A

Report on

Comparative Analysis of Message Broker Software

By:

Asrhdeep Singh Syal

Jubeen Shah

Rayan Dasoriya

Sujal

Table of Contents

Overview **3**
Implementation Details **4**
Apache JMeter **4**
Apache ActiveMQ 4
Apache Kafka 6
RabbitMQ 8
Gatling **9**
Apache Kafka 12
RabbitMQ 13
SonarLint **14**
Installation 14
Analyzing source code with SonarLint 14
FindBugs **16**
Installation 16
Reports Browsing 16
Configuration 17
Results **18**
Performance Analysis using Apache JMeter **18**
Throughput 18
Latency 27
SonarLint **28**
Apache Kafka SonarLint 28
ActiveMQ SonarLint 29
Combined Analysis 29
FindBugs **30**
FindBugs ActiveMQ Broker 30
FindBugs ActiveMQ Core 30
Community Insights **31**
Industrial Usage **31**
Popularity in Search **32**
Community Statistics **34**
Commits per year 34
Challenges **37**
Conclusion **38**
References **39**

Overview

This project is a comparative analysis of the message queuing broker tools like Apache ActiveMQ, Apache Kafka, and RabbitMQ on the basis of performance and error finding tools like JMeter, Gatling, SonarQube, FindBugs along with some interesting GitHub stats.

A message broker is a module which translates a message from the formal messaging protocol of the sender to the formal messaging protocol of the receiver. Message brokers are elements in telecommunication or computer networks where software applications communicate by exchanging formally-defined messages. The primary purpose of a broker is to take incoming messages from applications and perform some action on them. Message brokers can decouple end-points, meet specific non-functional requirements, and facilitate reuse of intermediary functions. For example, a message broker may be used to manage a workload queue or message queue for multiple receivers, providing reliable storage, guaranteed message delivery and perhaps transaction management. The following represent other examples of actions that might be handled by the broker.

Apache ActiveMQ is an open-source messaging and Integration Patterns server written in Java. It is fast, supports many Cross-Language Clients and Protocols, comes with easy to use Enterprise Integration Patterns and many advanced features.

Apache Kafka is an open-source stream-processing software platform written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. The advantage of Kafka's model is that every topic has both queuing and publish-subscribe model—it can scale processing and is also multi-subscriber—there is no need to choose one or the other.

RabbitMQ is an open-source message broker software written in Erlang that originally implemented the Advanced Message Queuing Protocol and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol, Message Queuing Telemetry Transport, and other protocols.

In the coming section, we have demonstrated about the usage of tools like Apache JMeter, Gatling, SonarLint, FindBugs to determine the performance measures, i.e. throughput and latency along with the errors, bugs and the community support of these open source software.

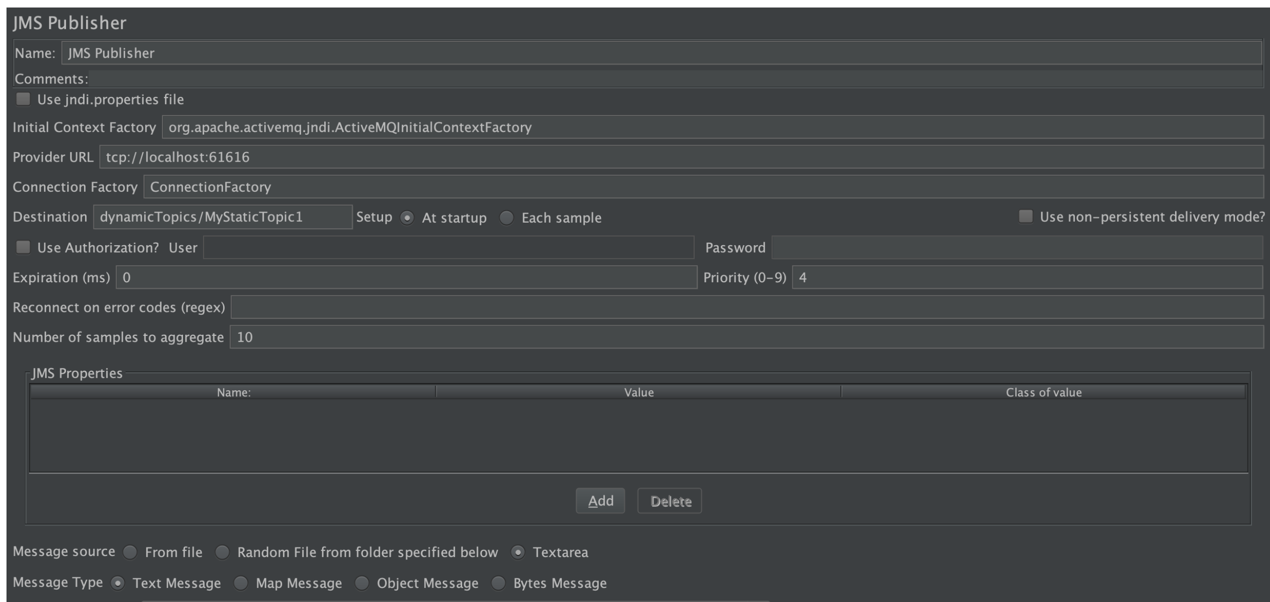
Implementation Details

Apache JMeter

The **Apache JMeter** is an open source Java based software, designed to load test functional behavior and measure performance. Apache JMeter may be used to test performance and to simulate a heavy load on a server, group of servers, network or object to test its strength or to analyze overall performance under different load types. We have used JMeter to find two parameters: throughput and latency of the software.

Apache ActiveMQ

- Publisher Configuration on JMeter



- Subscriber Configuration on JMeter

JMS Subscriber

Name: JMS Subscriber

Comments:

Use jndi.properties file

Initial Context Factory: org.apache.activemq.jndi.ActiveMQInitialContextFactory

Provider URL: tcp://localhost:61616

Connection Factory: ConnectionFactory

Destination: dynamicTopics/MyStaticTopic1 — Setup: At startup / Each sample (selected)

Durable Subscription ID

Client ID

JMS Selector

Use Authorization?

User

Password

Number of samples to aggregate: 10

Store Response (checked)

Timeout (ms): 2000

Client: Use MessageConsumer.receive() (selected) / Use MessageListener.onMessage() — Stop between samples?

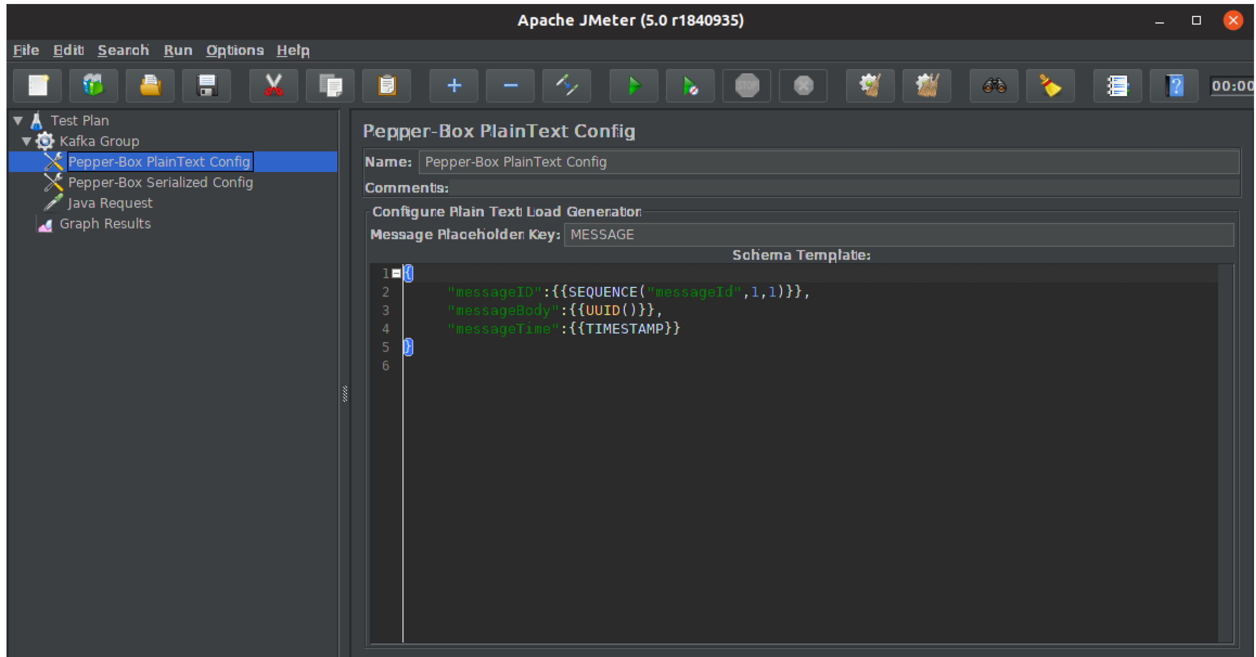
Separator

Reconnect on error codes (regex)

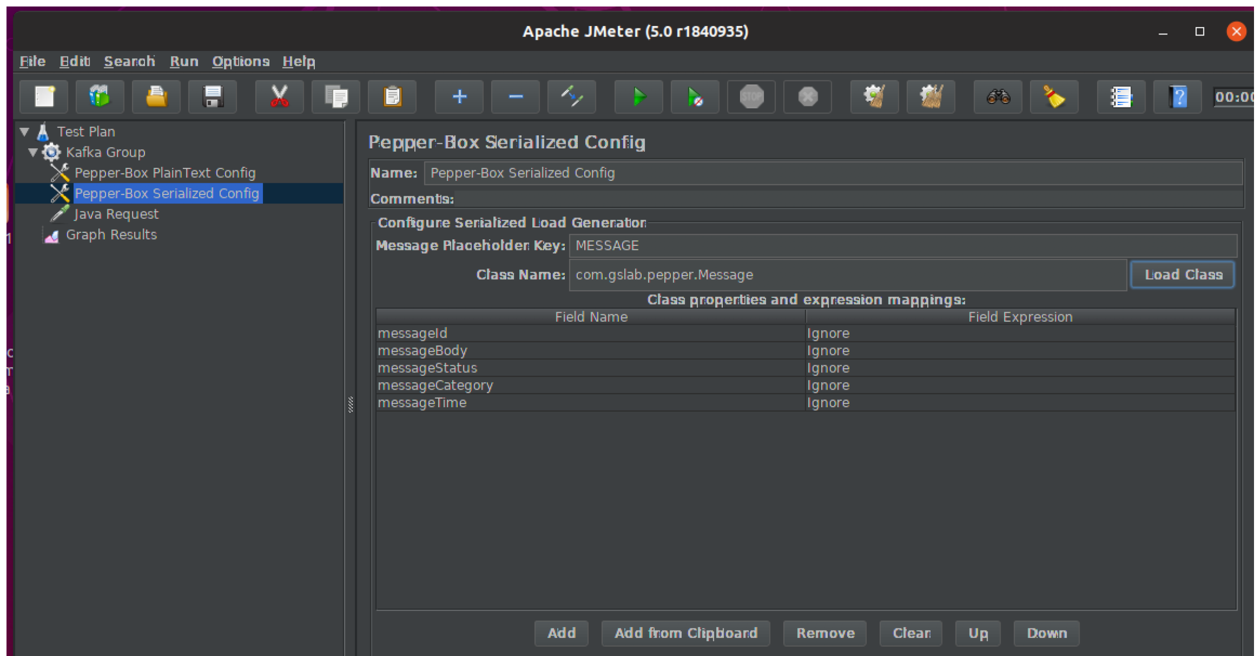
Pause between errors (ms)

Apache Kafka

- Publisher Configuration on JMeter

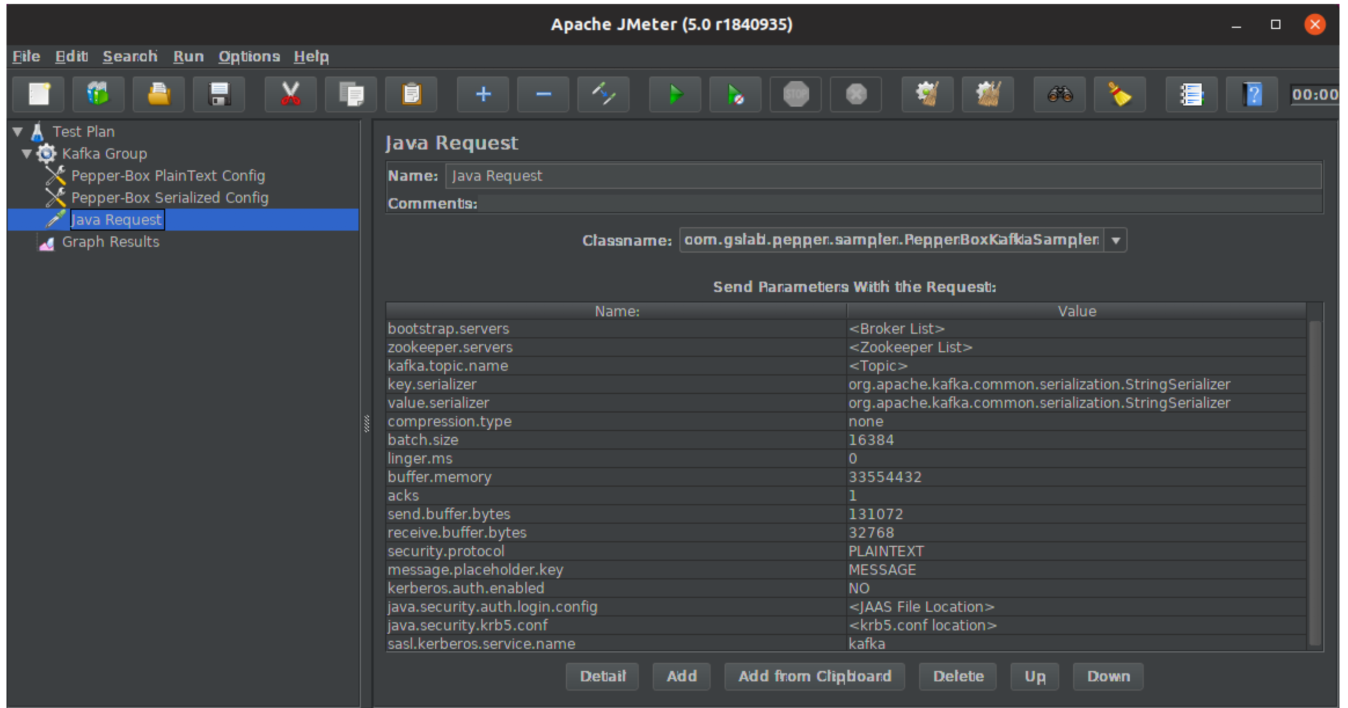


- Consumer Configuration on JMeter



https://github.com/rayandasoriya/Message_Broker_Analysis

- Java Request Configuration



RabbitMQ

- Thread Group Configuration on JMeter

Thread Group

Name: RabbitMQ Group

Comments:

