ColdFusion 2016 API Manager

Performance Analysis

API Manager – an overview

API Manager is an all new API management solution introduced with Adobe ColdFusion (2016 release). The API Manager can be used as a platform to publish and manage APIs based not only on ColdFusion, but also on other platforms or languages. The API Manager can:

- Authenticate an API via OAuth2, basicAuth, or apiKeys
- Enforce rate and throttle limits on API calls with SLAs
- Manage the life cycle and versioning of an API
- Monitor API usage statistics across applications and user profiles
- Cache API response to reduce load on server

The API Manager includes some additional capabilities, such as:

- Publishing a traditional SOAP based web service as a RESTful API.
- Proxy for traditional web services (SOAP PROXY)

The performance improvement results in the document are for representative purposes only. Adobe has conducted extensive internal tests to produce the results. However, actual improvement may differ for individual applications.

Executive Summary

The API Manager (APIM) is a highly scalable and efficient platform.

In our tests, when using a lightweight web-service, a single node of API Manager served 500 concurrent users at the rate of 22,000 requests per second, with a latency of less than 14 milliseconds. A single node of API Manager, can therefore, serve close to 2 billion requests per day. A cluster of API Manager Nodes can handle an even higher volume of traffic or it can lower the latency.

This whitepaper presents the results of API Manager's performance and scalability testing on this platform. The paper also presents the configuration settings that you can tune to achieve optimal performance.

Broadly, the test results are from a study that was conducted on the following configurations:

- **Single node configuration**: A single instance of API Manager, set up with a single instance of the back-end application server hosting the web-services.
- **Cluster configuration**: A cluster of three API Manager Instances arbitrated by a load balancer, set-up with three application server instances behind another load balancer.

Types of web services tested

This whitepaper discusses the performance of the following types of web services. These web services do not perform any CPU or file I/O intensive operations:

1. **HTTP GET** – A simple RESTful web service that serves pain text data.
2. **HTTP POST** – A service that takes JSON data in the request payload.
3. **SOAP Proxy** – A simple traditional web service that provides document-literal WSDL and takes a SOAP 1.2 message.
4. **SOAP to REST** - A simple traditional web service that provides document literal WSDL and takes a SOAP 1.2 message. The API Manager converts the SOAP service to a RESTful API.

Each web service has been tested with small and large payload size, with different number of concurrent users.

Performance Parameters measured

One of the principal design goals of API Manager is to maintain the lowest possible latency, so that the response time for the end user is largely unaffected due to API being served through this platform.

The tests used a wide range of the following parameters to gather the performance statistics:

1. Payload size
2. Number of concurrent users

The following metrics were collected as a result of the performance tests:

1. **Throughput** – The number of requests the application can handle over time. For example, 5000 requests/second
2. **Average Response Time (ART)** – The time a request is dispatched from the client to the time a response is received by the client. It is measured in milliseconds. For example, 25ms.
3. **Latency** – The difference in ART when a web service call is made directly to the end-point as compared to when it is routed through the API Manager. It is measured in milliseconds.

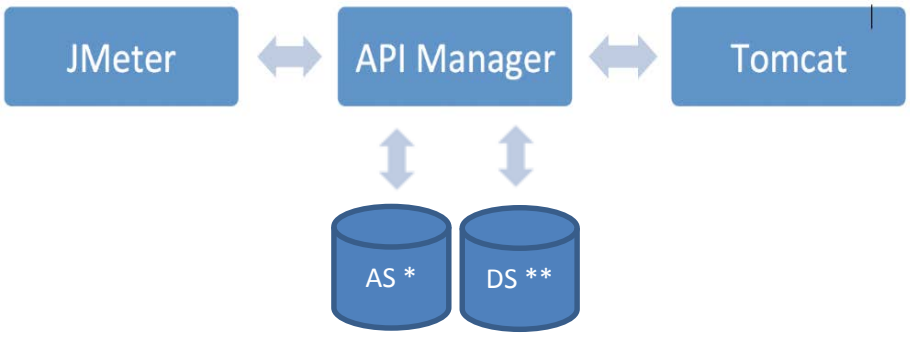
Test configurations

To find how the API Manager performed, we calculated the latency it introduced. We compared the ART of the baseline web service with the ART when APIM is put between the client and the web-service.

The metrics were collected for the following two configurations:

- **The Web-Service Baseline,** where the request is sent from the client directly to the back-end web service.

- **The APIM gateway**, where the load is injected to the back-end web-service through the APIM gateway. The Data Store and the Analytics Server are hosted on separate machines.



** Analytics server*
*** Data Store*

We used Apache Tomcat as the endpoint where the web-services were deployed. Apache JMeter was used to send HTTP requests to the API Manager. The API Manager processes and dispatches the requests to Tomcat, where the services are hosted.

