportional to produce maximum batch size
Messages Lag per follower	kafka.server:type=FetcherLagMetrics,name=ConsumerLag,clientId=([-.\w]+),topic=([-.\w]+),partition=([0-9]+)	lag usually proportional to producer maximum batch size
Requests waiting in producer purgatory	kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Produce	>0 if ack=all is used
Requests waiting in fetch purgatory	kafka.server:type=DelayedOperationPurgatory,name=PurgatorySize,delayedOperation=Fetch	size depends on consumer config fetch.wait.max.ms
Request total time	kafka.network:type=RequestMetrics,name=TotalTimeMs,request={Produce|FetchConsumer|FetchFollower}	broken into queue, local, remote and response send time
Leader replica counts	kafka.server:type=ReplicaManager,name=LeaderCount	Should be even

Common Metrics for Clients 1 of 2

Metric	Description
connection-close-rate	Connections closed per second JMX MBean Name kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+)
connection-creation-rate	New connections established per second
network-io-rate	Average network operations count on all connections per second.
outgoing-byte-rate	Average outgoing bytes count sent per second to all servers.
request-rate	Average requests count sent per second.
request-size-avg	Average size of all requests
request-size-max	Maximum size of any request

TM

Common Metrics for Clients 2 of 2

Metric	Description
incoming-byte-rate	Average incoming byte count received by all sockets JMX MBean Name (kafka.[producer|consumer|connect]:type=[producer|consumer|connect]-metrics,client-id=([-.\w]+))
response-rate	Responses received sent per second.
select-rate	I/O layer checked for new I/O to perform per second count
io-wait-time-ns-avg	Average duration I/O thread spent waiting for a socket ready for reads/writes
io-wait-ratio	Fraction of time the I/O thread spent waiting.
io-time-ns-avg	Average duration for I/O per select call in nanoseconds.
io-ratio	Fraction of time I/O thread spent doing I/O.
connection-count	Current number of active connections.

Per Kafka Broker Client Monitoring

Metric	Description
outgoing-byte-rate	Average outgoing byte count sent per second for node JMX MBean Name: kafka.producer:type=[consumer|producer|connect]-node-metrics,client-id=([-.\w]+),node-id=([0-9]+)
request-rate	Average requests count sent per second for a node.
request-size-avg	Average size of all requests for node
request-size-max	Maximum size of any request sent for node
incoming-byte-rate	Average responses received count per second for node
request-latency-avg	Average request latency in ms for node
request-latency-max	Maximum request latency in ms for node
response-rate	Responses received sent per second for node

TM

Kafka Producer Monitoring - 1 of 3

Metric	Description
waiting-threads	User threads blocked count waiting for buffer memory to enqueue their records. JMX MBean Name kafka.producer:type=producer-metrics,client-id=([-.\w]+)
buffer-total-bytes	Maximum buffer memory size client can use
buffer-available-bytes	Total buffer memory size that is not being used
bufferpool-wait-time	Fraction of time an appender waits for space allocation
batch-size-avg	Average byte count sent per partition per-request.
batch-size-max	Max byte count sent per partition per-request.
compression-rate-avg	Average compression rate of record batches.
record-queue-time-avg	Average time in ms record batches spent in record accumulator.
record-queue-time-max	The maximum time in ms record batches spent in the record accumulator.

TM

Kafka Producer Monitoring - 2 of 3

Metric	Description
request-latency-avg	Average request latency in ms. JMX MBean Name kafka.producer:type=producer-metrics,client-id=([-.\w]+)
request-latency-max	Maximum request latency in ms.
record-send-rate	Average record count sent per second
records-per-request-avg	Average record count per request
record-retry-rate	Average per-second retried record send count
record-error-rate	Average per-second record send count that resulted in errors.
record-size-max	Maximum record size.
record-size-avg	Average record size.
requests-in-flight	Current number of in-flight requests - waiting for a response.