Action to be taken after a Sampler error

(•) Continue ( ) Start Next Thread Loop ( ) Stop Thread ( ) Stop Test ( ) Stop Test Now

Thread Properties

Number of Threads (users): 1000

Ramp-Up Period (in seconds): 1

Loop Count: [ ] Forever 4

[ ] Delay Thread creation until needed

[ ] Scheduler

Scheduler Configuration

Duration (seconds)

Startup delay (seconds)

Start Time 2016/06/13 10:24:20

End Time 2016/06/13 10:24:20

- Publisher Configuration on JMeter

AMQP Publisher

Name: AMQP Publisher

Comments:

Exchange

Exchange auditlogs Exchange Type fanout

[ ] Durable? [ ] Redeclare?

Queue

Queue logstashqueue

Routing Key [ ] Durable? [ ] Redeclare?

Message TTL [ ] Exclusive

Expires [ ] Auto Delete?

Connection

Virtual Host /

Host localhost

Port 5672 [ ] SSL?

Username guest

Password guest

Timeout 1000

Number of samples to Aggregate 1

[ ] Persistent?

[ ] Use Transactions?

Routing Key

Message Type

Reply-To Queue

Correlation Id

ContentType

Message Id

https://github.com/rayandasoriya/Message_Broker_Analysis

Gatling

Gatling is an open-source load and performance testing framework based on Scala, Akka and Netty. The software is designed to be used as a load testing tool for analyzing and measuring the performance of a variety of services, with a focus on web applications. Two years ago, Gatling officially presented Gatling FrontLine, Gatling's Enterprise Version with additional features.

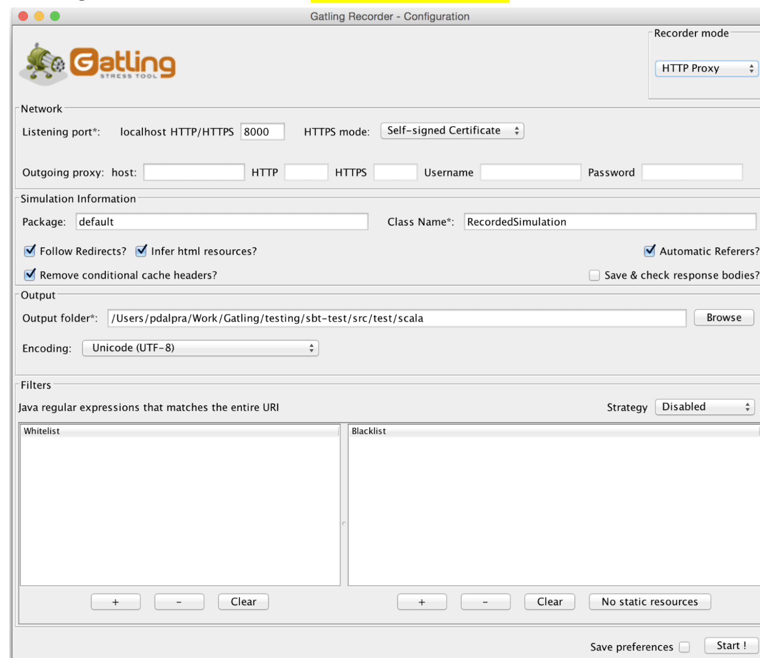
Reasons for choosing Gatling:

1. Enhanced user experience
2. Fast and quick results for improving the development cycle
3. Works better with REST APIs
4. Anticipates slow response times and crashes

Introduction to Gatling implementation working on hosted computer database and Gatling recorder.

System: MacBook Pro (2.9 GHz Intel Core i7, 16 GB)

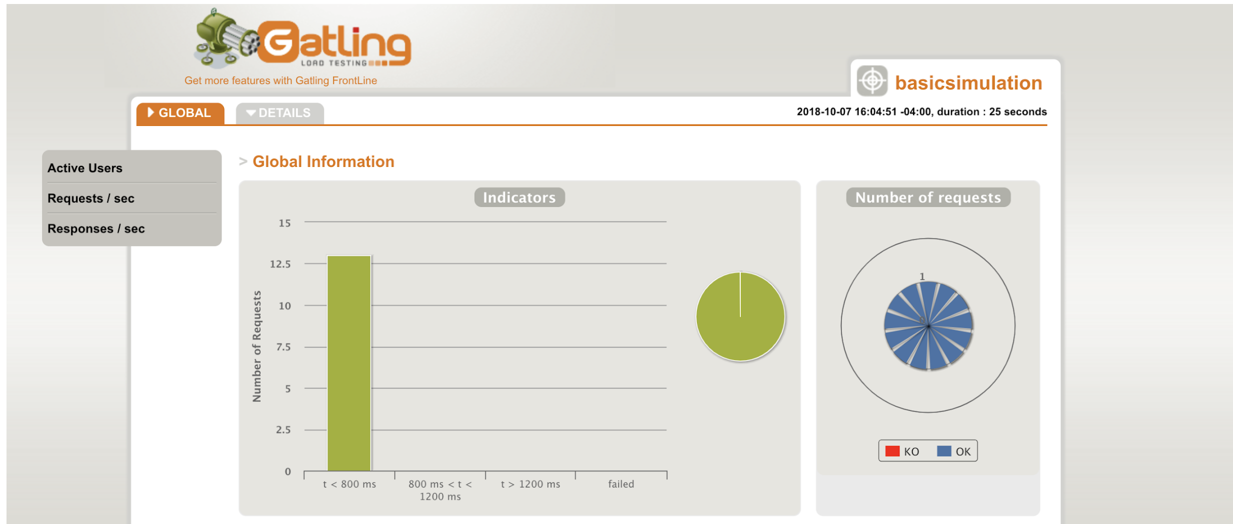
- API deployed at (http://computer-database.gatling.io/computers) hosting a computer database
- Using Gatling's recorder GUI: *bin/recorder.sh*



- To capture CRUD activities.
- Configure the recorder according to the network specifications

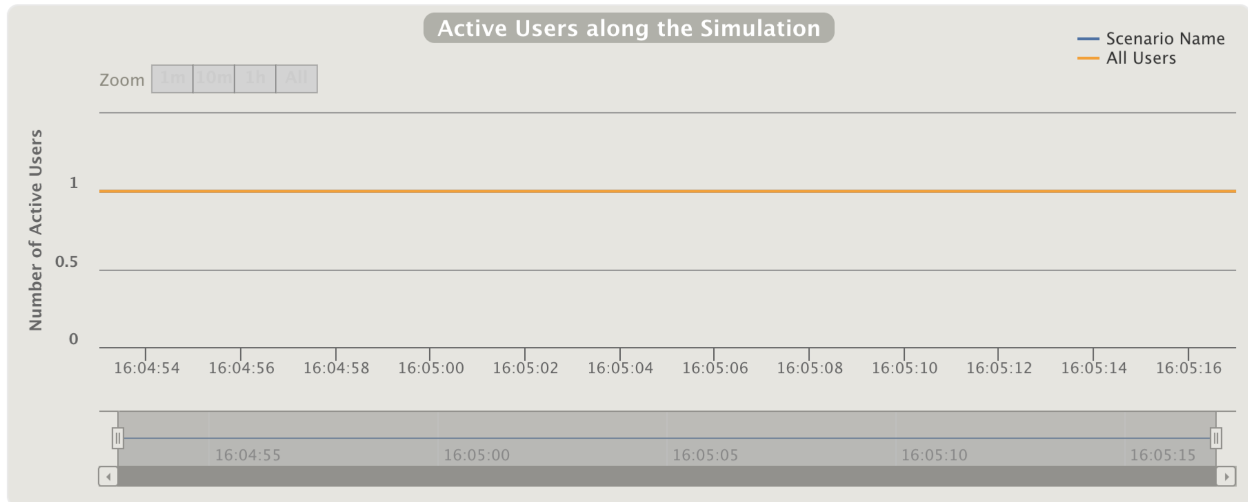
- If the recording is successfully captured, a simulation file is generated under *user-files/simulations/computerdatabase*
- Once the simulation file is successfully created, run Gatling using: *bin/gatling.sh*

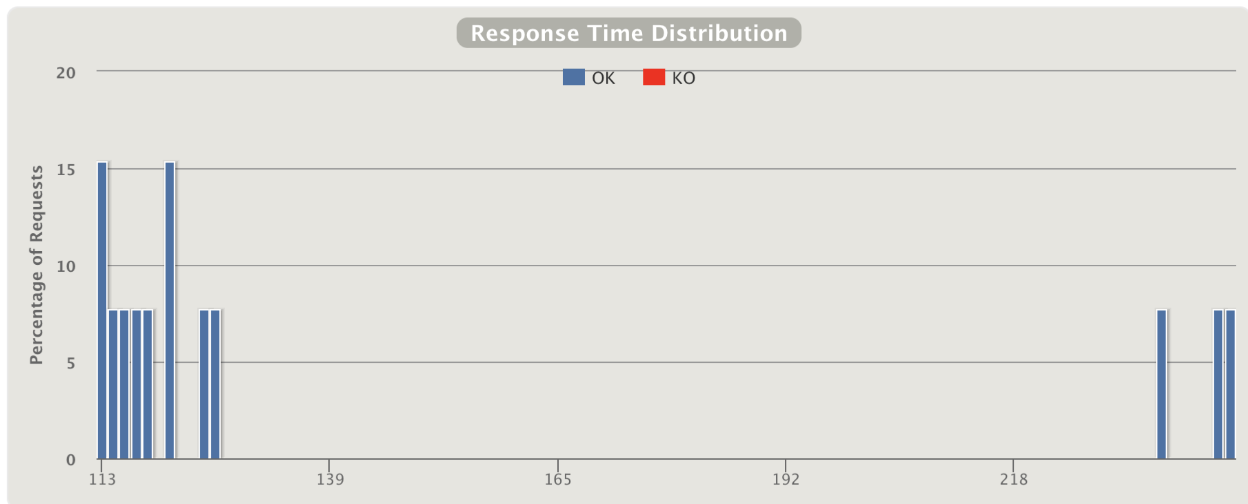
Sample Results:

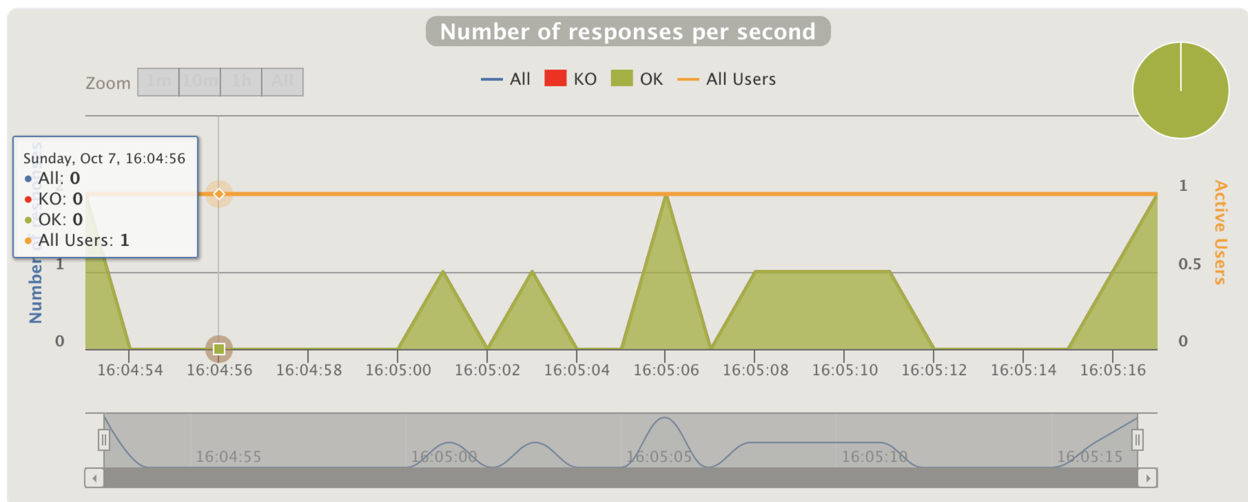


STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	13	13	0	0%	0.52	112	120	126	243	244	244	146	52
request_1	1	1	0	0%	0.04	242	242	242	242	242	242	242	0
request_...direct 1	1	1	0	0%	0.04	124	124	124	124	124	124	124	0
request_2	1	1	0	0%	0.04	236	236	236	236	236	236	236	0
request_3	1	1	0	0%	0.04	126	126	126	126	126	126	126	0
request_4	1	1	0	0%	0.04	116	116	116	116	116	116	116	0
request_...direct 1	1	1	0	0%	0.04	121	121	121	121	121	121	121	0
request_5	1	1	0	0%	0.04	114	114	114	114	114	114	114	0
request_6	1	1	0	0%	0.04	118	118	118	118	118	118	118	0
request_7	1	1	0	0%	0.04	120	120	120	120	120	120	120	0
request_8	1	1	0	0%	0.04	115	115	115	115	115	115	115	0
request_9	1	1	0	0%	0.04	244	244	244	244	244	244	244	0
request_10	1	1	0	0%	0.04	112	112	112	112	112	112	112	0
request_...direct 1	1	1	0	0%	0.04	112	112	112	112	112	112	112	0