Single Node configuration

HTTP GET API

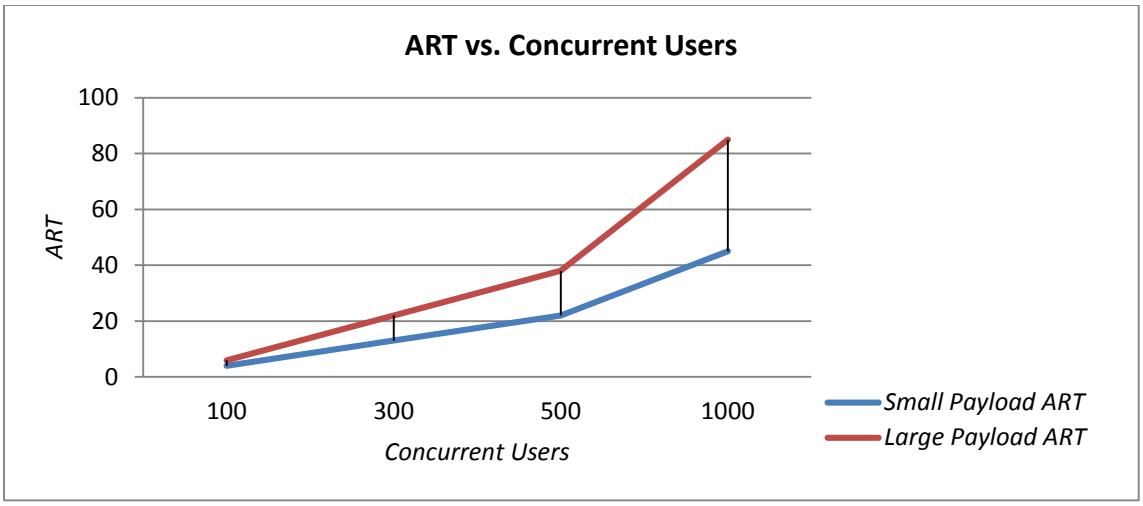
The test results below are from an HTTP GET web service that returns a simple text value. The tables below show the results for a payload size of 102 bytes and a comparatively larger payload of 5KB.

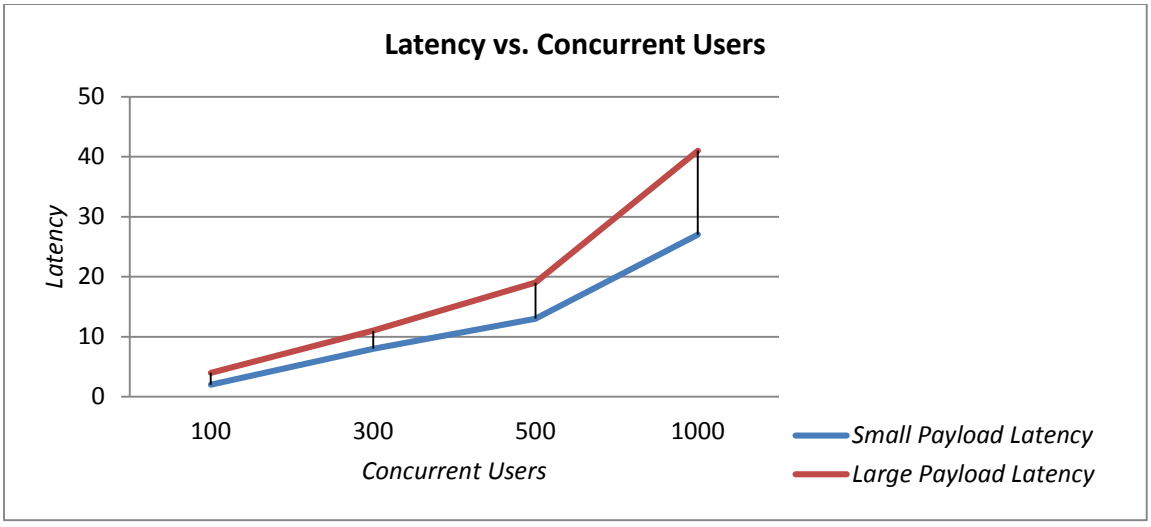
Users	APIM Throughput	ART	APIM Latency
100	22284	4	2
300	22097	13	8
500	21986	22	13
1000	21367	45	27

Performance measurement for GET - small payload (102 bytes)

Users	APIM Throughput	ART	APIM Latency
100	14589	6	4
300	12973	22	11
500	12747	38	18.7
1000	11345	85	40.7

Performance measurement for GET- large payload (5KB)





As seen in the table and the graph, at lower load, latency introduced by API Manager is negligible. Tomcat's ART and the latency introduced by API Manager increases linearly with the number of concurrent users.