TM

Kafka Producer Monitoring - 3 of 3

Metric	Description
metadata-age	Age in seconds of current producer metadata being used
record-send-rate	Average records sent count per second for topic
byte-rate	Average bytes sent count per second for topic
compression-rate	Average record batches compression rate for topic
record-retry-rate	Average per-second retried record send count for a topic
record-error-rate	Average per-second record sends that resulted in errors count for topic
produce-throttle-time-max	Maximum time in ms a request was throttled by a broker
produce-throttle-time-avg	Average time in ms a request was throttled by a broker
requests-in-flight	Current number of in-flight requests - waiting for a response.

Kafka Consumer Group Monitoring - 1 of 2

Metric	Description
commit-latency-avg	Average duration for commit request **kafka.consumer:type=consumer-coordinator-metrics,client-id=([-.\w]+)**
commit-latency-max	Max duration for a commit request
commit-rate	Commit call count per second
assigned-partitions	Partition count currently assigned to consumer
heartbeat-response-time-max	Max duration for heartbeat request to receive response
heartbeat-rate	Average heartbeat count per second
join-time-avg	Average duration for a group rejoin
join-time-max	Max duration for a group rejoin
join-rate	Group join count per second

Kafka Consumer Group Monitoring - 2 of 2

Metric	Description
sync-time-avg	Average duration for a group sync
sync-time-max	Max duration for a group sync
sync-rate	Group sync count per second
last-heartbeat-seconds-ago	Second count since last controller heartbeat

Kafka Consumer Monitoring

Metric	Description
fetch-size-avg	Average byte size fetched per request
fetch-size-max	Maximum byte size fetched per request
bytes-consumed-rate	Average byte count consumed per second
records-per-request-avg	Average record count in each request
records-consumed-rate	Average record count consumed per second
fetch-latency-avg	Average fetch request duration
fetch-latency-max	Max fetch request duration
fetch-rate	Fetch request count per second
records-lag-max	Max lag of record count for any partition
fetch-throttle-time-avg	Average throttle time in ms
fetch-throttle-time-max	Maximum throttle time in ms

