Technology Evaluation

The goal of the Gateway is to be a low latency, high throughput, highly scalable, stateless proxy for the services that make up our new platform. This will allow the incremental migration of services from the current system to smaller, individual services.

Note: In the following paragraphs, if "M", "S", "C" or "W" are used, they denote the prioritisation.

Principles

- Stateless - M
- Non-blocking I/O - M
- Fault tolerant with failover/retries easily implemented (built in or : fallback, fail silent, fail fast) - S
- No single dependency can take down an entire app - S
- Asynchronous processing of requests - M
- High concurrency - Semaphores over thread queuing - S

Constraints

- Use Ticketmaster well known languages - i.e. Java
- Easy to test with automated acceptance tests, i.e. run embedded
- Integrate with Splunk
- Able to send monitoring statistics to OpenTSDB/Metrilyx
- Easy to debug
- Easy to unit test

Technology short list

Candidates

- Ratpack - 0.9.11

- Vertx - 2.1.5
- Undertow from JBoss (undertow.io) - 1.1.2Final
- Grizzly - 2.3.18
- Spring asynch on Spring boot - 1.2.1.RELEASE
- Zuul (potentially via Spring Cloud) - 1.0.28

Mandatory requirements

These are the mandatory requirements that must be met for the framework to go into evaluation. If a technology does not have one of these, it is rejected.

Scoring:

- Yes = 1
- Partial = 0.5
- No = 0

Requirement	Ratpack	Vertx	Undertow	Grizzly	Spring asynch on SpringBoot	Zuul
Open source/free to use	Y	Y	Y	Y	Y	Y
Source code can be downloaded and built successfully	Y	2.1.5: Y, 3.0-SNAPSHOT: N	Y	Y	Y	Y
Non-blocking I/O	Y	Y	Y	Y	Y	P - on its own, no. Can integrate Apache Asych HTTP client.
Readable code (e.g. fluent API)	Y (Groovy version is cleaner)	Y (with mod-rxvertx RxJava extension)	Y	N - abstractions are a bit old school.	Partial - via annotations	Y (Groovy version is cleaner)
(Permits or includes) Dynamically configurable routing	P (will need to be coded)	P (will need to be coded)	P (will need to be coded)	P (will need to be coded)	N (uses annotations. Strictly speaking, is possible using JAX-RS regulary expressions but would be nasty code. Plus have to think about @Consumes and @Produces annotations. Programming paradigm doesn't fit our use case.)	P - simple = monitored directory on servers, central = Cassandra. For other databases will need to code DAO.
Ability to manipulate request/response headers	Y	Y	Y	P (will need to be coded)	Y	Y
(Integration of framework that allows) HTTP connection pooling	Y	Y (native)	Y	Y	Y	Y
(Integration of framework that allows) Circuit breaking	Y - native Hystrix integration	Y	Y	Y	Y	Y
WebSocket support	Y	Y	Y (needs Undertow servlet libraries)	Y	Y	N - maybe in Zuul 2.0 (https://github.com/spring-cloud/spring-cloud-netflix/issues/163)
Optional: Built in MBeans for easy monitoring	Y - uses Codahale	2.x: N, 3.x: Y with extension https://github.com/vert-x3/vertx-metrics (3.x on 3.0 milestone 2)	N	Y - uses GMBAL library	Y	Y
Comments	Uses a beta version of Netty: 4.1.0.Beta4-SNAPSHOT (not in Maven central, on Netty source code branch called 4.1 as of 3rd Feb 2015)	Uses Hazelcast for clustering which we don't need (thankfully). Event bus is the core mechanism by which messages are delivered to modules (verticles) in clusters.	Uses an old version of Netty, 3.6.6 (current is 4.0.25.Final, builds with 3.10.0.Final, not with 4.x). Netty 4 has well defined thread model => more thread safe	Doesn't take advantage of any of the new technologies or approaches. Perhaps in Grizzly 3, however, they have been working that since 9-May-13 with no release date	Programming paradigm doesn't fit with our use case.	Zuul depends on other Netflix services (e.g. Eureka) that focus on the AWS cloud. Further, not Spring Cloud Netflix was suggested as the wrapper (see http://cloud.spring.io/spring-cloud-netflix/spring-cloud-netflix.html), however, it is thought that this technology (currently on 1.0.0 RC2) is too immature and provides probably a little too much automagic capability when starting to work core technologies out of the Netflix camp. It is best to deal with them raw to start with, get a fundamental understanding of how they work, hold up in prod and can be tuned prior to laying an abstraction layer on top.
Future state	* Uses latest library versions * Under active development * Will most likely keep up with changes in Netty and reactive programming	2.1.5 uses Netty 4.0.21. 3.0-SNAPSHOT uses 4.0.24.	* New issue submitted to upgrade Netty * Not sure how quickly this can happen given Undertow is the core engine of Wildfly (= JBoss AS 8)	Not able to find details on what is in Grizzly 3	Well supported. Sping does a lot, though, and is squarely aimed at serving web requests rather than HTTP requests. We need HTTP request routing.	Well supported by Netflix and battle tested. Any advanced functionality deployed within Ticketmaster (i.e. not in AWS cloud) will have to have those aspects re-coded (e.g. Archaeus for configuration, Cassandra for central filter store etc.)
Total score	9.5	8.5	8.5	Rejected (doesn't meet mandatory requirements)	Rejected (doesn't meet mandatory requirements)	Rejected (doesn't meet mandatory requirements)