HTTP POST API

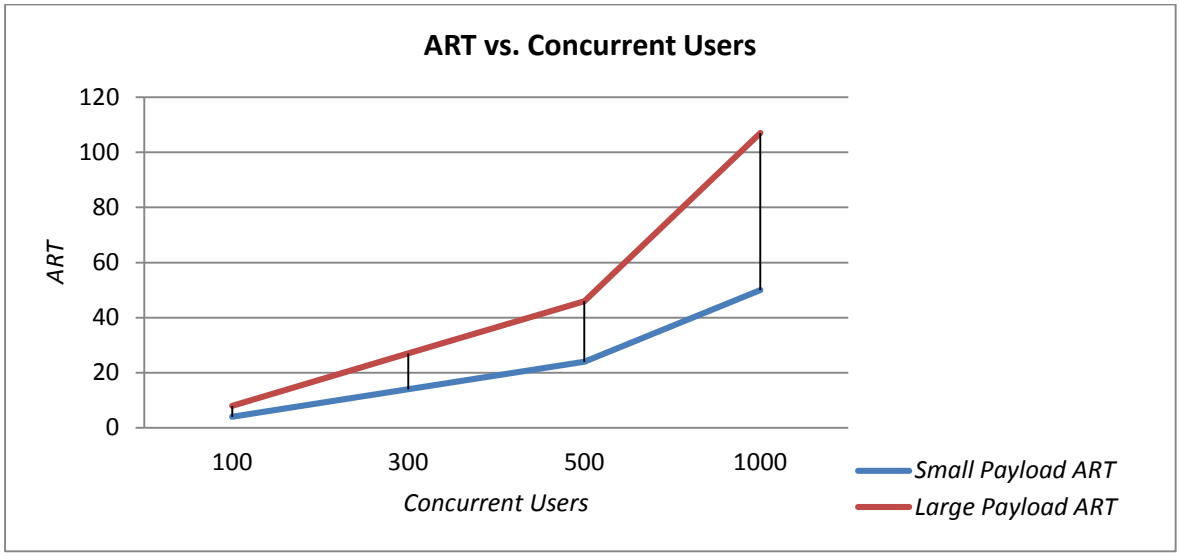
The test results below are from an HTTP POST web-service that accepts JSON data in the form of request payload. The tables below show the results for a payload size of 102 bytes and a comparatively larger payload of 5KB

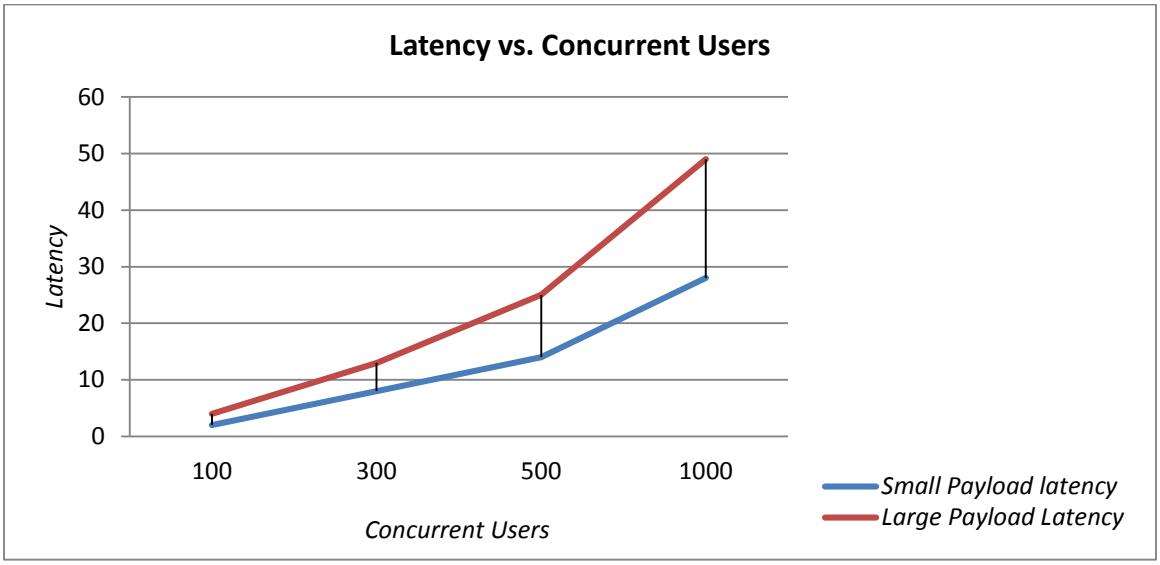
Users	APIM Throughput	ART	APIM Latency
100	20750	4	2
300	20495	14	8
500	20351	24	14
1000	19576	49.7	27.7

Performance measurement for small POST payload (102 bytes)

Users	APIM Throughput	ART	APIM Latency
100	11827	8	4
300	10688	27	13
500	10563	46	25.3
1000	9164	107	49

Performance measurement for large POST payload (5KB)





As seen in the data above, the latency introduced by API Manager is low for a small number of concurrent users. It increases linearly with the number of concurrent users.