TM

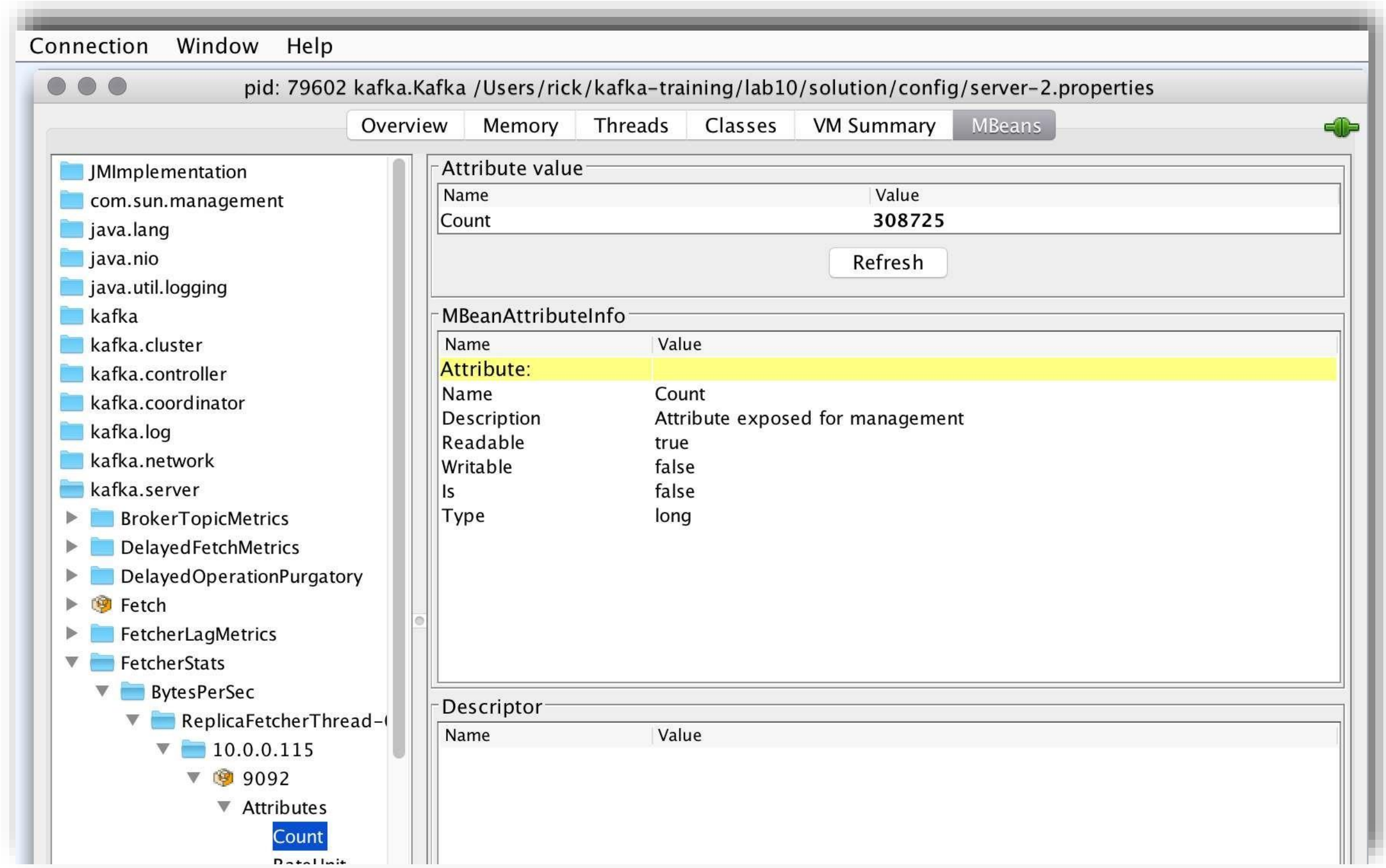
Kafka Consumer Topic Fetch Monitoring

Metric	Description
fetch-size-avg	Average byte size fetched per request for specific topic
fetch-size-max	Max byte size fetched per request for specific topic
bytes-consumed-rate	Average byte size consumed per second for specific topic
records-per-request-avg	Average record count per request for specific topic
records-consumed-rate	Average record count consumed per second for specific topic

