Main page
Recent changes
Server admin log (Prod)
Server admin log (RelEng)
Deployments
SRE/Operations Help
Incident status

Cloud VPS & Toolforge
Cloud VPS documentation
Toolforge documentation
Request Cloud VPS project
Server admin log (Cloud VPS)

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Incident documentation/20180711-kafka-eqiad

< Incident documentation

Work is still in progress.

Contents [hide]

1 Background
2 Summary
3 Kafka considerations
4 Eventbus considerations
5 Mirror Maker considerations
6 EventStreams considerations
7 Timeline
8 Conclusions
8.1 Kafka
8.2 Eventstreams
9 Links to relevant documentation
9.1 Kafka
10 Actionables

Background

Before starting, a little summary about the Kafka main clusters. We run Kafka on multiple clusters (Jumbo, Analytics, and Main eqiad/codfw), but the ones discussed in this report are Kafka main eqiad (`kafka100[1-3]`) and Kafka main codfw (`kafka200[1-3]`). They run one Kafka process on each node (called broker), forming two separate clusters (one in eqiad and one in codfw) that are not aware of each other. Each cluster handles several topics, each one of them is split in partitions (basically the topic is a queue, that is split into multiple partitions for scalability). Each topic can have multiple consumer groups, namely group of processes that coordinates to read partitions. There is only one process reading each partition within the same consumer group, and there might be "idle" ones waiting to take over in case of failures etc.. Change Prop is an example of a service that handles multiple consumer groups to Kafka Main: one consumer group for each topic that it is subscribed to.

Mirror Maker is another piece of the puzzle that is mentioned in this report. It is basically a consumer/producer, that runs on each Kafka main host consuming data from some assigned topics of the other cluster and producing the same data to the local cluster. For example, on `kafka100[1-3]` there are Mirror Maker instances that consume topics from the Kafka main codfw cluster and push the same data to Kafka main eqiad (prefixing the topics with "codfw." to avoid duplicate topic names on the same cluster). This flow of data is meant to ensure that important data/topics is replicated in both datacenters.

Zookeeper is used by Kafka for various things, mostly bookkeeping: it stores the list of topics, what brokers are alive, what is the ISR (In Sync Replicas) set for a specific topic/partition, what ACLs are set for a given topic, etc.. In previous versions it used to keep also the last offset for a given consumer, but now this functionality is maintained in Kafka itself (in special topics called __consumer* for the curious). Kafka main-eqiad uses the Zookeeper main-eqiad cluster (conf100[4-6]) meanwhile Kafka main-codfw uses the Zookeeper main-codfw cluster (conf200[1-3]), so no state is shared between DCs.

Summary

Services team maintains a dev version of ChangeProp in `restbase-dev` cluster, that's configured to read a sample of events from production `main-kafka` cluster and generate load for dev RESTBase version. It's configured not to post any messages back to Kafka or anyhow else interact with it except consuming. This installation was not updated for a while, and apparently did not contain an important bug fix. On July 11 at 14:16 a trivial deployment of a config change to production ChangeProp was made which made the dev cluster rebalance and think a new topic was added, which triggered a re-subscribtion to a new set of topics. Due to a bug, it subscribed to all the topics existing, automatically creating retry topics for them. This initiated an endless loop of creating retry-retry topics, which went unnoticed.

Creating lots of topics each 10 seconds brought Kafka into a degraded state at which point attempts to restart it were made. However, Kafka cluster could not start up due to OOM errors and too low `Xmx/Xms` settings. The faulty topics were identified as a cause of Kafka cluster problems, so and attempt was made to delete the topics, which hit a bug in core Kafka, bringing the whole cluster into unrecoverable state. In an attempt to decrease load and finally restart Kafka, all the consumers were switched off, but this still did not help, so JobQueue and ChangeProp were transferred to codfw.

After the core reason of Kafka not starting up was recognized, several attempts to remove faulty topics were made - first gently through Kafka-cli, that was not successful due to Kafka fetching faulty topics back from Zookeper, and then by hard-deleting topics from Zookeeper and from the disk on the broker nodes - that appeared to be successful and Kafka cluster was brought back online.