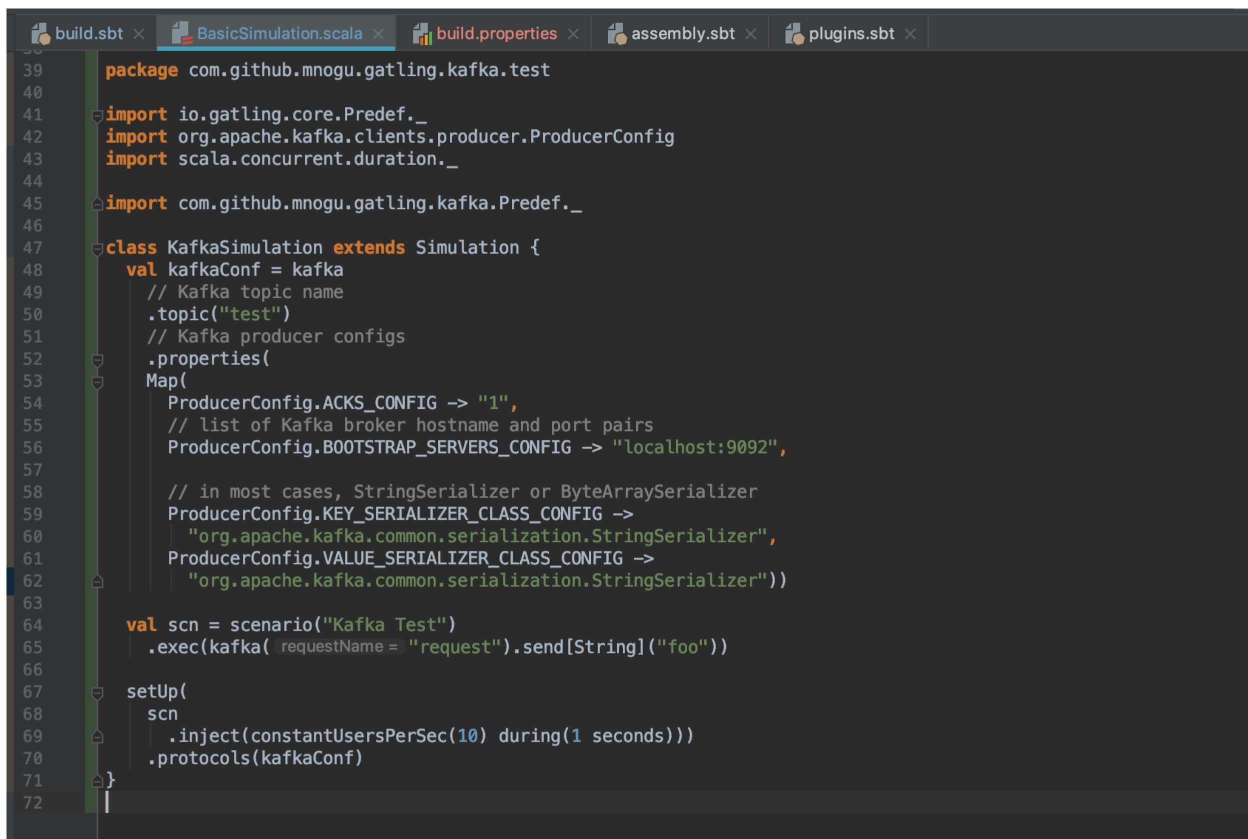
https://github.com/rayandasoriya/Message_Broker_Analysis

The analysis with two servers was the first attempt towards working with build tools, Scala and IntelliJ. The learning curve was steep working with multiple dependencies of Gatling, Scala, Java, SBT and Sever Releases.

Apache Kafka

- Versions studied: 2.12-2.0.0, 2.10-0.10
- Dependency Manager: SBT
- IDE: IntelliJ
- Language: Scala
- Using Gatling's plugin supporting the producer API, jar was build provided to Gatling. Following is a basic simulation file used to stress test the server.

Simulation configuration file:



```
build.sbt   BasicSimulation.scala   build.properties   assembly.sbt   plugins.sbt

39  package com.github.mnogu.gatling.kafka.test
40
41  import io.gatling.core.Predef._
42  import org.apache.kafka.clients.producer.ProducerConfig
43  import scala.concurrent.duration._
44
45  import com.github.mnogu.gatling.kafka.Predef._
46
47  class KafkaSimulation extends Simulation {
48    val kafkaConf = kafka
49      // Kafka topic name
50      .topic("test")
51      // Kafka producer configs
52      .properties(
53      Map(
54        ProducerConfig.ACKS_CONFIG -> "1",
55        // list of Kafka broker hostname and port pairs
56        ProducerConfig.BOOTSTRAP_SERVERS_CONFIG -> "localhost:9092",
57
58        // in most cases, StringSerializer or ByteArraySerializer
59        ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG ->
60          "org.apache.kafka.common.serialization.StringSerializer",
61        ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG ->
62          "org.apache.kafka.common.serialization.StringSerializer"))
63
64    val scn = scenario("Kafka Test")
65      .exec(kafka( requestName = "request").send[String]("foo"))
66
67    setUp(
68      scn
69        .inject(constantUsersPerSec(10) during(1 seconds)))
70      .protocols(kafkaConf)
71  }
72
```

However, we were constantly facing the following error but couldn't get a workaround for the same. According to our understanding, the Gatling (2.2) was not able to find a

class named *"io/gatling/commons/util/ClockSingleton"* which would've been used as a benchmark the clock timings to measure various parameters of the protocol. Since we were not able to execute Kafka (2.10-0.10) earlier, we tried implementing the same with latest Kafka version and couldn't get through this issue.

```
1
Simulation com.github.mnogu.gatling.kafka.test.KafkaSimulation started...
Uncaught error from thread [GatlingSystem-akka.actor.default-dispatcher-3] shutting down JVM since 'akka.jvm-exit-on-fatal-error' i
s enabled for ActorSystem[GatlingSystem]
java.lang.NoClassDefFoundError: io/gatling/commons/util/ClockSingleton$
        at com.github.mnogu.gatling.kafka.action.KafkaRequestAction$$anonfun$com$github$mnogu$gatling$kafka$action$KafkaRequestActi
on$$sendRequest$1.apply(KafkaRequestAction.scala:65)
        at com.github.mnogu.gatling.kafka.action.KafkaRequestAction$$anonfun$com$github$mnogu$gatling$kafka$action$KafkaRequestActi
on$$sendRequest$1.apply(KafkaRequestAction.scala:56)
        at io.gatling.commons.validation.Success.map(Validation.scala:32)
        at com.github.mnogu.gatling.kafka.action.KafkaRequestAction.com$github$mnogu$gatling$kafka$action$KafkaRequestAction$$sendR
equest(KafkaRequestAction.scala:56)
```

RabbitMQ

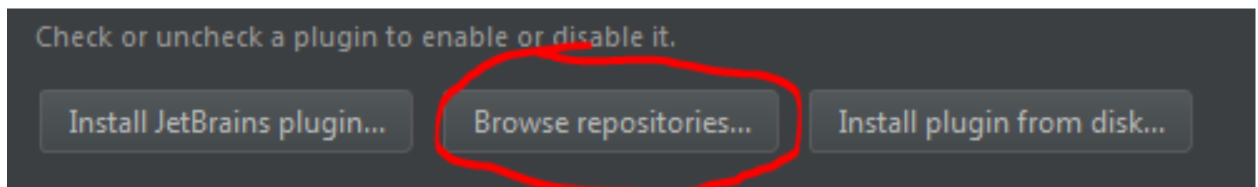
- Dependency Manager: Gradle
- IDE: IntelliJ
- Language: Erlang
- Gatling version required for the plugin Gatling-2.0.0-M3a.
- Similar issues were faced in implementing, Gatling's plugin of RabbitMQ server in addition to the inexperience in Erlang programming language.

SonarLint

SonarLint is an IDE extension that helps you detect and fix quality issues as you write code. Like a spell checker, SonarLint squiggles flaws.

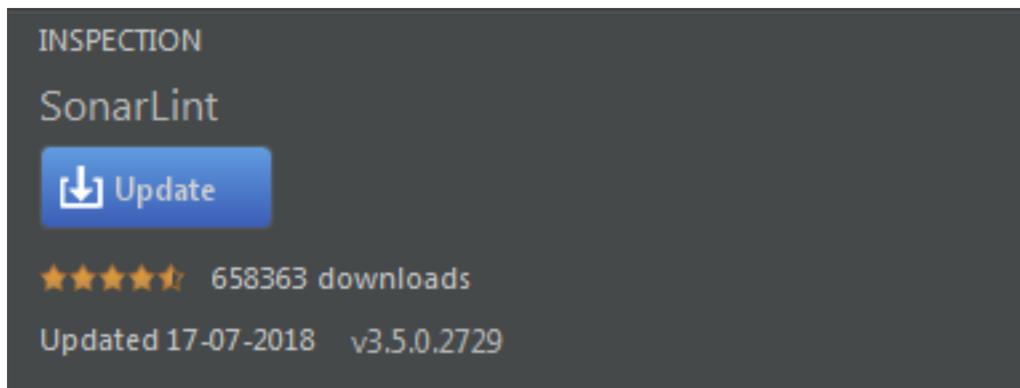
Installation

1. Click on Settings > Plugins



○

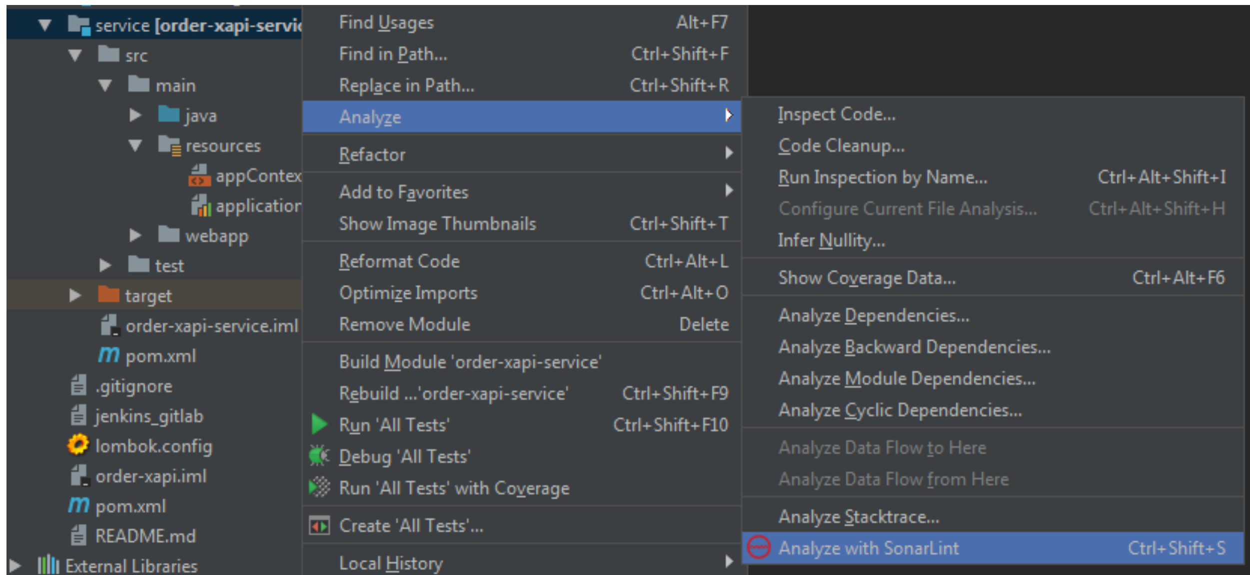
2. Click on Browse Repositories button
3. Type in "SonarLint" and click on the install button



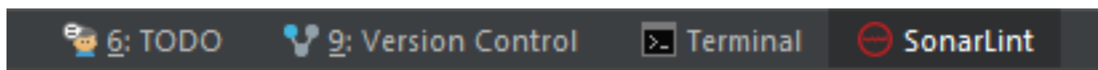
4. Restart your IDE if asked for.