SOAP Proxy

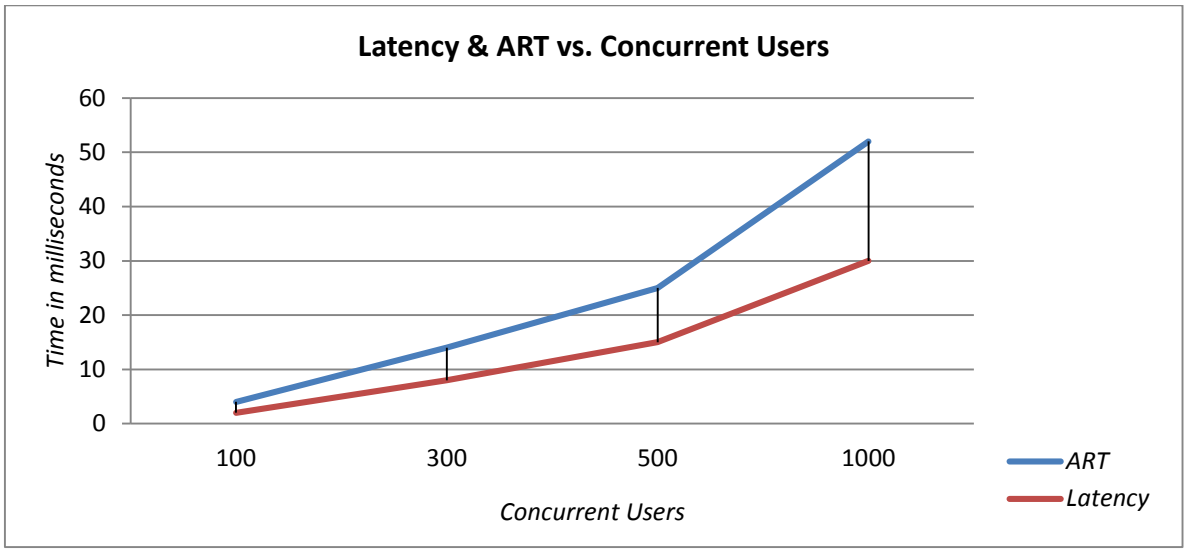
The test results below are from an SOAP web-service for which API Manager acted as a proxy. The service accepts JSON data in the form of a request payload. The tables below show the results for a payload size of 102 bytes and 5KB.

Users	APIM Throughput	ART	APIM Latency
100	19699	4	2
300	19728.5	14	8
500	19470.4	25	15
1000	18522.7	52	30

Performance measurement for small SOAP proxy payload (102 bytes)

Users	APIM Throughput	ART	APIM Latency
100	11967	8	5
300	10860	27	15
500	10434	46	25
1000	9357	101	56

Performance measurement for large SOAP proxy payload (5KB)



As seen in the data above, the latency introduced by API Manager is low for a small number of concurrent users. It increases linearly with the number of concurrent users. In case of larger payload the increase is bit steep for both ART and latency.

SOAP to REST Conversion

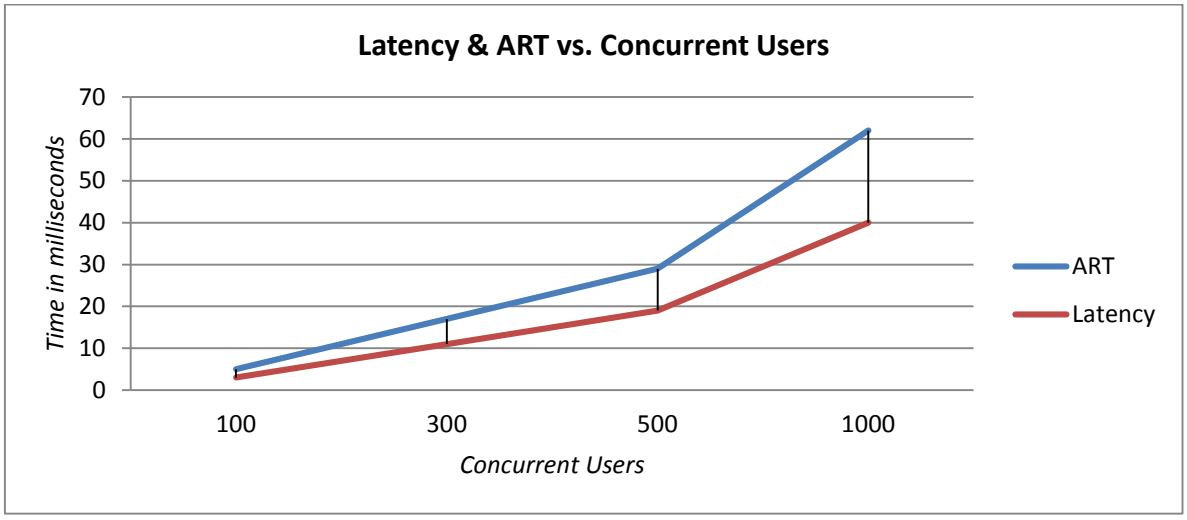
The test results below are from an SOAP web-service that was transformed to a REST web-service by API Manager. It accepts JSON data in the form of request payload. The result below is for a payload size of 102 bytes and a comparatively larger payload of 5KB.