Kafka considerations

Kafka suffered three different issues during the outage, two that might be related and one caused by a pre-existing core bug. For reason that are still not clear, after restbase-dev started to misbehave creating new topics two severe errors were registered:

- **`java.lang.OutOfMemoryError: Java heap space`**
- **`java.io.IOException: Too many open files`**

The former sadly does not have a heap dump that we can analyze and from the logs it is not completely clear what triggered it, meanwhile the latter seems a simple file descriptor exhaustion due to change prop creating topics and opening sockets to Kafka. On `kafka1001` the Kafka process is handling, in steady state, on average `~15500 fds` opened (~11000 sockets, 4200 opened files). The original limit was 65k in the Kafka systemd unit, raised to infinity while dealing with the outage. The burst of TCP connections and open files triggered by change-prop running on restbase-dev should be the responsible for breaching the 65k limit:

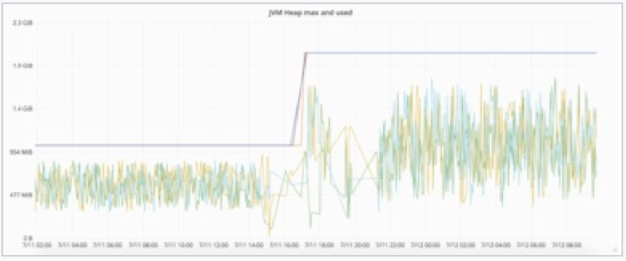
- 2086 topic added, with three partitions each, and a minimum of three files opened in turn for each partition (two index files and the first log file) should result in ~12500 more open files, that added to the regular baseline it means reaching 28k+.
- almost all the TCP sockets opened are created by change-prop (one consumer group for each topic more or less, with all the nodejs workers on the host subscribed to each of them) and the baseline is 10k, so we can make a simple proportion: 10000 : 460 = x : 2086 => x = 45300

If we sum 28000 + 45300 we easily breach the 65k limit, reaching a whopping 73300 open file descriptors!

The third problem was caused in an attempt to delete the faulty topics created by change-prop, hitting https://issues.apache.org/jira/browse/KAFKA-4893. The complete recovery happened after shutting down all the brokers, deleting any trace of the faulty topics in Zookeeper and on the brokers' disk partitions, and finally restarting again the brokers.

It must be said that the aforementioned issues were not isolated, but they were present at the same time and have probably influenced each other (especially the first two). As the timeline states, Kafka on `kafka1002` was restarted at around 15:20 UTC to raise the maximum open files limit, but the same thing didn't happen on `kafka1001` that was showing the same behavior. By the time that Kafka on `kafka1001` was restarted, the first run of the topic cleanup script was already running, causing the third problem.

The out of memory error is still a big question mark, but a few considerations about the current usage might help in having a better picture in mind. As stated in the timeline, one action item that was done during the outage was to raise the maximum heap size for the Kafka broker JVMs from 1G to 2G (`-Xms2g -Xmx2g`), as it is clear for the picture. Before the outage every Kafka broker was consuming up to 800MB of heap, and the GC was regularly cleaning up the young generation without any sign of distress. This means that when change-prop started to create topics and opening connection, in several moments ~200MB were left to use before ending up in a OOM. It is interesting to notice how after the change in maximum heap size, the baseline changed as well showing a +500MB peak increase in heap consumption (and a slight decrease in young generation's collections).



Kafka heap size used vs maximum

This does not mean that a JVM needs to work with a huge heap and low young generation activity to be healthy, but only that without any restriction on what clients can do there is a little margin allowed before ending up trashing when a sudden change in client's behavior happens.

An interesting test was made in the analytics labs project to verify how Kafka's JVM heap size react to more topics added. A three nodes cluster, similar to the main-eqiad prod one, was already there with a few topics added, and mostly no traffic. Roughly 500 topic were added, and then the real experiment started. A script added one topic at the time until the cluster reached a bit more than 2000 topics, and the heap size of one of the brokers was observed via Jconsole, leading to the following graph.