Analyzing source code with SonarLint

With SonarLint, you can analyze the code at codebase level, package level, file level or even a block of code. Select a source folder, package or file or block or code then right click and click on "Analyze with SonarLint" (Ctrl + Alt + S)

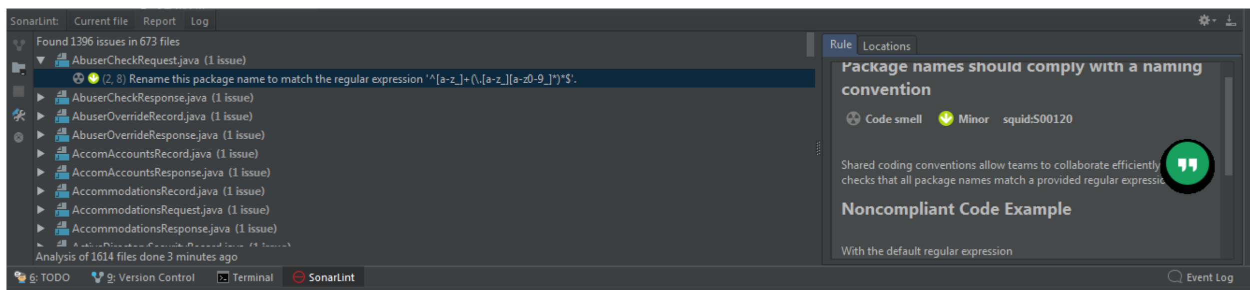


This option will analyze the selected code and generates a report. Once the analysis is complete you would see some results in the SonarLint tab.



You can see the reports for the current file, or complete report by clicking on these tabs in the SonarLint tab.

Select any item in the report to see the rule and location on the right side as shown below.



FindBugs

FindBugs is an open source static code analyzer which detects possible bugs in Java programs. Potential errors are classified in four ranks:

(i) Scariest

(ii) Scary

(iii) Troubling

(iv) Concern.

This is a hint to the developer about their possible impact or severity. FindBugs operates on Java bytecode, rather than source code. The software is distributed as a stand-alone GUI application. There are also plug-ins available for Eclipse, NetBeans, IntelliJ IDEA, Gradle, Hudson, Maven, Bamboo and Jenkins.

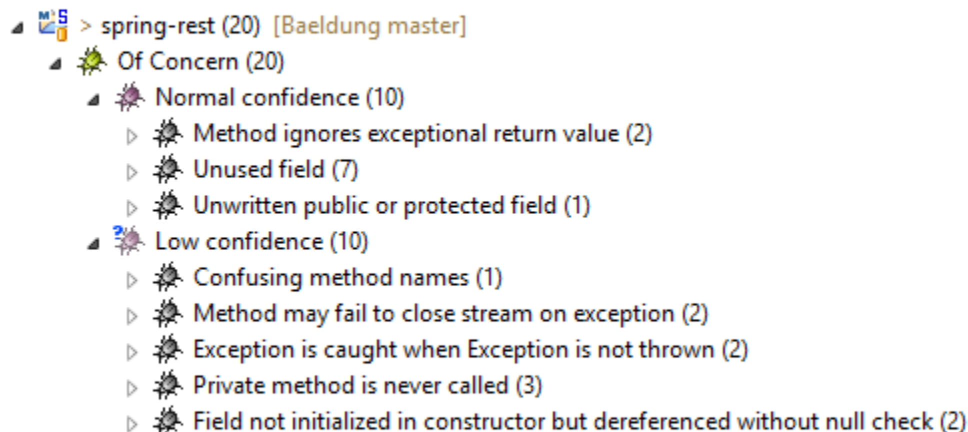
Installation

The steps for installation are:

1. Plugin installation package from the official JetBrains site and extract it to the folder %INSTALLATION_DIRECTORY%/plugins.
2. Restart your IDE and you're good to go.
3. Alternatively, you can navigate to Settings -> Plugins and search all repositories for FindBugs plugin.
4. To make sure that the FindBugs plugin is properly installed, check for the option labeled "Analyze project code" under Analyze -> FindBugs.

Reports Browsing

In order to launch static analysis in IDEA, click on "Analyze project code", under Analyze -> FindBugs, then look for the FindBugs-IDEA panel to inspect the results:



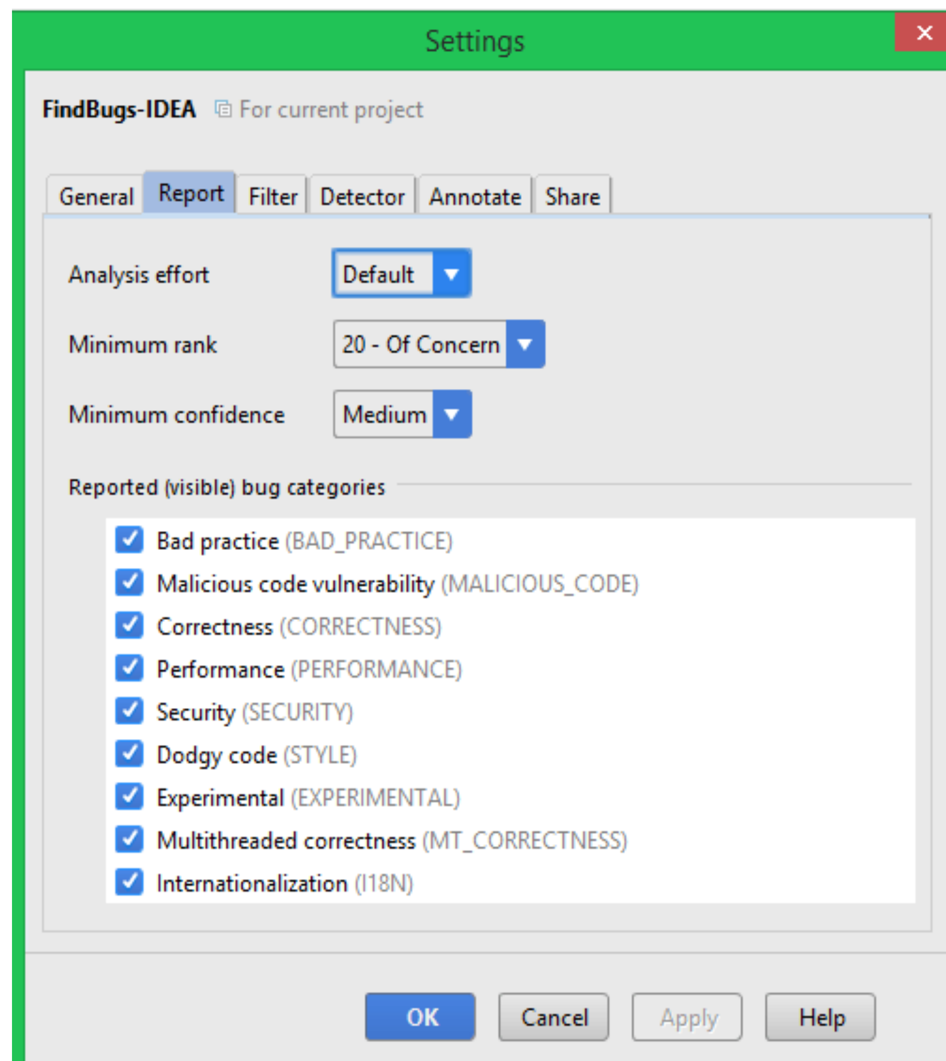
You can use the second column of commands on the left side of the screenshot, to group defects using different factors:

1. Group by a bug category.
2. Group by a class.
3. Group by a package.
4. Group by a bug rank.

It is also possible to export the reports in XML/HTML format, by clicking the "export" button in the fourth column of commands.

Configuration

The FindBugs plugin preferences pages inside IDEA is pretty self-explanatory:



Results

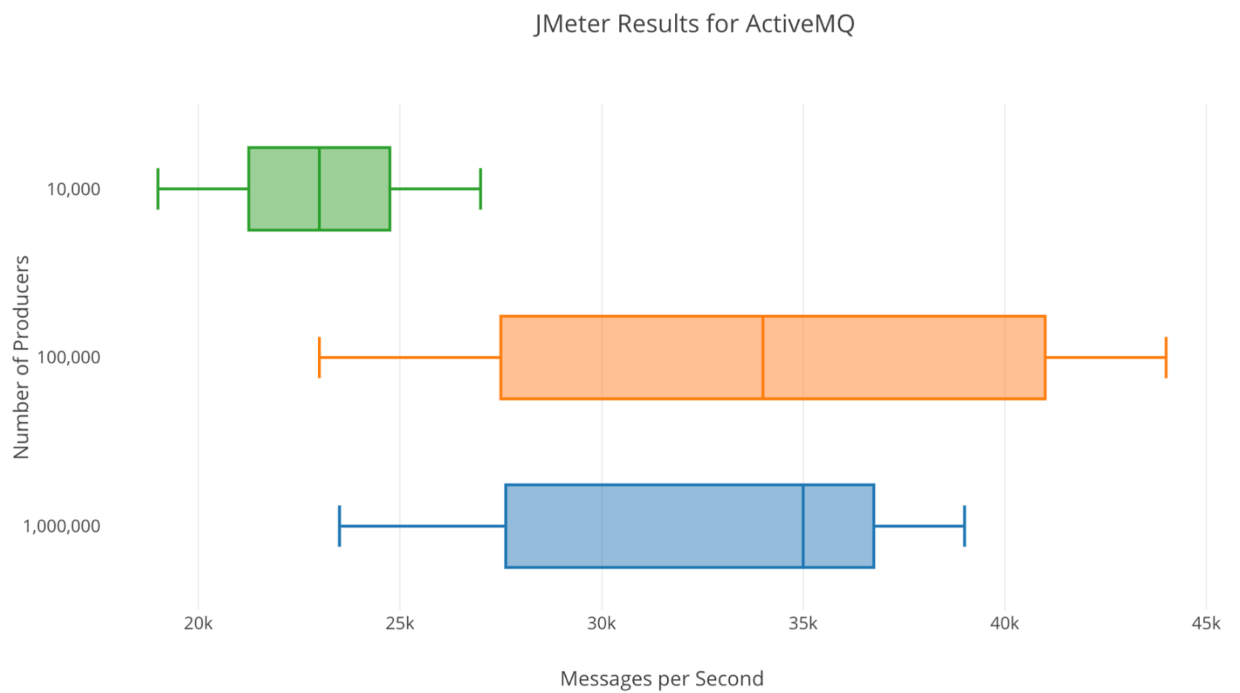
Performance Analysis using Apache JMeter

Throughput

Throughput is the rate of successful message delivery over a communication channel.

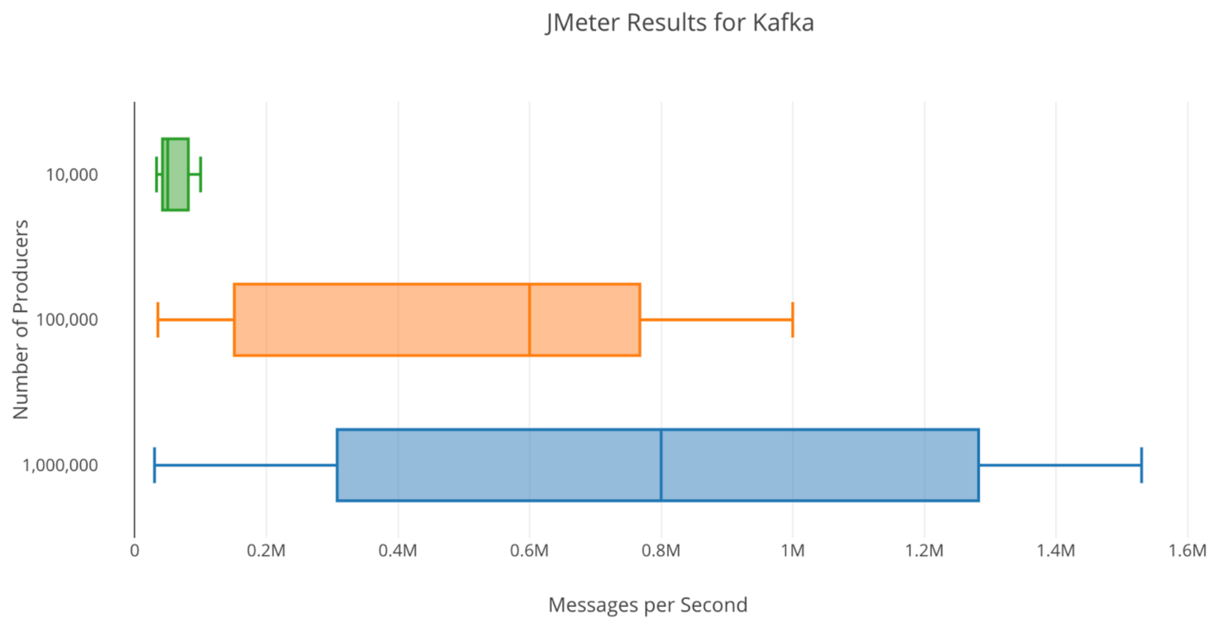
Comparison of Number of Samples vs Throughput