Users	APIM Throughput	ART	APIM Latency
100	16911	5	3
300	16881	17	11
500	16720	29	19
1000	15794	62	40

Performance measurement for small SOAP to REST payload (102 bytes)

Users	APIM Throughput	ART	APIM Latency
100	10416	9	6
300	9922	29	17
500	9728	50	29
1000	8972	107.6	63

Performance measurement for large SOAP to REST payload (5Kb)



As seen in the data above, the latency introduced by API Manager is low for a small number of concurrent users. It increases linearly with the number of concurrent users. In case of SOAP to REST the heavy lifting is done by API manager but the results are close to SOAP proxy.

Testing scalability in a clustered configuration

A single-node API Manager scales really well, but we also see that the throughput decreases and the ART increases with an increase in the number of concurrent users. To maintain a high level of throughput and to keep ART at acceptable levels, in cases where a very high number of concurrent users are expected, a cluster of APIM instances can be deployed. The test results below establish how well API Manager can scale up in a horizontal cluster, to meet the demands of higher volumes of traffic.

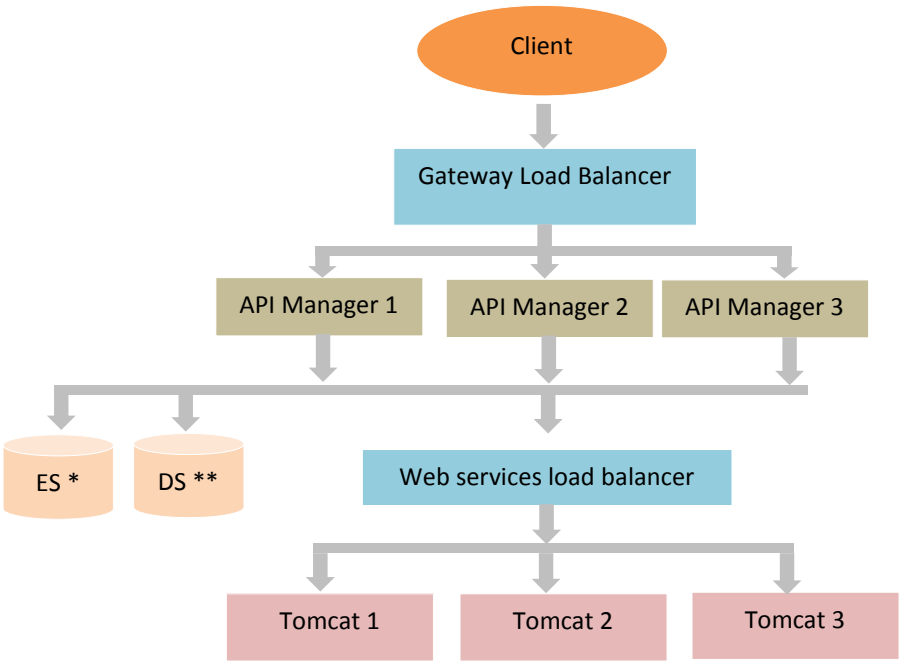
The Cluster Test Environment

The clustered configuration comprises of the following components at different levels:

- Three instances of Apache Tomcat on separate machines, load balanced by a web-services ELB. The back-end web-services are hosted on these server instances.
- Three nodes of API Manager on separate machines, load balanced by a Gateway ELB. These instances share the Data Store and the Analytic Server which are hosted on separate machines.
- A distributed set of three slave JMeter servers, managed by a single JMeter master controller on separate machines.

The APIs are registered in the API Manager with the web-services load balancer as the endpoint, so that the requests from API Manager instances are load balanced evenly across all Tomcat instances. The API Manager instances are in turn load balanced by a gateway load balancer which distributes the load from the JMeter servers evenly across all instances.

A graphical representation of the configuration used for scalability testing is shown below.



** Anayltics server*
*** Data Store*

Cluster Test results

The tables below represent the cumulative statistics from the three-node cluster configuration. The latency shown is the difference in ART when the Tomcat cluster was hit directly as compared to when it was hit via the APIM gateway.

Again, the statistics were collected for a small and a large payload size.