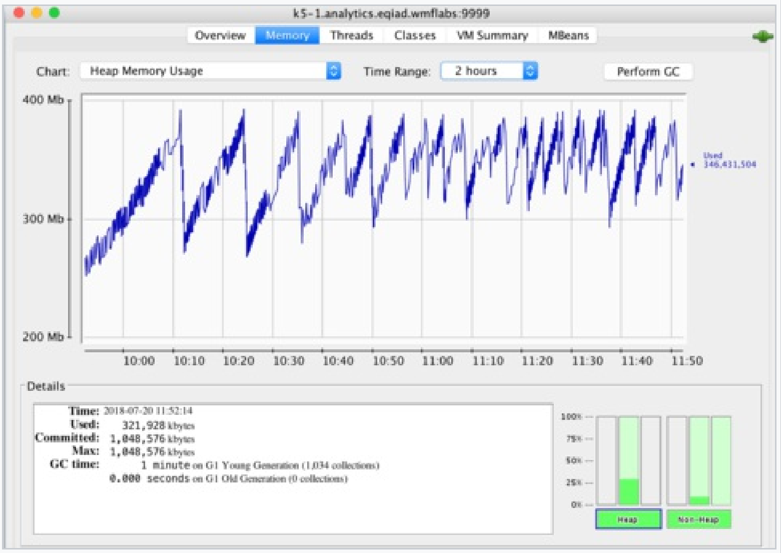


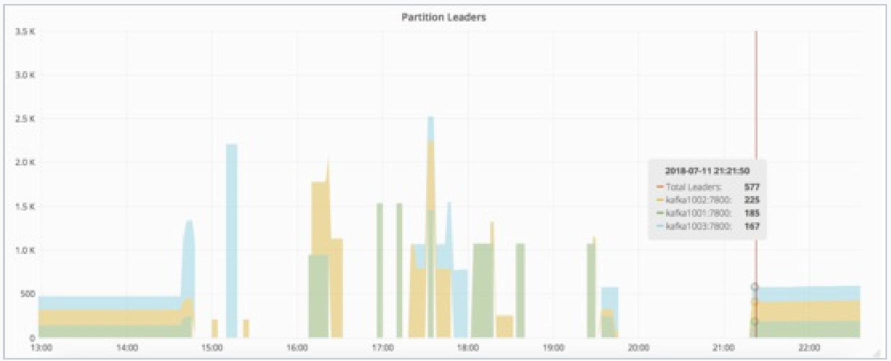
Chart: Heap Memory Usage
Time Range: 2 hours

```
      Time: 2018-07-20 11:52:14
      Used:    321,928 kbytes
Committed: 1,048,576 kbytes
       Max: 1,048,576 kbytes
   GC time:        1 minute on G1 Young Generation (1,034 collections)
             0.000 seconds on G1 Old Generation (0 collections)
```

JVM heap size consumption of a Kafka broker in labs while adding 1500 topics (one by one).

The graph was mistakenly cut on its left side but the baseline heap consumption (starting point) of the experiment was around 230MB. It is very clear how adding topics is not a free operation, albeit it must be said that the trade off between heap used and topics added is not that bad. This test was done simply adding one topic at the time to a cluster that was receiving no traffic, and it is clear to see that the overall heap usage bumped up by 100/120MB. As described before, the 1G heap size of the main-eqiad's Kafka brokers showed peaks of 800+ used MBs, so adding 2000 topics in a small time (not a couple of hours like the test) surely contributed to the aggravation of the brokers' health, almost certainly causing the Java OutOfMemory errors registered.

It is also possible that the excessive number of file descriptor opened on `kafka1001` and `kafka1002` might have slowed down the brokers and their topic partition replication threads, leading to a trashing state that caused some out of memory exceptions as well.



Kafka Partition distribution during the outage. The healthy distribution is depicted at the very left and right of the graph, where every broker is leader for roughly the same number of partitions. During the outage brokers were intermittently taking the lead of an imbalanced number of partitions due to the other brokers being down or starting up.

Interesting numbers also for the number of times that Kafka brokers have been started up (calculated grepping the number of times that the startup script preamble was seen in the logs):

- `kafka1001` - 11 (between 2018-07-11 14:52 and 2018-07-11 21:13 UTC)
- `kafka1002` - 10 (between 2018-07-11 14:52 and 2018-07-11 21:14 UTC)
- `kafka1003` - 6 (between 2018-07-11 15:04 and 2018-07-11 21:16 UTC)

Precise UTC timeline of Kafka brokers starting (tried to align each broker's timeline to better cluster the starting timings):