Integration technologies

- HTTP connection pooling, asynch HTTP client (asynch client uses zero copy technique): http://hc.apache.org/index.html
- Circuit breaking, failover, fallback, failsilent,
  Hystrix: https://github.com/Netflix/Hystrix
    - Note: Hystrix is currently manages its own threadpools and is not asynch/non-blocking. It will be of version 1.4.0, currently on RC6. https://github.com/Netflix/Hystrix/pull/218

Accepted for evaluation

1. Ratpack
2. Undertow (in second position over Vertx as highest performance in benchmarks)
3. Vertx

Rejected for evaluation

- Grizzly - abstractions and programming paradigm a bit old school and not eloquent. Results in messy code. No reactive implementation.
- Spring asynch on Spring boot - Not all requirements met. Doesn't fit with our programming paradigm.
- Jersey asynch - Not officially reviewed, however, has same issues with dynamic, configurable routing as Spring asynch.
- Zuul - doesn't meet mandatory requirement of Websocket support.

Evaluation matrix

The evaluation will assess at least three technologies against a number of criteria.

How to score:

- Weighting 0, 5, 10 or 15

- Criteria for each framework: 0, 0.5 or 1.0

Criteria	Weighting	Ratpack	Vertx	Undertow
Release version at least 1.0 OR used by high performance companies (e.g. Linkedin/Kafka)	15	0.5 - close to 1.0 but not yet. Not used in production by any major companies.	1.0 - v2.1.5. v3 in active development.	1.0 - 1.1.2.Final available. 1.2 in Beta 9. Is core engine of JBoss Application Server 8 (Wildfly).
Community support	10	0.5 - only Luke Daly answers the forums but seems pretty active. Key man dependency.	1.0 - active Google vertx forum & recent book (http://www.amazon.co.uk/Real-time-Application-Development-using-Vert-x/dp/1782167951)	1.0 - 304 resolved issues, 40 open: https://issues.jboss.org/browse/UNDERTOW/?selectedTab=com.atlassian.jira.jira-projects-plugin:issues-panel
Asynchronous, non-blocking I/O	15	1.0 - also contains an adapter to plug in third party asych APIs (i.e. Context.promise().then()	1.0	1.0
Seamless integration of blocking processes separate from non-blocking I/O	10	1.0 - ctx.blocking().then()	1.0 - can create a separate Verticle that has its own threadpool for blocking operations.	1.0 - http://undertow.io/documentation/core/undertow-handler-guide.html
Runs in Java 8	10	1.0	1.0	1.0
Developer Productivity: Ease of creating automated tests - unit	10	0.5 - a fixture framework exists, but it expects a response within 5 seconds or fails. The timing is not the issue, but the fact it needs a response. This means you have to have written the code up to the response before the fixture will return (i.e. you can't just write the request promise code and test that in isolation from the response (then) action code), making incremental TDD difficult.	0.5 - not necessarily framework specific, unit test should not be testing any vertx code. See below for integration test support. Provides a framework if necessary: http://vertx.io/dev_guide.html and https://github.com/vert-x/testtools, though it is argued that for event driven applications mocking doesn't make sense: https://groups.google.com/forum/#!topic/vertx/GgS5EagrohQ	0.5 - requires PowerMock to effectively test handlers as the HttpServerExchange class passed into the handler is a final class.
Developer Productivity: Ease of creating automated acceptance tests - BDD	10	1.0 - there is an acceptance test fixture, but we don't need it for our BDD set up.	1.0 - comes with TestVerticle for integration tests	1.0 - no unit test framework but not needed. BDD tests treat the spun up server as a whitebox and can be integrated into BDD test cycle.
Simple deployment - no external dependencies except Java (pref single executable jar)	10	1.0	0.5 - can create a fat jar although recommended to install vertx platform	1.0 - executable JAR.
WebSocket support	5	1.0 - http://www.ratpack.io/manual/current/all.html#websockets	1.0 - http://vertx.io/core_manual_java.html#websockets. Example: https://github.com/vert-x/vertx-examples/tree/master/src/raw/java/websockets	1.0 - http://undertow.io/documentation/core/websockets.html