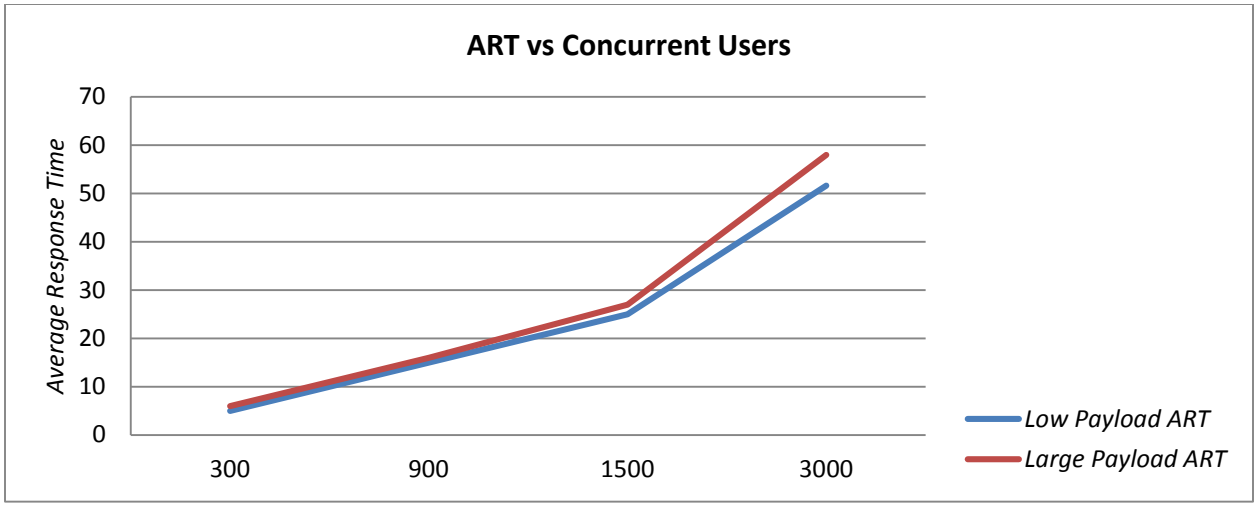
HTTP GET API

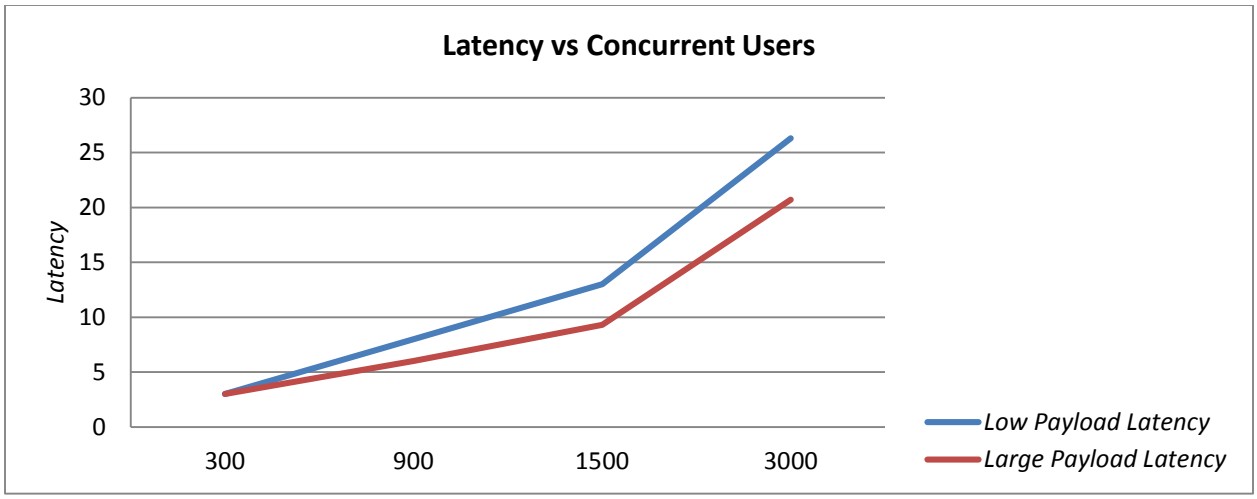
Users	APIM Throughput	ART	Latency
300	56573	5	3
900	54643	15	8
1500	57994	25	13
3000	54594	51.6	26.3

Performance measurement for small GET payload (102 bytes)

Users	APIM Throughput	ART	Latency
300	45995	6	3
900	51595	16	6
1500	53237	27	9.3
3000	50060	58	20.7

Performance measurement for large GET payload (5KB)





A three-node API Manager cluster handles GET requests from 1500 concurrent users, with a latency of 13ms. To put this in perspective, a single-node APIM can handle 1000 users with a latency of 27ms.

HTTP POST API

Users	APIM Throughput	ART	Latency
300	61732	4	2
900	64853	13.3	7.3
1500	64696	22.3	11.3
3000	66446	44	22

Performance Measurement for small POST payload (102 bytes)

Users	APIM Throughput	ART	Latency
300	49255	5.3	1
900	45970	18	2
1500	44908	32	5
3000	41266	69	18.7

Performance measurement for large POST payload (5KB)

A three-node API Manager cluster handles POST requests from 1500 concurrent users, with a latency of just 11ms. To put this in perspective, a single-node APIM can handle 1000 users with a latency of 28ms.