1. Apache ActiveMQ

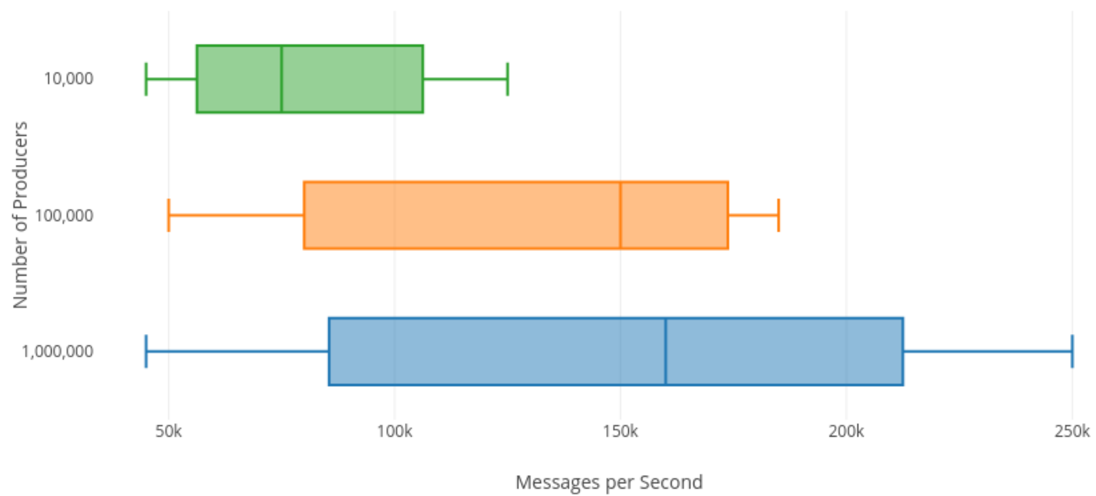


https://github.com/rayandasoriya/Message_Broker_Analysis

2. Apache Kafka

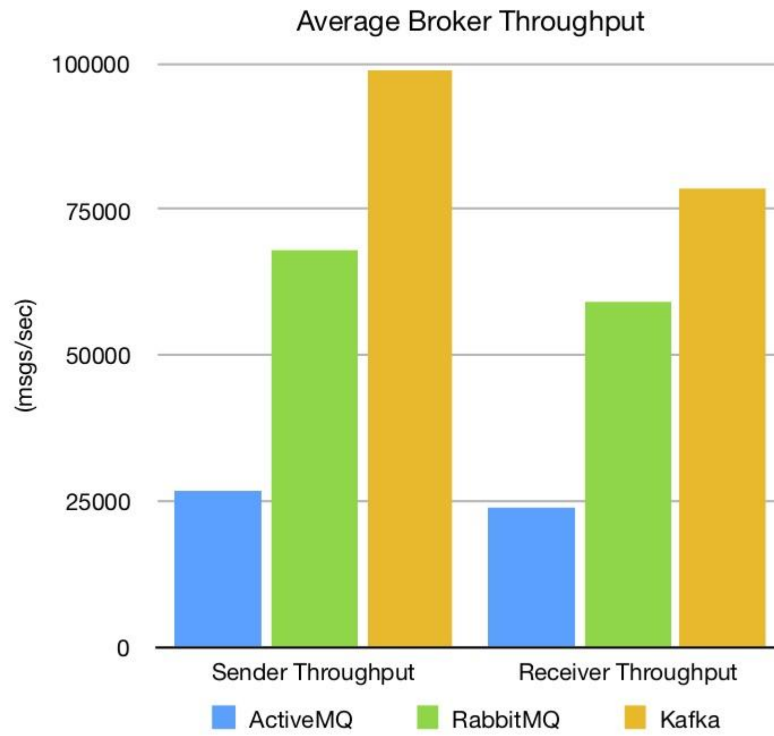


3. RabbitMQ



https://github.com/rayandasoriya/Message_Broker_Analysis

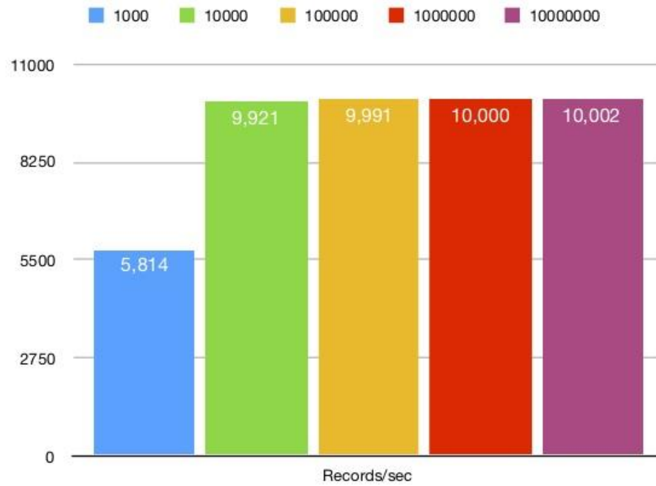
Comparison of Average Broker Throughput

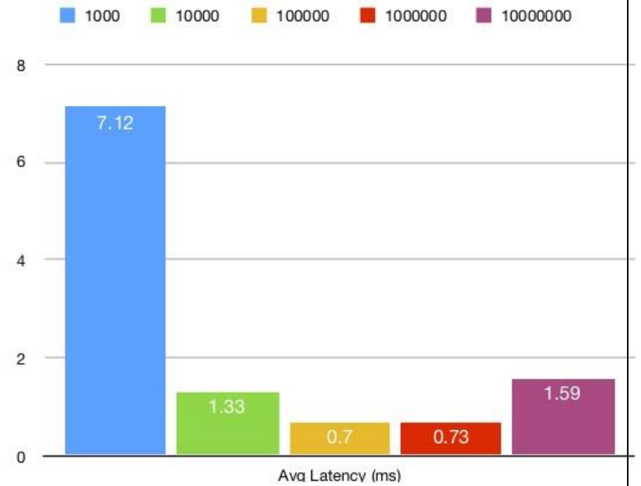


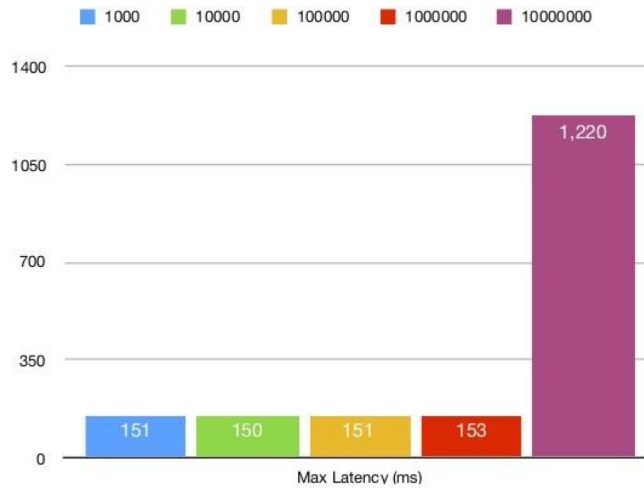
https://github.com/rayandasoriya/Message_Broker_Analysis