- `1001` 14:52:07
  - `1002` 14:52:46
    - `1003` 15:04:30
  - `1002` 15:09:39
- `1001` 15:11:00
  - `1002` 15:14:14
  - `1002` 15:18:27
    - `1003` 16:19:58
- `1001` 16:52:59
- `1001` 17:06:26
  - `1002` 17:11:21
    - `1003` 17:13:50
- `1001` 17:29:20
- `1001` 17:59:28
  - `1002` 18:12:34
- `1001` 18:30:53
- `1001` 19:21:46
  - `1002` 19:26:33
- `1001` 19:31:39
  - `1002` 19:32:43
    - `1003` 19:33:26
  - `1002` 19:37:00
    - `1003` 19:39:30
- `1001` 19:41:27
- `1001` 21:13:35
  - `1002` 21:14:05
    - `1003` 21:16:23

Note: Not all of these are manual actions, there are also forced startup due to periodic puppet runs.

Another interesting note is about the timings of the Java out of memory exceptions registered in the Kafka broker logs. All three brokers were intermittently logging those exceptions for various reasons, but their last occurrence logged differs a lot:

`1001` : 2018-07-11 17:57:37

`1002` : 2018-07-11 16:46:45

`1003` : 2018-07-11 16:27:01

As reported in the timeline, at around 17 UTC the JVM's maximum heap size was bumped to 2G for all the brokers, but only `1001` kept failing for out of memory exception after that.

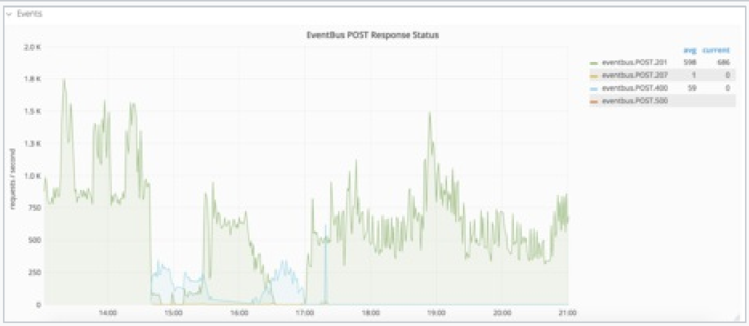
Eventbus considerations

The Eventbus service demonstrated to be resilient to Kafka errors but of course it didn't work when all the brokers were unavailable. We successfully failed it over to codfw (even if the action should have been done way before), one of the few positive remarks of this outage since we hadn't it done before.

It is interesting to notice in the Eventbus POST response status graph that two main dips were registered (keep in mind though that the failover to codfw happened at around 16:57 UTC):

- from ~14:40 to ~15:30 UTC - this corresponds to the time window between the start of the outage and kafka on `kafka1002` restarted with increased max open files settings.
- from ~16:10 to ~17:10 UTC - this corresponds to the time window between the restart of kafka on `kafka1003` and the restart of all the Kafka brokers with increased max heap size setting.

This means that Eventbus was able to survive even with only one broker up, probably due to the fact that it doesn't require any broker to be in sync with the current partition leader (required acks in Kafka terminology) to consider an event enqueued.



Eventbus HTTP POST responses returned to clients during the Kafka outage.

Mirror Maker considerations

A major relief while handling the Kafka Main eqiad outage was the failover of Eventbus and Eventstreams to codfw, that went incredibly well. The Kafka Main codfw cluster though should have got the faulty topics replicated via Mirror Maker, what saved us was that those topics didn't receive any event, so Mirror Maker didn't replicate anything over to codfw and the other Kafka cluster was spared.

EventStreams considerations

The EventStreams service was impacted heavily when Kafka main eqiad started to misbehave, so we tried to fail it over to codfw right after Eventbus. The main problem of this service is that it allows external clients to consume from a fixed set of Kafka topics using an arbitrary offset. This means that every time that a failover occurs, all the clients' offset (related to eqiad Topics) will not be valid anymore when consuming from codfw, since they'll need to be reset. The clients then will need to either re-consume past events or implement logic to prevent duplicates or reprocessing (so the failover is impactful).

The failover procedure was also not clear (even if it was a simple hiera change), and it delayed further more the time to recovery of the service.