SOAP Proxy

Users	APIM Throughput	ART	APIM Latency
300	54073	5	3
900	54539	16	11
1500	54532	26.3	14.3
3000	54823	52	26

Performance measurement for small SOAP Proxy payload (150 bytes)

Users	APIM Throughput	ART	APIM Latency
300	40319	7	3
900	40244	21	7
1500	38750	37	12.3
3000	35876	79	39

Performance measurement for large SOAP Proxy payload (5 KB)

While a single node API Manager can service SOAP requests for 500 concurrent users at a throughput of just under 19,500 requests per second, a three-node API Manager cluster delivers a throughput of almost 55,000 requests for 1,500 users.

SOAP to REST Conversion

Users	APIM Throughput	ART	APIM Latency
300	49336	5	3
900	59529	14	7
1500	60605	23.5	11
3000	53253	54.7	28.7

Performance measurement for small SOAP to REST payload (102 bytes)

Users	APIM Throughput	ART	APIM Latency
300	33661	8	4
900	34576	25	11
1500	34520	41	16.3
3000	27122	98.5	58.5

Performance measurement for large SOAP to REST payload (5KB)

While a single-node API Manager can service SOAP-based REST requests for 500 concurrent users at a throughput of just under 17000 requests per second, a three-node API Manager cluster does the same at a throughput of over 60000 requests for three times the number of users.

As observed in the single node configuration statistics, the throughput increases with an increase in the number of concurrent users. If the number of concurrent users are higher than what can be handled efficiently by a single node of API Manager, a cluster of API Managers can be used to maintain high throughput.

Determining Number of Cluster Nodes

Use a cluster for two reasons:

1. Maintain low latency
2. Have more throughput

Based on latency (GET)

For 1000 concurrent users, the latency introduced by a single-node API Manager is 27ms. Using a three-node cluster under similar conditions, lowers the latency to under 10ms.

Based on throughput (GET)

A single-node API Manager delivers a throughput of over 22,000 requests per second. A three-node cluster can boost it to around 55000 requests per second.

Performance Tuning Guidelines

This section describes the configuration settings and the parameters that can be tuned to achieve an optimal performance from API Manager. The settings can be segregated into the following components:

- API Manager
- Data Store
- Analytics Server
- OS

API Manager – core server tuning

These are general guidelines to get the best performance. These settings can be tuned by monitoring the CPU usage, throughput, and CPU-run queue with tools such as the vmstat on Linux or the Performance Monitor on Windows.

Acceptor and Worker threads

These settings are in the config.xml file in the <APIM_HOME>\conf\ directory. The optimal value for these settings depend on the number of CPU cores in the host machine.

The Acceptor thread count determines the number of non-blocking threads that accept a request and pass it on to a worker thread for processing. The recommended value for acceptor thread is two per core. For example, if the machine has four cores, then the acceptor thread count should be set to eight.

The Worker thread count determines the number of core threads that process a request. A reasonable value of this count is 16 per core.

JVM Heap Size

The max (*Xmx*) and min (*Xms*) JVM heap size, can be set in the jvm.config file in Windows, or the start.sh script in non-Windows platforms, in the <APIM_root>/bin directory. To prevent the heap from resizing during runtime, keep the max and the min size, the same.

Data Store connection pooling

This setting is present in the pooling.properties file at the <APIM_HOME>\conf directory. The ideal value for the connection pool size depends on the number of worker threads. For best results, set ***maxTotal***, ***minIdle,*** and ***maxIdle*** to the same value as the number of worker threads. Every runtime request processed by the API Manager makes a call to the Data Store to check and enforce the rate limit set in the SLA plan. If the Data Store connection pool is small, then the runtime requests will have to wait till a connection is available from the pool to move forward.

HTTP Connection Pooling

The HTTP connection pooling properties can be found in the config.xml file at <APIM_HOME>\conf directory. The ***max-connections*** and ***max-default-connections-per-route*** should be set to a value greater than or equal to the number of worker threads. Every runtime call processed by the API Manager will use a connection form the HTTP connection pool (unless the response is cached by API Manager).

If the response time for the endpoint web-service is high increase the timeout for the following settings in the same configuration file.

connection-request-timeout : The timeout for a connection request from the HTTP connection pool.
connect-timeout : The timeout for connecting to the web-services-endpoint.
socket-timeout: The timeout for inactive period between data packets in a socket connection.

Asynchronous Logging

The log level used for API Manager is INFO by default. INFO level messages are logged only for the internal server, but not for all the requests processed by API Manager, to avoid impacting the server performance.

The messages logged by APIM, either have ERROR priority (if an error occurs in runtime) or DEBUG priority (if debugging is enabled. By default, it is not.) So in the runtime flow, no messages are logged, if there are no issues. API Manager uses synchronous logging by default. If API Manager encounters a lot of errors in runtime, its performance may be impacted.

As a precaution, you can make logging asynchronous, thus avoiding the costly file I/O operations in the runtime flow. To enable asynchronous logging, add the following system property to the JVM arguments, in the jvm.config in Windows, or the start.sh script, in a non-Windows OS.