Detailed Analysis of Throughput of Apache Kafka

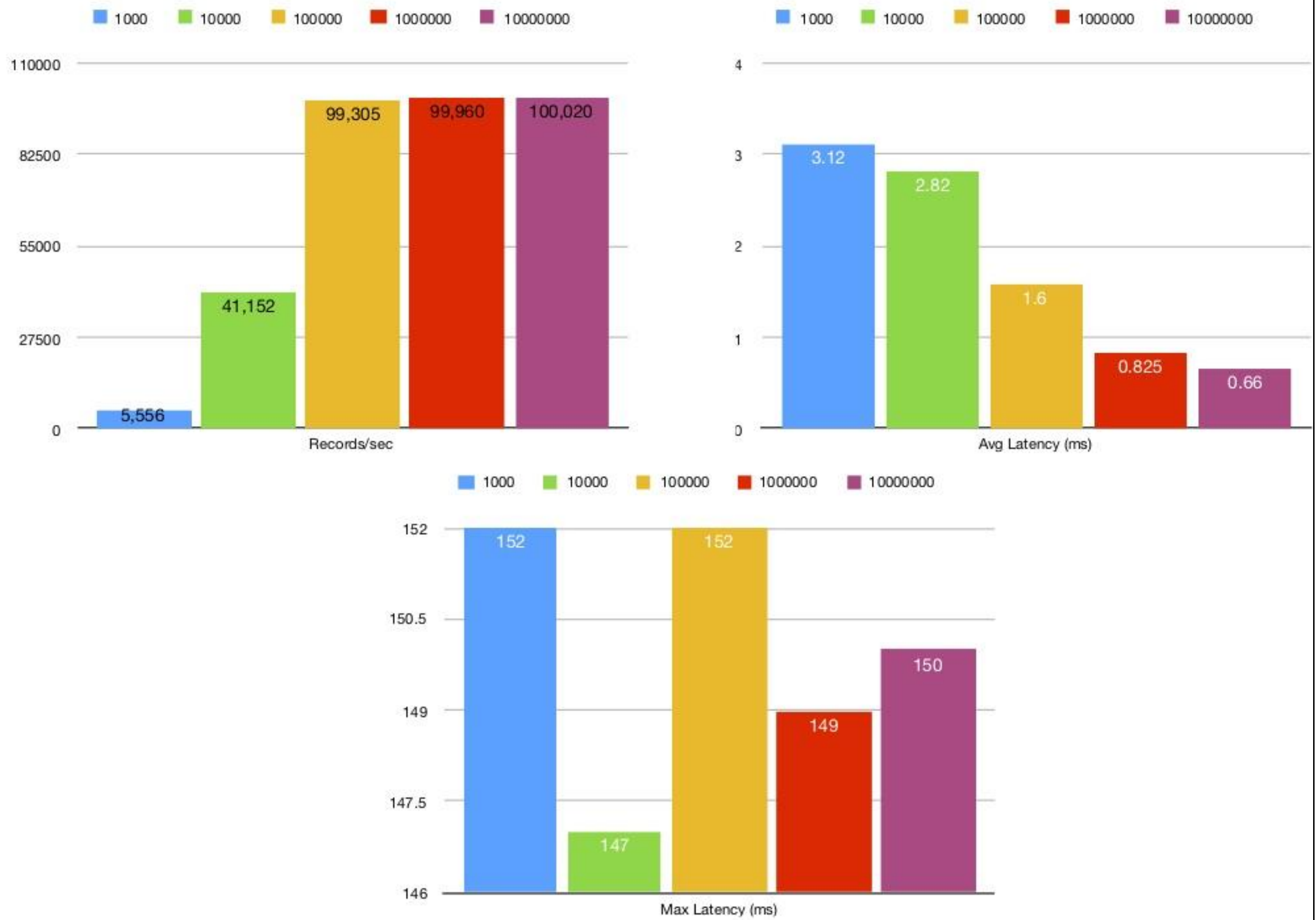
- For 10,000 messages per second



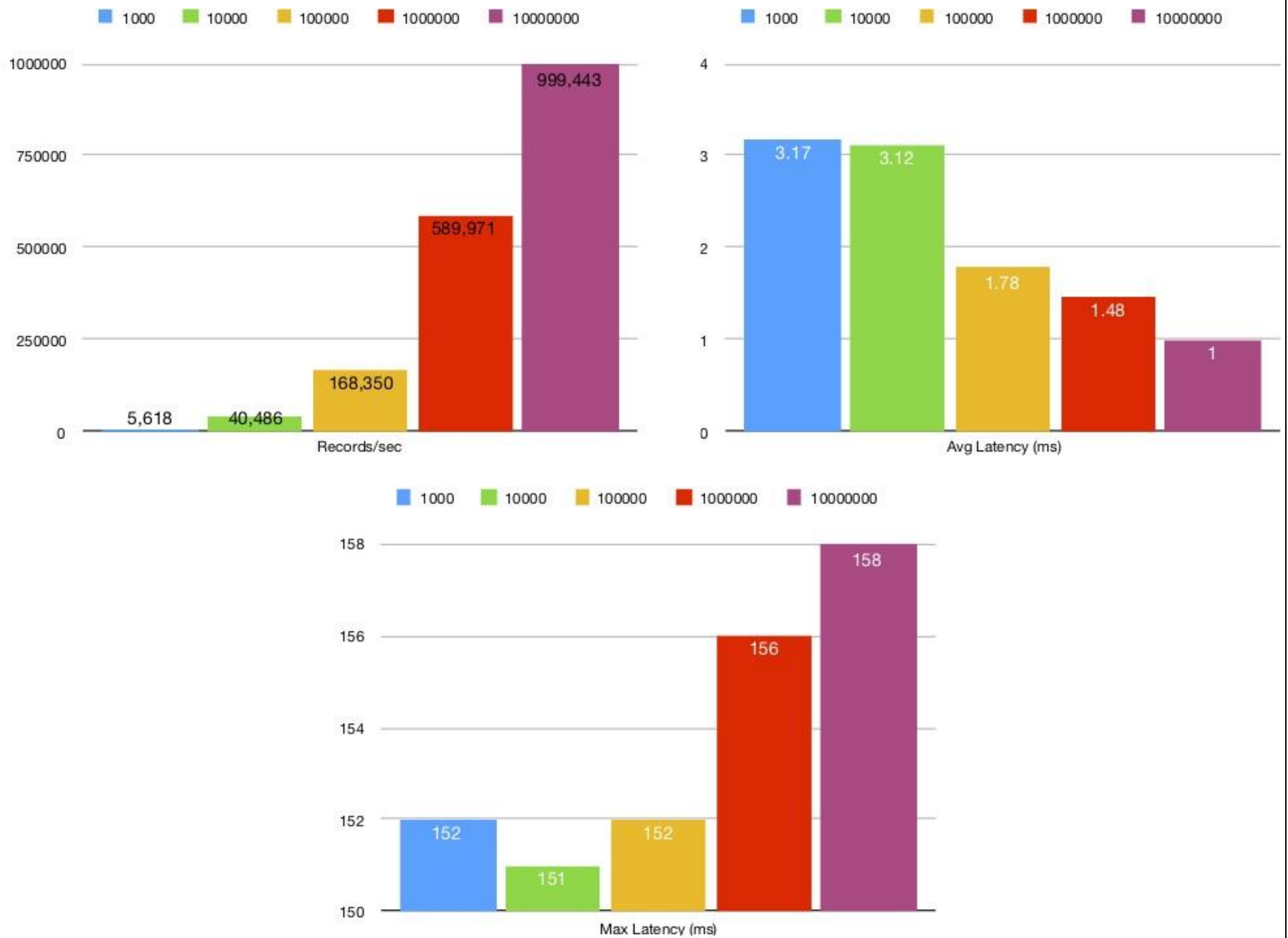




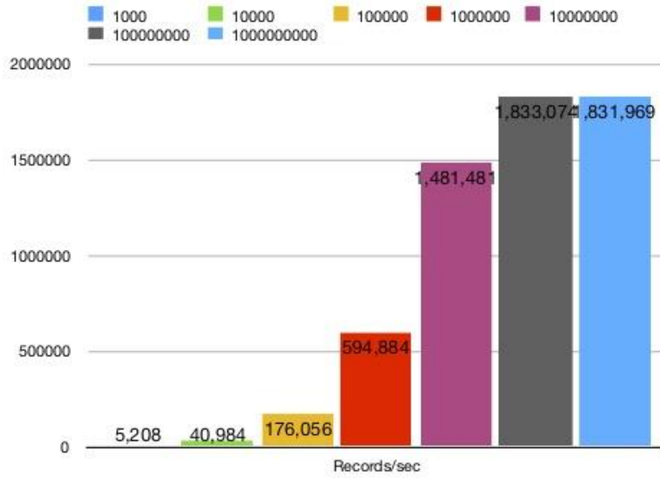
- For 100,000 messages per second

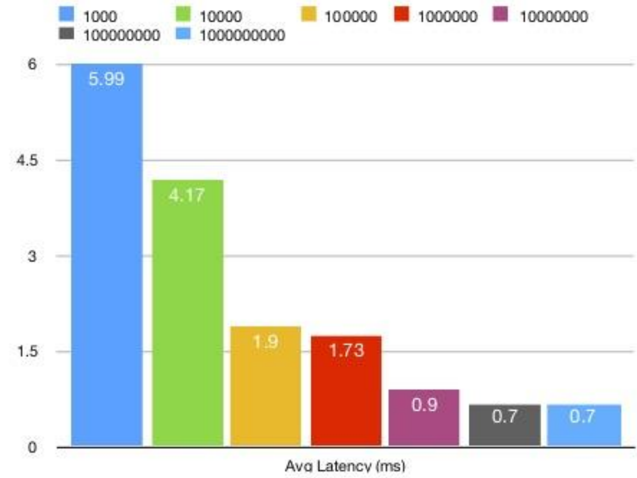


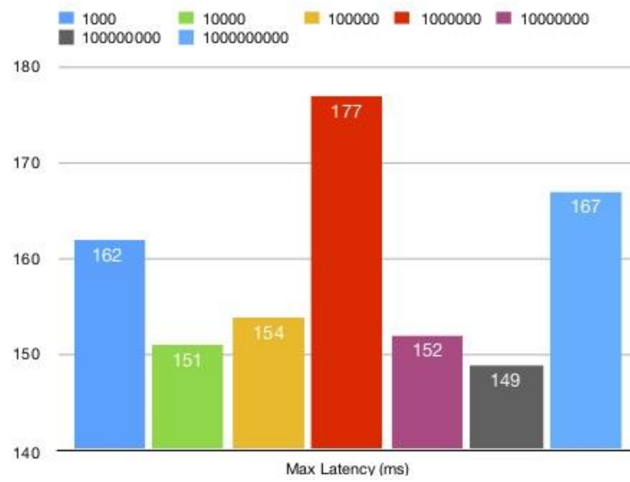
- For 1,000,000 messages per second



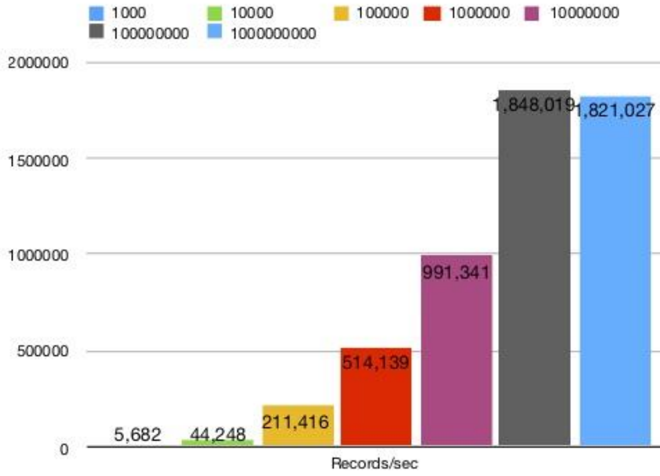
- For 10,000,000 messages per second

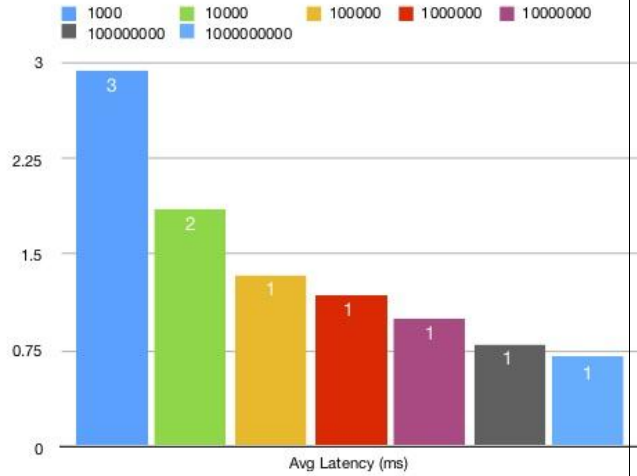


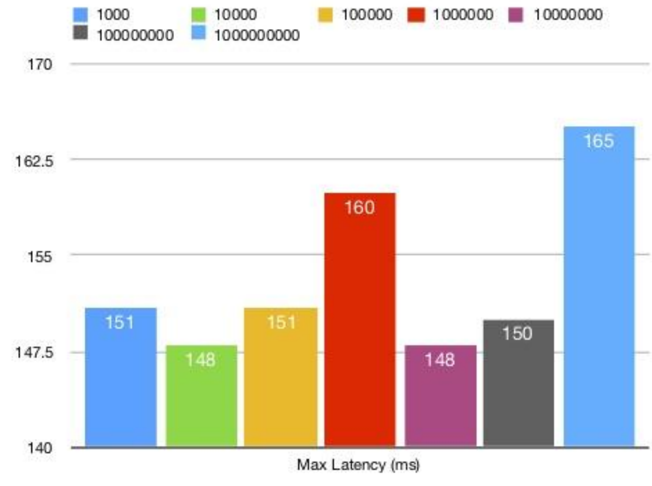




- For 100,000,000 messages per second

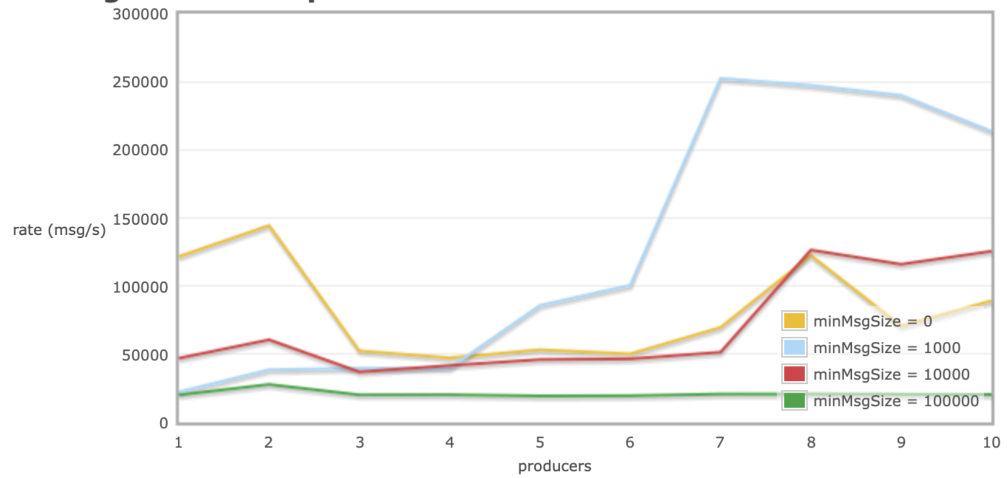


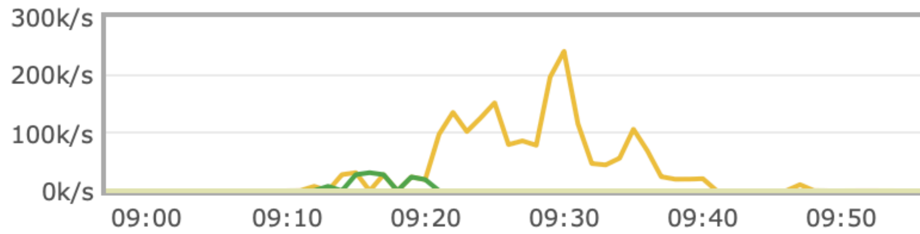




https://github.com/rayandasoriya/Message_Broker_Analysis

Detailed Analysis of Throughput for RabbitMQ

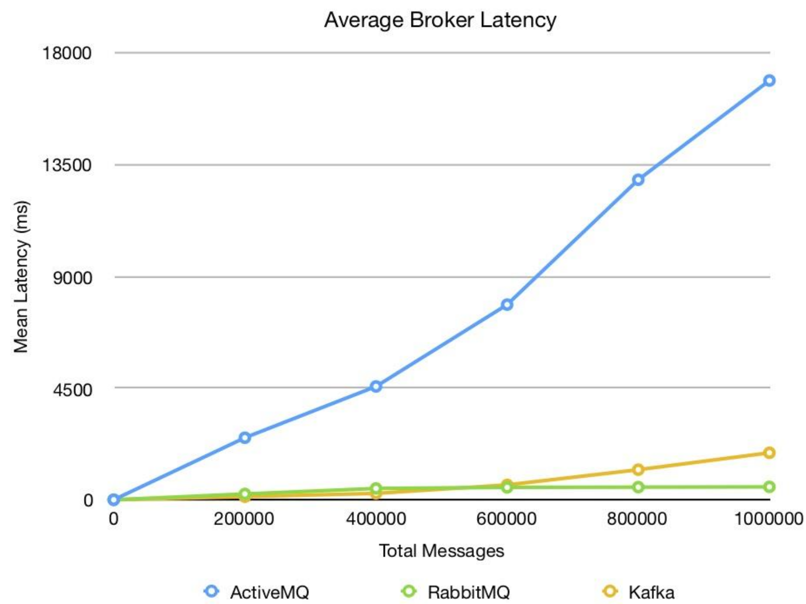




https://github.com/rayandasoriya/Message_Broker_Analysis

Latency

Latency is a time interval between the stimulation and response, or, from a more general point of view, a time delay between the cause and the effect of some physical change in the system being observed.



https://github.com/rayandasoriya/Message_Broker_Analysis

SonarLint

While running an analysis, SonarLint raises an issue every time a piece of code breaks a coding rule. The set of coding rules is defined through the associated quality profile for each language in the project.

Each issue has one of five severities:

1. **Blocker:** Bug with a high probability to impact the behavior of the application in production. Eg. memory leak, unclosed JDBC connection. The code MUST be immediately fixed.
2. **Critical:** Either a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw. Eg. empty catch block, SQL injection. The code MUST be immediately reviewed.
3. **Major:** Quality flaw which can highly impact the developer productivity. Eg. uncovered piece of code, duplicated blocks, unused parameters.
4. **Minor:** Quality flaw which can slightly impact the developer productivity. Eg. lines should not be too long, "switch" statements should have at least 3 cases.
5. **Info:** Neither a bug nor a quality flaw, just a finding.