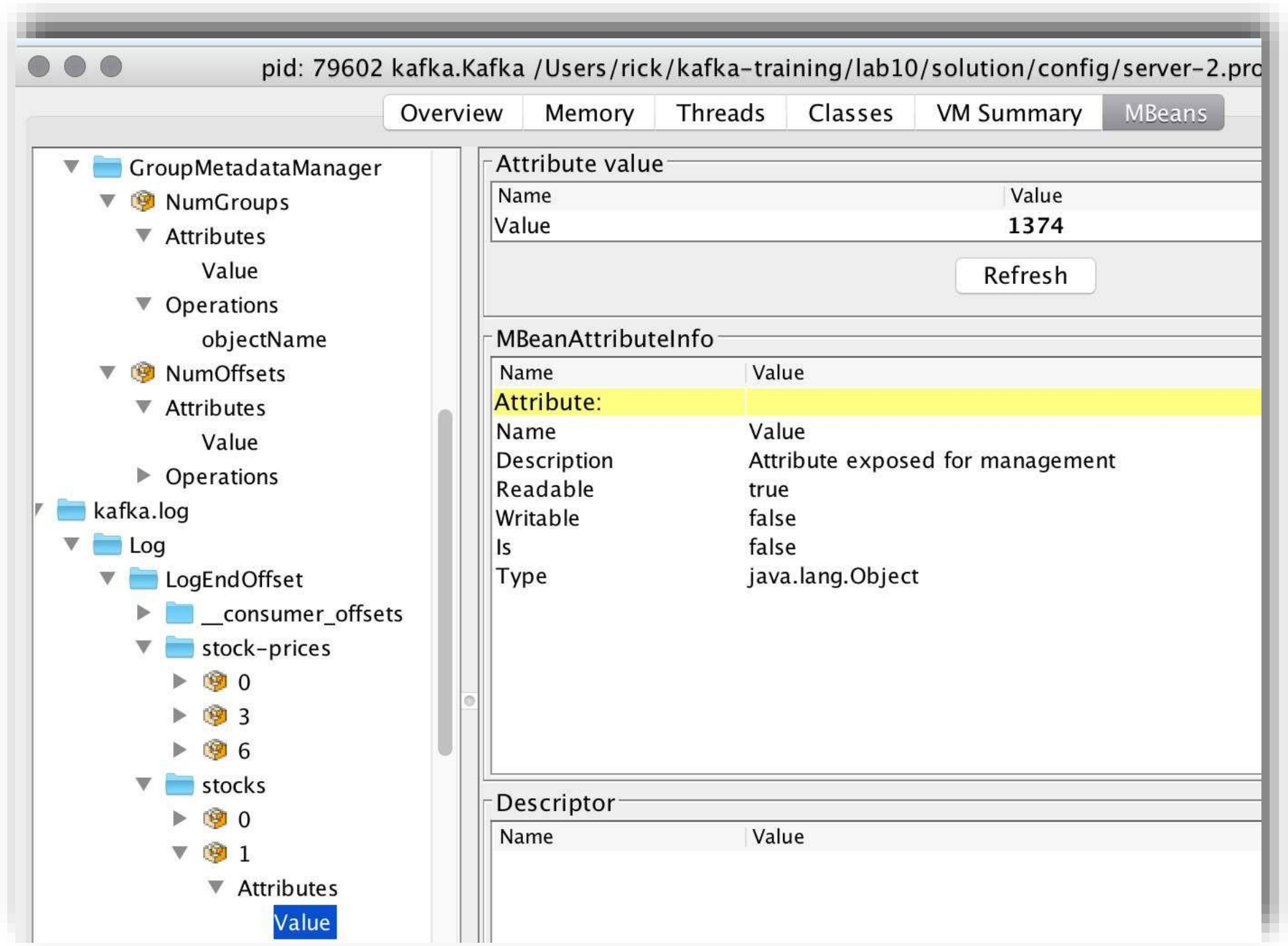
Other Metrics

- Low level metrics
- Thread metrics
- Task Metrics
- Processor Node Metrics
  - Forwarding to other nodes
- State Store Metrics
- Good idea to monitor GC, JVM threads, etc.
- See metrics available with JConsole