-DLog4jContextSelector=org.apache.logging.log4j.core.async.AsyncLoggerContextSelector

Analytic Server client tuning

This component of the API Manager is an indexing-heavy application. It collects the metrics for each request, indexes it and stores it for record keeping.

API Manager sends data to the Analytics sever in batches of bulk requests. Each bulk request can contain multiple metrics documents. The manner in which this bulk request is dispatched can be controlled by four key settings that you can configure in the “Analytics Server Settings” section in the API Manager Admin portal. These configurable settings are:

Flush Interval – The frequency at which bulk request is made to the Analytics server. The default value is one second. On a low load system, this value can be increased. The drawback of increasing the flush interval is the result displayed in the Analytics dashboard will be less real-time.

Maximum Actions per Bulk Request – The number of requests processed for which a bulk request is made to the Analytics server. The default value is 1000 actions per bulk request. This means that whenever, 1000 requests are received by the API Manager, a bulk request is dispatched to the Analytics server.

Maximum Volume per Bulk Request – This setting determines the physical size of the bulk request that is dispatched to the Analytics server. The default value is 5 MB. When the cumulative size of the documents reach 5 MB, a bulk request is dispatched.

Maximum Concurrent Bulk Requests – The number of simultaneous bulk requests that can be made to the Analytics server. The default value is one. That means only one bulk request can be made to the Analytics server at a given time. In our test environment, we had increased it to 15 without any issues.

Data Store Tuning

The Data Store uses Redis as the underlying server. Redis is a single-threaded server. Therefore, using a CPU with high clock speed and large cache is advised as opposed to using a multi-core processor.

In a multi-core CPU-based machine, it is recommended that you isolate and dedicate a core to the Redis process, so that the system does not schedule any system tasks on that core. On a Linux based system, use the *isolcpus* option, in the bootloader, to isolate a core and *taskset* command, to bind it to the redis process.

If the Data Store is hosted a remote machine, ensure ample network bandwidth. If a high volume of traffic is expected, use a high bandwidth NIC card (10 Gbit/s or above) or use multiple NIC cards with TCP/IP bonding.

For details on the factors affecting the performance of Redis, refer - http://redis.io/topics/benchmarks

Analytic Server Tuning

The Analytics server uses Elasticsearch as the underlying indexing and search server. Since Analytics Server performs file I/O intensive operations, the use of SSD drives is recommended.

The refresh interval is one of the key parameters in tuning the Analytics server. The API Manager uses a refresh interval of 60 seconds by default. This can be configured in the APIM administrator portal. A higher refresh interval translates to better indexing performance. The trade-off here is that the analytics data will be searchable only after the refresh interval time. If refresh interval is set to 60 seconds, then even after a request is served by API Manager, the data will be visible in Analytics dashboard only after a minute.

You should allocate an optimal heap size to the Analytic server, based on the availability of memory and other processes on the machine and disable swapping between main memory and secondary storage, in the host system.

You can set the JVM heap size for Analytics server by using the following environment variable:
*export **ES_HEAP_SIZE**=30g*

To prevent swapping, use the following system command on Linux:
sudo swapoff –a

You can also enable mockall by using the following setting in the elasticsearch.yml file at *<APIM_root>\database\analytics\config* directory.

bootstrap.mlockall: true

This **enables JVM to lock** its memory to prevent it from being swapped by the OS.

For further details on sizing the JVM heap, refer:
https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html

For optimizing the Elasticsearch indexing, refer:
https://www.elastic.co/guide/en/elasticsearch/guide/current/indexing-performance.html

OS Tuning

To set the number of open file descriptors for system users, configure the following options in the /etc/security/limits.conf file

** soft nofile <no_of_files>*
** hard nofile <no_of_files>*

Alternatively, use the ulimit command as follows:

ulimit –n <no_of_files>

To set the maximum number of processes available to a user, configure the following options in the /etc/security/limits.conf file

** soft nproc <no_of_files>*
** hard nproc <no_of_files>*

Alternatively, use the ulimit command as follows:

ulimit –u <no_of_files>

If the changes are done in the config file, reload the file using the *sysctl* command as follows:

sysctl –p <config_file>

Increase the port range and decrease the TCP connection timeout by using the following directives in the /etc/sysctl.conf file:

net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_fin_timeout = 30

Test Environment Specifications

All the components in our performance testing were hosted on c3.8xlarge type HVM AWS instances. The following specifications were common to these instances.

- RHEL 7.2 64-bit
- 32-core CPU (2.8 GHz)
- 60 GB RAM
- ESB storage.
- 10 Gbps NIC

Apache Tomcat 8.0.24 was used to host the back-end web-services. A distributed set of Apache JMeter 2.13 servers were used as the load testing tool.
The AWS Elastic Load Balancers were pre-warmed before use.

Conclusion

The API Manager is designed to be an ideal solution to add value to APIs, with minimal performance overhead.