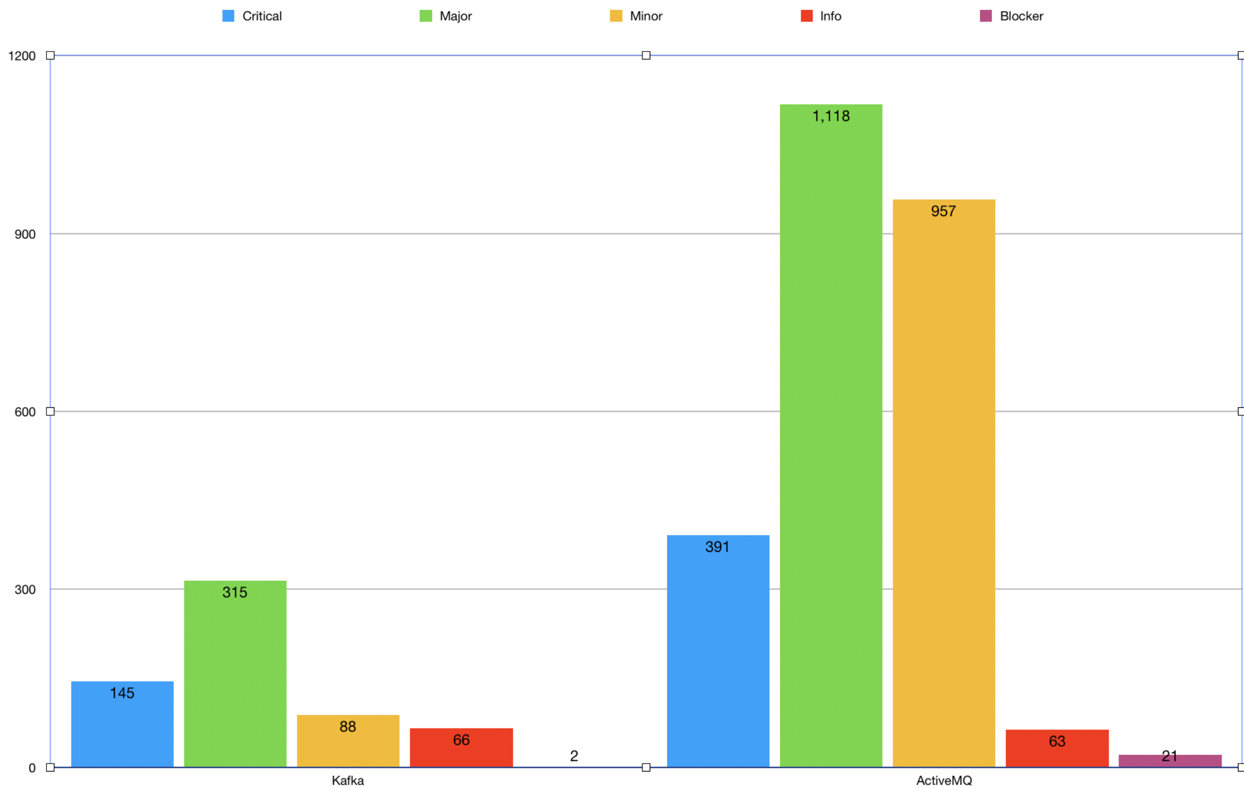
Apache Kafka SonarLint

File Name	Types of Bug				
	Critical	Major	Minor	Info	Blocker
org.apache.kafka.clients.admin	2	12	2	1	0
org.apache.kafka.clients.consumer	24	44	7	45	1
org.apache.kafka.clients	4	17	6	0	0
org.apache.kafka.clients.producer	21	37	16	1	0
org.apache.kafka.connect	94	205	57	19	1

ActiveMQ SonarLint

File Name	Types of Bug				
	Critical	Major	Minor	Info	Blocker
org.apache.activemq.amqp	21	63	68	5	3
org.apache.activemq.broker	295	753	687	51	12
org.apache.activemq.console	51	162	98	6	1
org.apache.activemq.transport.mqtt	9	37	40	0	3
org.apache.activemq.transport.http	15	103	64	1	2

Combined Analysis



https://github.com/rayandasoriya/Message_Broker_Analysis

FindBugs

FindBugs divide defects in many categories:

- Correctness – gathers general bugs, e.g. infinite loops, inappropriate use of *equals()*, etc
- Bad practice, e.g. exceptions handling, opened streams, Strings comparison, etc
- Performance, e.g. idle objects
- Multithreaded correctness – gathers synchronization inconsistencies and various problems in a multi-threaded environment
- Internationalization – gathers problems related to encoding and application's internationalization
- Malicious code vulnerability – gathers vulnerabilities in code, e.g. code snippets that can be exploited by potential attackers
- Security – gathers security holes related to specific protocols or SQL injections
- Dodgy – gathers code smells, e.g. useless comparisons, null checks, unused variables, etc

FindBugs ActiveMQ Broker

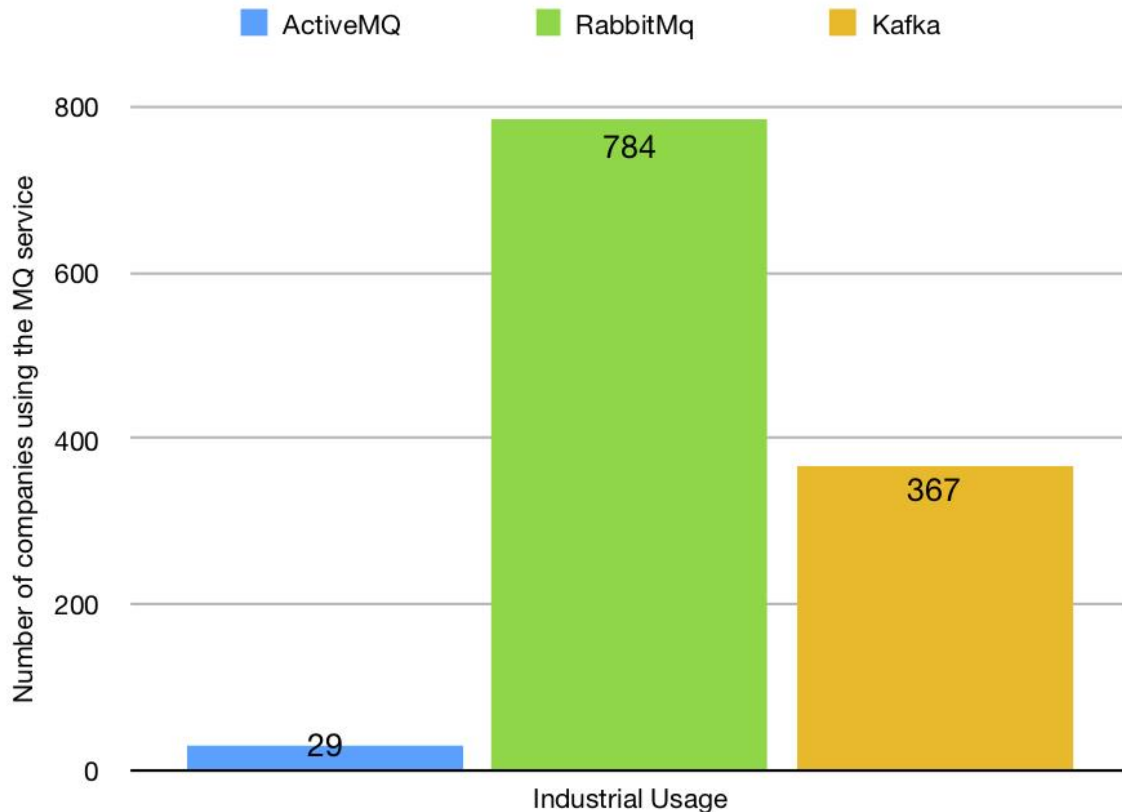
Classes	Bugs	Errors	Missing Classes
539	175	0	0

FindBugs ActiveMQ Core

Classes	Bugs	Errors	Missing Classes
2113	659	0	0

Community Insights

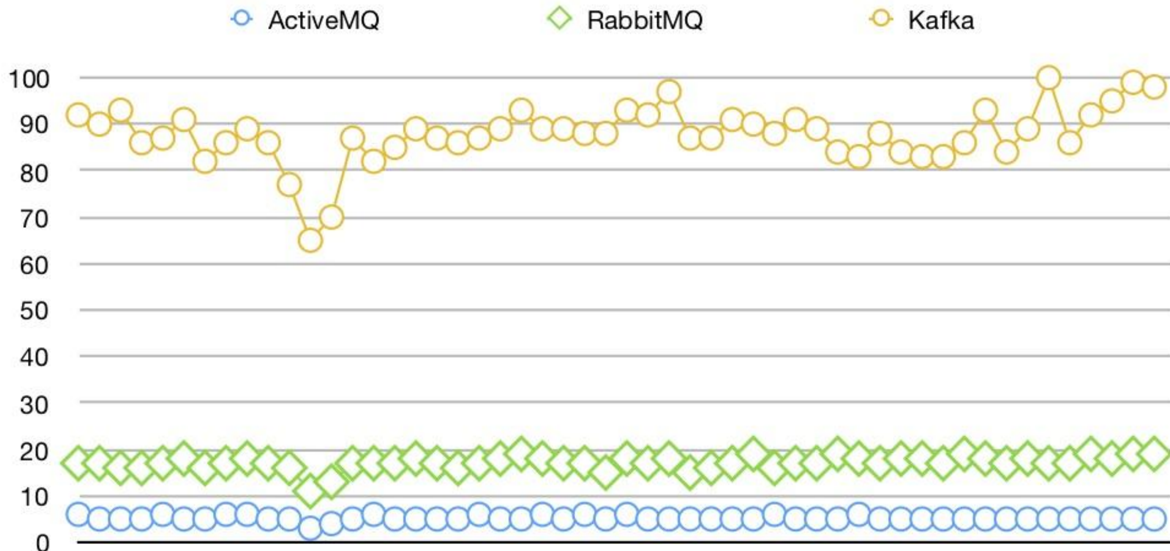
Industrial Usage



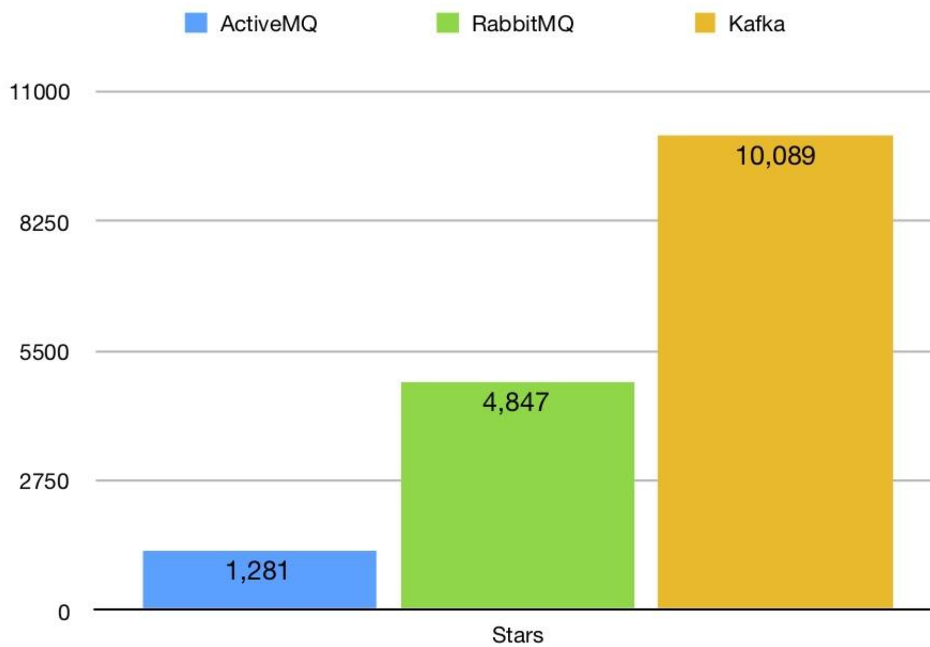
- RabbitMQ is the most popular in the industry, despite Kafka having better performance.
    - This can be because, Kafka was late to the market, and by then RabbitMQ had already taken over the market share from ActiveMQ
        - This can be because a majority of the companies that were previously using ActiveMQ found it very complex.
    - The switching costs associated to RabbitMQ are very low
        - It is simple, flexible, and has several tool integrations available

Popularity in Search

- Looking at the google internet search for the topics directly related to RabbitMQ, Kafka, and ActiveMQ, it can clearly be seen that in the past year, the most popular message queueing service has been Kafka.



- This fact can be further supported by the fact that Kafka has the highest number of stars (amongst the three) on GitHub, translating to very high preference amongst developers.

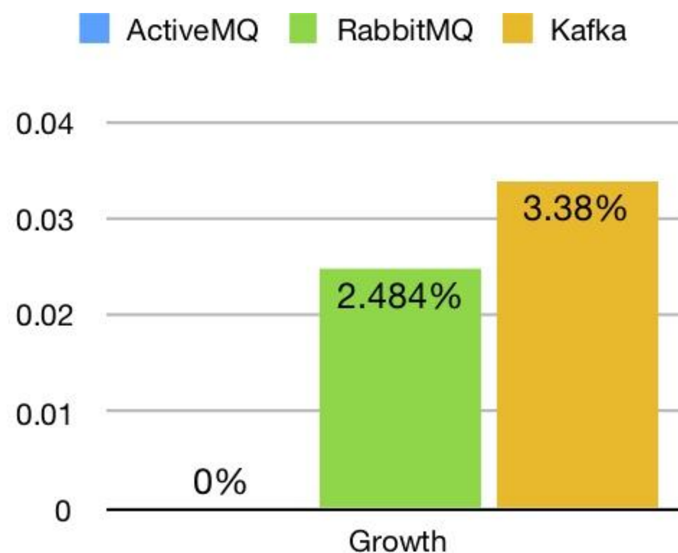


https://github.com/rayandasoriya/Message_Broker_Analysis

- This statistic is important to know, as it can help us with the growth trend towards a particular message queuing broker, which should be higher for Kafka, given its increased popularity amongst developers.
    - For example, a comparison is made between the industrial usage of the message queuing broker, as it was in the beginning of the semester and as it can be seen now.

	RabbitMQ	Kafka	ActiveMQ
Industrial Usage (Beginning)	765	355	29
Industrial Usage (Now)	784	367	29
Growth	2.484%	3.380%	0.000%

- As hypothesized, the growth in the number of companies using Kafka has 1% more increase than in RabbitMQ. This has resulted in more tools being developed for the integration with Kafka.