Can integrate Ticketmaster Spring based component libraries	15	1.0	1.0 - provide a @Configuration bridge, e.g. https://git.tm.tmcs/unified-api/unified-work-distributor/blob/vertx-java-spring/src/main/java/com/livenation/tap/unifiedapi/uwd/wiring/HealthcheckConfiguration.java	1.0
Dynamically configurable routing	5	0.5 - this can be coded but it will be unnecessarily complex and error prone. It is better to hard code and take advantage of the fluent API, coupling this with frequent, small deployments.	0.5 - easy to externalise routing into config, can update and undeploy/redeploy verticle. To be discussed - dynamic routing vs small, fast, 'boring' releases?	0.5 - this can be coded but it will be unnecessarily complex and error prone. It is better to hard code and take advantage of the fluent API, coupling this with frequent, small deployments.
Forum	5	1.0 - good, although Luke Daly seems to be the only one responding.	1.0 - fairly active with nearly all posts getting replies (https://groups.google.com/forum/#!forum/vertx)	1.0 - mailing list is actively answered by teh authors: http://lists.jboss.org/pipermail/undertow-dev/2015-January/thread.html
Open Source issue submission and review	10	1.0 - Seems decent. As of 5/Feb/15, 362 issues closed, 75 open.	1.0 - 500+ closed, 8 open.	1.0 - active mailing list which the contributors respond to in a timely fashion, e.g. http://lists.jboss.org/pipermail/undertow-dev/2015-January/thread.html
Stack overflow presence	5	0 - 5 questions under tag "ratpack" to date.	0.5 - over 200 posts although Google is preferred forum	0.5 - Only 320 posts, but the mailing list is preferred and appears more active.
Google trends	5	0 - Not enough search volume to show a trend : http://www.google.com/trends/explore#q=ratpack%20java	1.0 - increasing volume from mid 2007.	0.5 - some volume from June 2014
Future composition - avoid callback hell	10	1.0 - fluent API prevents this, i.e. context.promise().then().promise().then()	0 - chaining of Handlers so could get into a bit of a mess if need to compose	0 - chaining of Handlers so could get into a bit of a mess if need to compose
Performance - no more than 50ms added onto upstream responses at 99th percentile	15	0	1.0	0
Comments		We are definitely dealing with an immature tech. WebSocket support is not mature (see issues 482 and 458) and nobody major is using it in prod. There is definitely significant risk here. A more mature tech, if it offers the same advantages as Ratpack, would probably be wiser.	Vertx is a platform/server that you deploy modules and verticles to. Has features that we are likely not to need now (clustering, event bus), but core would suit our purposes. Appears mature and well-adopted. v3 is in beta and looks to remove to module architecture found in v2. v3 docs being written but looks easier to understand - https://vertx.ci.cloudbees.com/view/vert.x-3/job/vert.x3-website/ws/target/site/docs/manual.html . Further, Apex allows us to write code the way we want to: https://github.com/vert-x3/vertx-apex/blob/master/src/main/asciidoc/java/index.adoc	Overall it was reasonably easy to work with. Could have done with a better, more fluent API. Testing the chaining and routing of requests was accomplished quickly but doesn't make for easy to read test code.
Scores	165	122.5	145	127.5

Performance results

- SLA = no more than 50ms slower than downstream server response at the 99th percentile. e.g. if a downstream service takes 50ms, the Gateway cannot take longer than 100ms at the 99th percentile.
- Test framework: https://github.com/wg/wrk/wiki/Installing-Wrk-on-Linux (installed on same machine as app under test)
- Hardware: AWS m3.xlarge with 4 cores all in same AWS zone
    - One instance running custom downstream mock service
    - One instance running app under test and wrk
- Call: wrk --latency -c 1000 -t 1 -d 50s --timeout 17s http://localhost:8080/ud/wait/15000
- 1 thread, 1000 connections, looped call for 50 seconds. Calls app under test which proxies onto a test server that waits 15000ms before replying. Run a minimum of 3 times before times are recorded and 3 runs recorded.