Kafka Broker Metrics via JConsole 1 of 2

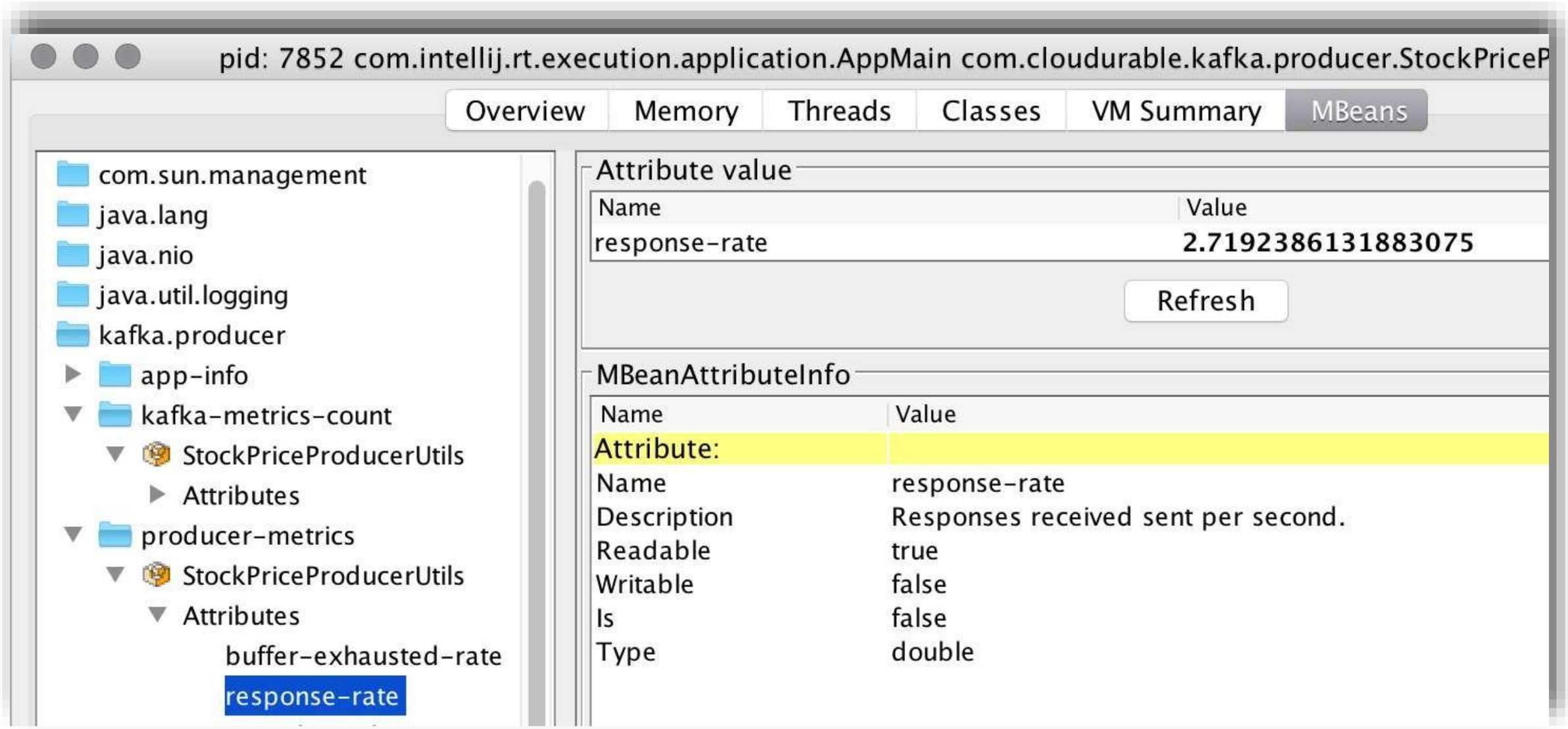


Kafka Broker JConsole Metrics 2 of 2



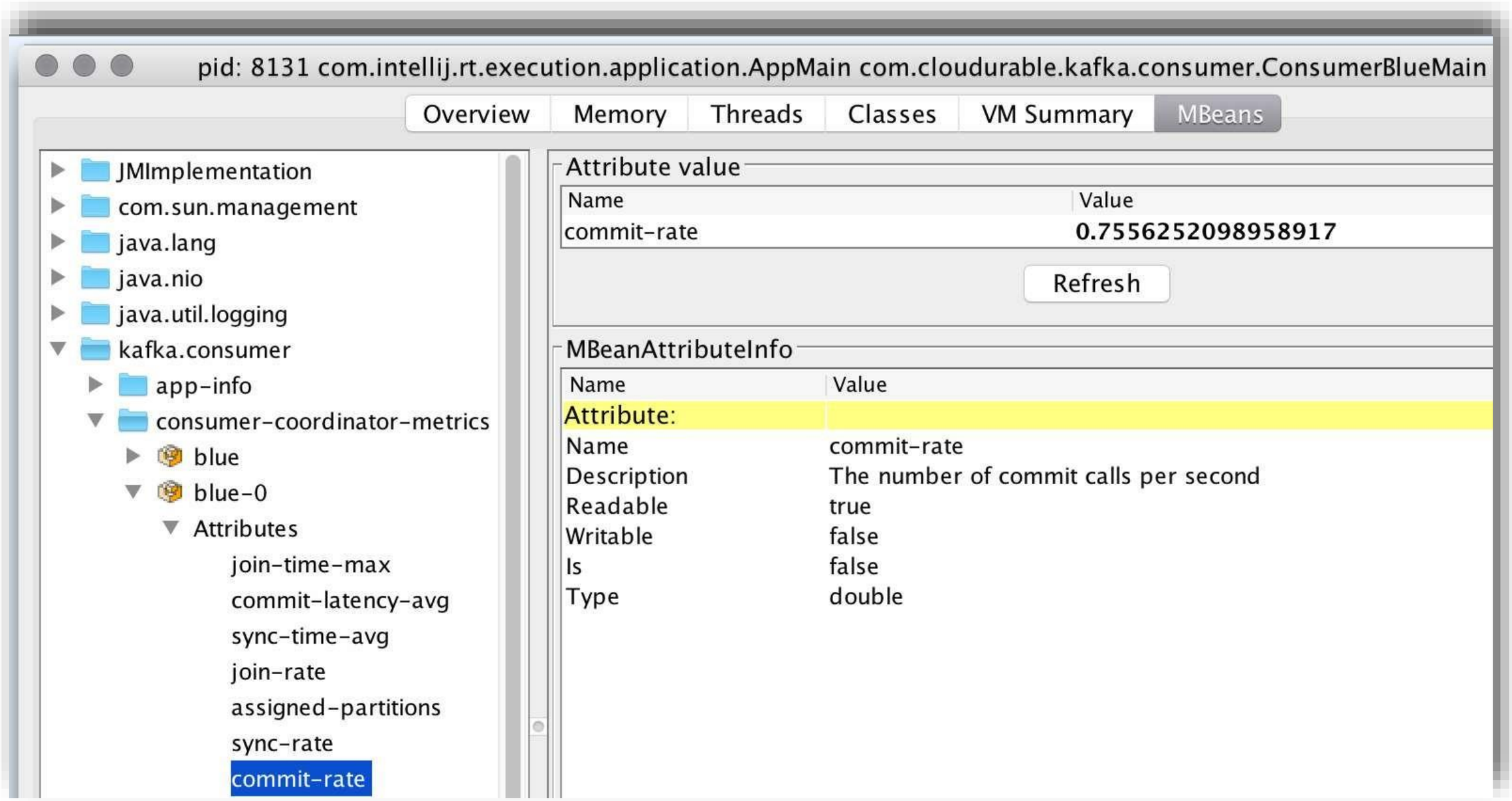
TM

Kafka Producer Metrics JConsole



TM

Kafka Consumer JConsole Metrics



ZooKeeper Setup 1 of 3

- Don’t put all ZooKeeper nodes in same same rack or in a single AWS availability Zones
- Decent hardware; don’t use T2 Micro
- Use 5 to 7 servers for production tolerates 2 to 3 servers down
- For small deployment using 3 servers is ok (only 1 allowed down)
- Put transaction logs on dedicated disk group (***dataLogDir***)
- Put snapshots, message log, and OS on another disk/disk group (***dataDir***)
- Writes to transaction log are synchronous batches
  - Concurrent writes can significantly affect performance

ZooKeeper Setup 2 of 3

- Use dedicated ZooKeeper cluster for Kafka
- ZooKeeper needs 3 to 5GB of heap with some room for OS (30% to 50% of System total)
- Monitoring ZooKeeper use JMX and or 4 letter words
- Keep ZooKeeper cluster small
  - Reduce quorums on the writes and subsequent cluster member updates
  - But don't go too small either
  - More ZooKeeper servers increases read capacity of ZooKeeper

ZooKeeper Setup 3 of 3

- ZooKeeper requires little administration, but…
- ZooKeeper takes periodic snapshots of its data
  - snapshot plus log can rebuild ZooKeeper state
- ZooKeeper does not purge snapshots by default
  - Let’s you back up snapshots
- You want to purge snapshots so disk does not fill up
  - autopurge.snapRetainCount (how many snapshots to keep)
  - autopurge.purgeInterval (duration in hours)
- Make sure you use rolling log files for logging