Timeline

All times are in UTC

- 2018-07-11 14:16 - [SAL] Started deploy [changeprop/deploy@2e56855]: Move static blacklisting to change-prop T198386
- 2018-07-11 14:17 - Kafka logs show a neat increase in messages like " **Preparing to rebalance group change-prop-$something**", indicating that a Change Prop instance was updating its Kafka consumer groups (probably re-subscribing to them).
- 2018-07-11 14:18 - [SAL] Finished deploy [changeprop/deploy@2e56855]: Move static blacklisting to change-prop T198386
- 2018-07-11 14:37 - First logs on Kafka indicating new partition topics being created. Some kafka brokers starts logging error messages like "**leader reported an error: UNKNOWN_TOPIC_OR_PARTITION**".
- 2018-07-11 14:38 - First Eventbus produce to kafka errors logged by icinga
- 2018-07-11 14:39 - Kafka Main Eqiad inbound traffic graphs shows a sudden drop.
- 2018-07-11 14:39 - Kafka brokers start showing up connection errors to each other (" **java.io.IOException: Connection to 100X was disconnected before the response was read**").
- 2018-07-11 14:44 - Kafka broker on kafka1002 starts logging " **java.io.IOException: Too many open files**"
- 2018-07-11 14:45 - Kafka on kafka1001 reports " **java.lang.OutOfMemoryError: Java heap space**" in the logs.
- 2018-07-11 14:47 - Kafka broker on kafka1001 starts logging " **java.io.IOException: Too many open files**"
- 2018-07-11 14:50 - Luca restarts kafka on kafka100[12]"
- 2018-07-11 14:53 - Kafka on kafka1003 reports " **java.lang.OutOfMemoryError: Java heap space**"
- 2018-07-11 14:59 - Giuseppe merges https://gerrit.wikimedia.org/r/#/c/operations/puppet/+/445183/ as first attempt to raise Kafka's max open files allowed (didn't work as expected).
- 2018-07-11 15:03 - Kafka on kafka1002 reports " **java.lang.OutOfMemoryError: Java heap space**"
- 2018-07-11 15:08 - [SAL] <Pchelolo> !log stop cpjobqueue in eqiad
- 2018-07-11 15:16 - Last "**java.io.IOException: Too many open files**" logged on kafka1002
- 2018-07-11 15:22 - Giuseppe merges https://gerrit.wikimedia.org/r/445188 as second attempt to raise Kafka's max open files allowed (worked). From puppet logs on kafka hosts, LimitNOFILE was 65536, but we didn't take a snapshot of the amount of file descriptors opened (via lsof or similar). **The change is applied**

only to the kafka1002's broker, at the time we didn't notice the same error on kafka1001 too (but not on kafka1003).

- 2018-07-11 15:33 - Filippo starts checking why the Kafka metrics are not showing up correctly in Grafana, identifying a timeout from the masters while scraping the Kafka's prometheus agents (probably due to the ton of new topics created).
- 2018-07-11 15:33 - Guillaume merges https://gerrit.wikimedia.org/r/445193 to remove a new WDQS Kafka consumer added the day before.
- 2018-07-11 16:06 - Luca tries kafka topics --delete --topic eqiad.change-prop.retry.change-prop.retry.ChangeNotification
- 2018-07-11 16:16 - Luca restarts Kafka on kafka1003 (Kafka on kafka1001 stil misbehaving due to too many open files).
- 2018-07-11 16:17 - Change prop stopped on restbase-dev1004
- 2018-07-11 16:27 - Last "**java.lang.OutOfMemoryError: Java heap space**" logged for kafka1003
- 2018-07-11 16:36 - Luca starts the cleanup procedure to delete the topics with duplicate naming (using kafka topics --delete --topic). At this time it wasn't clear what was the root cause of the issue, and the newly created topics were the only thing matching the start of the outage.
- 2018-07-11 16:40 - [SAL] <_joe_> masking and stopping cpjobqueue, changeprop everywhere
- 2018-07-11 16:46 - Last "**java.lang.OutOfMemoryError: Java heap space**" logged for kafka1002
- 2018-07-11 16:48 - Last "**java.io.IOException: Too many open files**" logged on kafka1001
- 2018-07-11 16:52 - Kafka of **kafka1001** gets restarted and **picks up the new LimitNOFILE config**. It threw errors for too many open files up to this point.
- 2018-07-11 16:57 - [SAL] oblivian@puppetmaster1001 conftool action : set/pooled=false; selector: dnsdisc=eventbus,name=eqiad
- 2018-07-11 17:03 - Luca **bumps** manually in /etc/default/kafka **the heap size of kafka** from Xmx1G Xms1G **to Xmx2G Xms2G** and restarts kafka brokers one at the time.
- 2018-07-11 17:57 - Last "**java.lang.OutOfMemoryError: Java heap space**" logged for kafka1001
- 2018-07-11 18:03 - Change prop stopped on restbase-dev1005.
- 2018-07-11 18:14 - Luca found in the Kafka logs an error message indicating a topic with correspondent filename on disk too long, causing cascading failures and eventually the shutdown of the broker.
- 2018-07-11 18:14 - Luca starts again the script on kafka1001 that was meant to execute kafka topics --delete --topic $topicname for each faulty/duplicate topic added by changeprop.
- 2018-07-11 19:18 - Luca's script finished its work. All the /srv/kafka/data directories were cleaned up from directories with faulty/duplicate topic names (Kafka completely stopped while doing it) and Kafka brokers restarted again. The attempt didn't work, same file name too long errors reported and cascading failures.
- 2018-07-11 20:03 - After a lot of attempts, Luca prepares a list of zookeeper paths to purge and a script executing zkCli.sh rmr $path-to-faulty-topic for each faulty/duplicate topic. The script starts executing.
- 2018-07-11 21:13 - Start kafka brokers after the zk cleanup script finished its work (remove again all the directories with long topic names in /srv/kafka/data). This time the brokers start up.
- 2018-07-11 21:44 - Luca re-enable puppet on kafka100[1-3] to bring up Mirror Maker again.
- 2018-07-12 08:18 - Giuseppe moves back Eventstreams to eqiad.
- 2018-07-12 08:22 - [SAL] oblivian@puppetmaster1001 conftool action : set/pooled=true; selector: dnsdisc=eventbus,name=.*