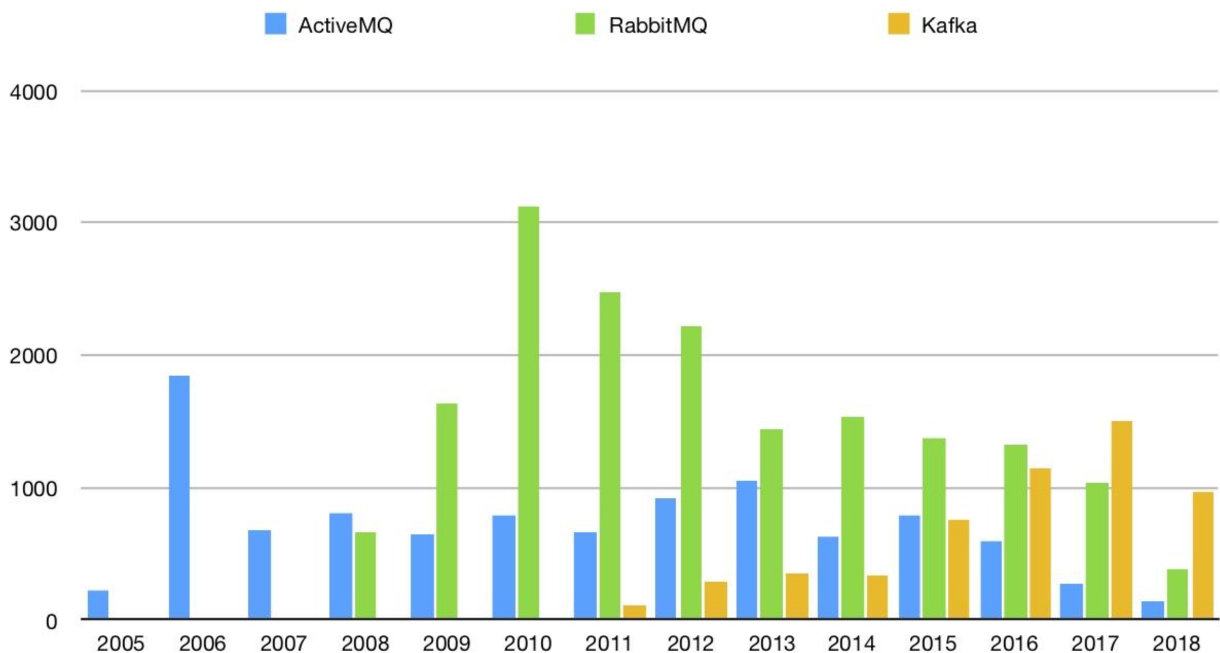


- Having a look at the country-wise statistics, for each of the message queuing services, it was surprisingly a monopoly for Kafka, since in a total of 66 countries, the most popular message queuing broker being searched on google was Kafka.

- It was also interesting to know that, even though there were some countries out of the 66, that did not search for either RabbitMQ or ActiveMQ, — like Albania, and Estonia respectively, in contrast Kafka was searched by all of the 66 countries.

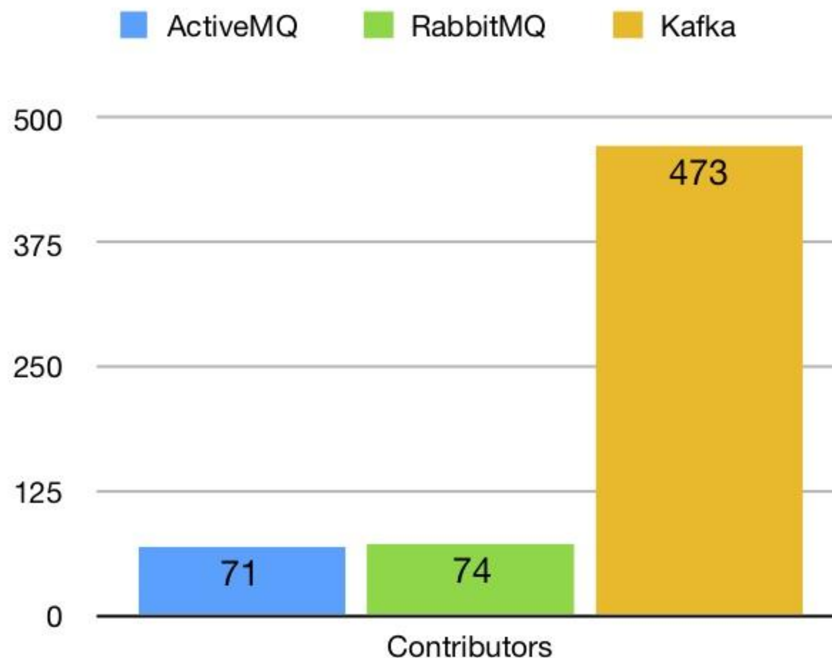
Community Statistics

Commits per year

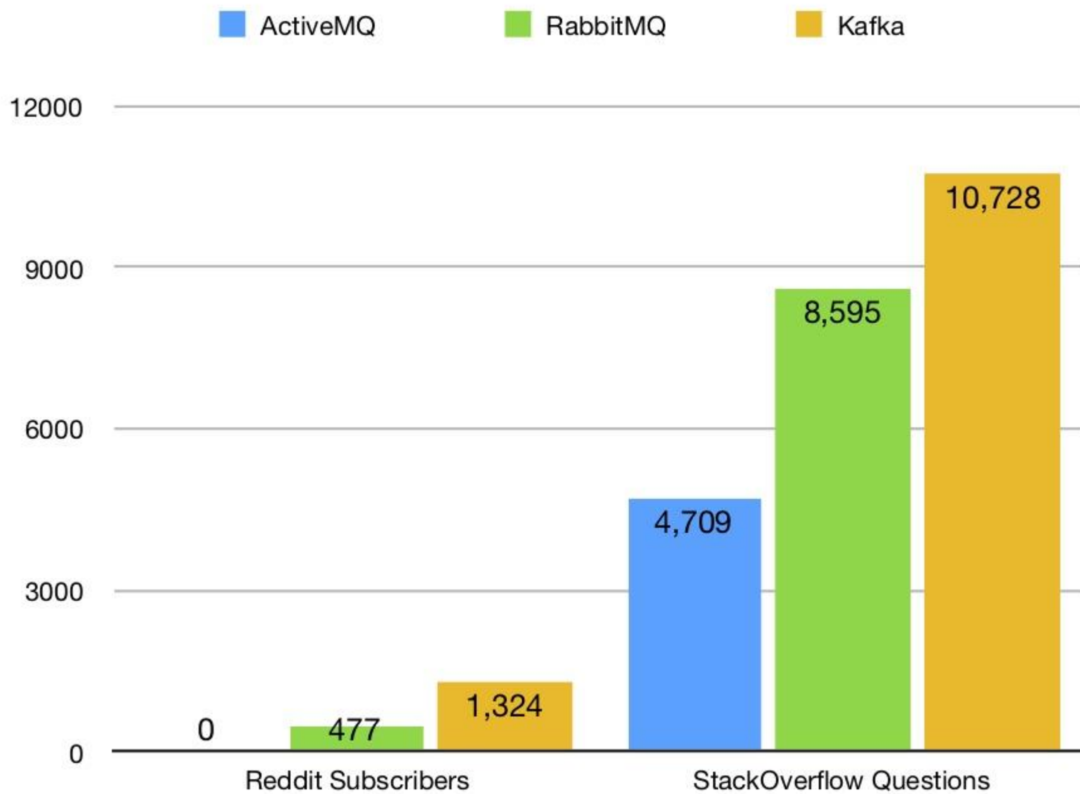


- Based on commits per year for each of the message queuing services, it can be seen that the community for ActiveMQ is becoming less and less active over the years, being the most active in the year 2006, and the least active being this year (2018), which could be correlated with the decline in popularity of the broker amongst developers and organizations alike.
- RabbitMQ's community seemed to be the most active during the years 2009 through 2016. After which it was swiftly taken over by Kafka.
  - It is interesting to know that the number of commits per year for Kafka increased at an average rate of *61%* from 2011 to 2017
  - For the same period, RabbitMQ had a growth rate *-12%*, with ActiveMQ having a growth rate of *-6%*.
  - These results further concretely help us understand the reason for growing popularity amongst developers and organizations and their tendency of moving towards Kafka, thereby resulting in a slightly higher growth rate in adoption.

- Talking about other aspects of community, the retention of contributors very low for RabbitMQ and Kafka where only three and four top 10 contributors respectively still play an important role in the community. In contrast, Kafka has eight of the top 10 contributors still working on the project.
    - It could mean that the community is really helpful in case of Kafka, and that developers are willing to work more on Kafka than other Message brokers.
    - Also, the number of contributors for each of the brokers is a clear indicator of how well received the Kafka community is.



- This could also be a result of the actively accepting pull requests by the community. Kafka community is making sure to include as many developers as possible to grow the community.



   - The community support is somewhat reflected in other aspects of community apart from GitHub. For example, relatively speaking, Kafka has much better support on Stack Overflow and Reddit than RabbitMQ and ActiveMQ.

- ActiveMQ doesn't even have a dedicated reddit channel, which is just a discussion topic in the java channel

Challenges

- The extension was built on Gatling 2.2 and the current version is 3.0 series. Moreover, the Apache Kafka server needed for the plugin to build is 2.10-0.10. We were able to start the latest 2.12 release of Apache Kafka. Same was the case with RabbitMQ
- The version compatibility between SBT, Java, Scala and the producer API was extremely tedious, and it was quite cumbersome.

Conclusion

RabbitMQ is currently the most favored amongst the industry, but there is a shift in affinity towards Kafka, both from the perspective of developers and the industry adoption. The rate at which Kafka is growing is much higher than RabbitMQ, which in contrast seems to be slowly declining its growth rate. ActiveMQ is the least favored from both the developers and industrial perspective given the low industry adoption and developers retention rate. So, if a new developer wishes to contribute to a community, we would recommend contributing to the Kafka community, because of its high rate of activity, retention, support and overall clarity in the documentation. If, however, a developer wants to start learning about message brokers, Message Oriented Middleware, and its implementation we personally found ActiveMQ to be a good starting point and then transitioning towards Kafka. RabbitMQ would require a higher learning curve if the developer is unfamiliar with Erlang.

According to our understanding, Gatling was not able to find the *"ClockSingleton"* class in current release of Kafka server. We have raised the same issue on GitHub repository of the plugin we were trying to implement but haven't been able to resolve it yet. Moreover, one possible way of implementing both the servers could be hosting an API on the servers and then exposing them using Gatling to test, which we did not implement.

Talking about the performance, Apache Kafka gave the best performance with a very high throughput and a low latency rate. ActiveMQ is preferred over Kafka when traditional enterprise messaging is taken into consideration, however, RabbitMQ does a much better job at throughput, latency and overall community support than ActiveMQ. Kafka, because of its low latency, and very high throughput, fault-tolerance, and its highly distributed architecture is most useful in stream processing, event sourcing, commit log and log aggregation, and traditional messaging. RabbitMQ would be more useful in pub-sub messaging, request-response messaging, and also act as an underlaying layer for IoT applications. Hence, depending on the specific use-case you can choose either RabbitMQ or Kafka.

References

[1] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. 2007. Youtube traffic characterization: a view from the edge. In Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC '07). ACM, New York, NY, USA, 15-28. DOI: https://doi.org/10.1145/1298306.1298310
[2] “Apache ActiveMQ” [Online]. Available: https://activemq.apache.org [Accessed: 26-Nov-2018]
[3] “Apache Kafka.” [Online]. Available: https://kafka.apache.org/ [Accessed: 26-Nov-2018]
[4] “Rabbit MQ” [Online]. Available: http://www.rabbitmq.com [Accessed: 26-Nov-2018]
[5] “JMeter” [Online]. Available: https://jmeter.apache.org [Accessed: 26-Nov-2018]
[6] “Gatling” [Online]. Available: https://gatling.io [Accessed: 26-Nov-2018]
[7] “SonarLint” [Online]. Availale: https://www.sonarlint.org [Accessed: 26-Nov-2018]
[8] “FindBugs” [Online]. Available: http://findbugs.sourceforge.net [Accessed: 26-Nov-2018]
[9] “AMQP plugin for RabbitMQ to use with JMeter” [Online]. Available: https://github.com/jlavallee/JMeter-Rabbit-AMQP [Accessed: 26-Nov-2018]
[10] “Pepper-box Plugin for kafka to use with Jmeter” [Online]. Available: https://github.com/GSLabDev/pepper-box [Accessed: 26-Nov-2018]
[11] “Kafkameter plugin for kafka to use with Jmeter” [Online]. Available: https://github.com/BrightTag/kafkameter [Accessed: 26-Nov-2018]
[12] “Kafka Plugin for Gatling” [Online]. Available: https://github.com/mnogu/gatling-kafka [Accessed: 26-Nov-2018]
[13] “RabbitMQ Plugin for Gatling” [Online]. Available: https://github.com/fhalim/gatling-rabbitmq [Accessed: 26-Nov-2018]
[14] N. Nannoni, “Message-oriented Middleware for Scalable Data Analytics Architectures.” KTH, Skolan for informations-och kommunikationsteknik (ICT), 01-Jan-2015 [Online]. Available: https://www.openaire.eu/search/publication?articleId=od_______260::426cdfd2d497eeac93862aef4960f8
[15] “Different takes on messageingh” [Online]. Available: https://jack-vanlightly.com/blog/2017/12/4/rabbitmq-vs-kafka-part-1-messaging-topologies [Accessed: 26-Nov-2018]

[16] “When to use” [Online]. Available: https://content.pivotal.io/blog/understanding-when-to-use-rabbitmq-or-apache-kafka [Accessed: 26-Nov-2018]