Results

Vertx with 1 verticle.

Framework	Avg	Stdev	Max	50%	75%	90%	99%
Vertx (1v)	15.03s	5.91ms	15.04s	15.03s	15.04s	15.04s	15.04s
Vertx (1v)	15.03s	5.91ms	15.04s	15.03s	15.04s	15.04s	15.04s
Vertx (1v)	15.10s	197.88ms	15.83s	15.03s	15.03s	15.22s	15.83s

Vertx with 4 verticles, taking advantage of all cores.

Framework	Avg	Stdev	Max	50%	75%	90%	99%
Vertx (4v)	15.02s	3.92ms	15.03s	15.02s	15.03s	15.03s	15.03s
Vertx (4v)	15.02s	4.35ms	15.03s	15.02s	15.03s	15.03s	15.03s
Vertx (4v)	15.03s	10.99ms	15.07s	15.03s	15.04s	15.04s	15.06s

Framework	Avg	Stdev	Max	50%	75%	90%	99%
Ratpack	15.06s	174.38ms	16.05s	15.03s	15.03s	15.04s	16.04s
Ratpack	15.02s	220.60ms	16.04s	15.02s	15.03s	15.05s	16.04s
Ratpack	15.02s	330.52ms	16.24s	15.02s	15.03s	16.02s	16.24s

Framework	Avg	Stdev	Max	50%	75%	90%	99%
Undertow	15.14m	277.60ms	16.04s	15.02s	15.03s	15.42	16.04
Undertow	15.11m	243.80ms	16.03s	15.02s	15.02s	15.42	16.03
Undertow	15.16m	339.03ms	16.65s	15.02s	15.02s	15.83	16.65

- Extra test to saturate over a period of time
- wrk --latency -c 1000 -t 1 -d 5m --timeout 17s http://localhost:8080/ud/wait/150

Framework	Avg	Stdev	Max	50%	75%	90%	99%	Requests
Vertx	170.63ms	3.17ms	284ms	170ms	170ms	171ms	182ms	1,757,517
Ratpack	184.50ms	95.96ms	3.17s	170ms	176ms	181ms	386ms	1,628,271
Undertow	170.36ms	8.09ms	1.17s	169ms	170ms	171ms	180ms	1,757,202

Conclusion

As can be seen from the above results, Vertx wins out of the three selected candidates. In fact, both Ratpack and Undertow would fail because they do not meet our performance criteria of adding no more than 50ms to an upstream's response as dictated by the standard deviation and the 99th percentile. Vertx is ridiculously fast and, in fact, in recent benchmarks at Ticketmaster only beaten by a service written in Go. Vertx 3 is coming out in two months that has a more reactive style of programming and a far less opinionated architecture. Redhat supports it and they are recruiting more developers to work on it. Given its maturity and knowledge of it among the tech community (c.f. Google Trend for Vertx shows a steady increase since 2007), we are confident Vertx is the right technology for the Gateway. We must make sure we upgrade as soon as 3.1+ of Vertx is released. We may even do a very slow roll out of 3.0 to take earlier advantage of the reactive programming style and better Spring integration via its Apex framework.

References

- How to evaluate web frameworks: http://www.slideshare.net/mraible/comparing-jvm-web-frameworks-february-2014
- Benchmarks: https://www.techempower.com/benchmarks/#section=data-r9&hw=peak&test=json&b=1&s=2&l=3k
- Ratpack execution model: http://blogs.atlassian.com/2013/07/http-client-performance-io/
- Rudimentary HTTP client benchmarks: http://blogs.atlassian.com/2013/07/http-client-performance-io/
- Netty 4 new and noteworthy: http://netty.io/wiki/new-and-noteworthy-in-4.0.html

- Undertow Netty 4 support issue
  filed: https://issues.jboss.org/browse/UNDERTOW-376