Conclusions

There are several topics that have been brought up in this incident report, it might be useful to break them down into multiple subsections for clarity.

Kafka

We don't have any solid fence against client misbehaving like change-prop in Kafka, plus during the outage there was very little expertise available (due to the maximum expert of the field being rightfully on vacation) and there was a big lag between the start of the outage and partial recovery (eventbus/eventstreams failed over to codfw). As demonstrated above, creating topics is not entirely cheap and we might want to review our policy of topic auto-creation, but a more generic discussion must be made about general authentication/authorization to the Kafka main-eqiad cluster, since any client within production now can theoretically put strain on Kafka simply bombarding it with requests (create topic, consumer group subscription, etc..).

Eventstreams

Eventstreams was designed to store state on the clients, and it wasn't designed to failover transparently between datacenters. A follow up of this incident report is to verify if anything can be done to improve this situation, for example transforming it to a service truly active/active. It is not an easy work and it might probably not be feasible, but a discussion is worth nonetheless.

Links to relevant documentation

Kafka

- Kafka/Administration
- EventBus/Administration
- EventStreams/Administration

Actionables

- Status: 🟥 **Pending** Separate dev Change-Prop from production Kafka cluster phab:T199427
- Status: 🟥 **Pending** Puppetize dev cluster change-prop installation phab:T199428
- Status: 🟥 **Pending** Consider the possibility of separating ChangeProp and JobQueue on Kafka level phab:T199431
- Status: 🟩 **Done** Upstream kafka ticket created: Deleting topics with long names can bring all brokers to unrecoverable state
- Status: 🟥 **Pending** Consider disabling automatic topic creation in main-kafka phab:T199432
- Status: 🟥 **Needs discussion** Consider getting rid of retry topics and re-posting messages back to original topic in ChangeProp
- Status: 🟩 **Done** Redesign EventStreams for better multi-dc support phab:T199433
- Status: 🟩 **Done** Document the process for hard-deleting topics in kafka phab:T199441
- Status: 🟩 **Done** Remove `kafka-mirror` unit from main kafka cluster phab:T199443
- Status: 🟩 **Done** Set a proper max open files limit for Kafka clusters phab:T200177
- Status: 🟥 **Pending** After switchover to codfw Job queue ChangeProp logged KafkaConsumer is not connected phab:T199444
- Status: 🟩 **Done** Check EventBus doesn't need broker sync to enqueue phab:T200025
- Status: 🟩 **Done** Raise Kafka Java heap size settings to 2G - gerrit:445304

Category: Incident documentation