

Nachiket Joshi

SJSU ID: 011408956

Extra Credit Assignment

NETFLIX ATLAS

Github Link

<https://github.com/TheBloodMage/Netflix-Atlas-Extra-Credit-Assignment>

TABLE OF CONTENTS

<u>CHAPTER</u>	<u>PAGE</u>
CHAPTER 1 – INTRODUCTION TO NETFLIX ATLAS	3
CHAPTER 2 – OVERVIEW OF NETFLIX ATLAS.....	4
CHAPTER 3 – REAL TIME ANALYSIS WITH A NETFLIX DEMO	10
CHAPTER 4 – CONCLUSION	17

INTRODUCTION TO NETFLIX ATLAS

Every company nowadays is investing much into the realms of Big Data analysis. The sentiment analysis to know the near future analysis of the data is going to be useful for the business. Every company generated multiple terabytes of data every day. The real time analysis to give the operational insights of this data will generate positive results to decide the direction of the business.

In pursuit of above-mentioned phenomenon, Netflix developed Atlas to manage dimensional time series data for a near future real-time operational insight. The main feature of Atlas is that it provides in memory data storage. This allows gathering and reporting of very large number of metrics and that too quickly.

Atlas is a tool specifically designed to study operational intelligence. This study is done over the business data that is gathered over time. The main aim of operational system is to know what exactly is happening inside a system currently. This also helps establish trends among the customers.

The business that the Netflix does suddenly increased from 2011 to 2014 and the existing operational intelligence methods were not suitable to study this gigantic leap in data to study thus they designed this new Atlas based on the stack language which is able to handle big data easily. Netflix can also scale with the hardware that we use.

OVERVIEW OF NETFLIX ATLAS

- **OVERVIEW**

Atlas is a system that the Netflix uses to manage highly complex to manage dimensional time-series data. The issues it addresses are scalability and high query attendance capability in the previous system.

The main goal of this system was to build a system that provided,

- **A Common API**

A common AP to serve following aspects,

- Normalization and consolidation of business data
- Flexible legends that scale independently of the chart
- Math, especially handling of **NaN** values representing no data
- Holt-Winters used for alerting
- Visualization options
- Deep linking

- **Scale**

- Metrics volume was growing as the growth in the business wa evident and Netflix needed a system that could keep up.
- One of the biggest concerns for a considerable amount of time was the write volume, however, it was important to scale in terms of the amount of data we could read or aggregate as part of a graph request.

- **Dimensionality**

The users were already doing this it was evident to attend this.

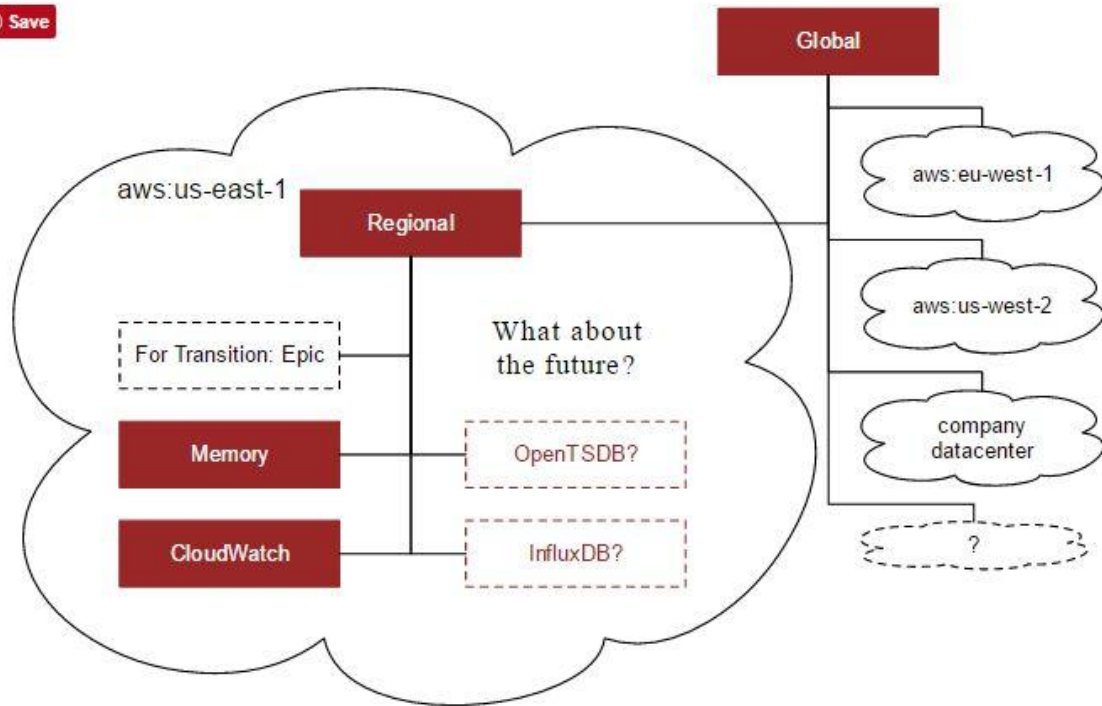
The names users created were as follows:

**com.netflix.eds.nccp.successful.requests.uiversion.nccprt-authorization.devtypid-101.clver-
PHL_0AB.uiver-UI_169_mid.geo-US**

- **Query Layer**

Following is the diagram, which explains the interaction of the query with the created interface.

Netflix has allowed the query and rendering layer to work on multiple backends. This also makes it easier for Netflix to consider transitioning to other backends in the future such as OpenTSDB or InfluxDB.



- **Stack Language**

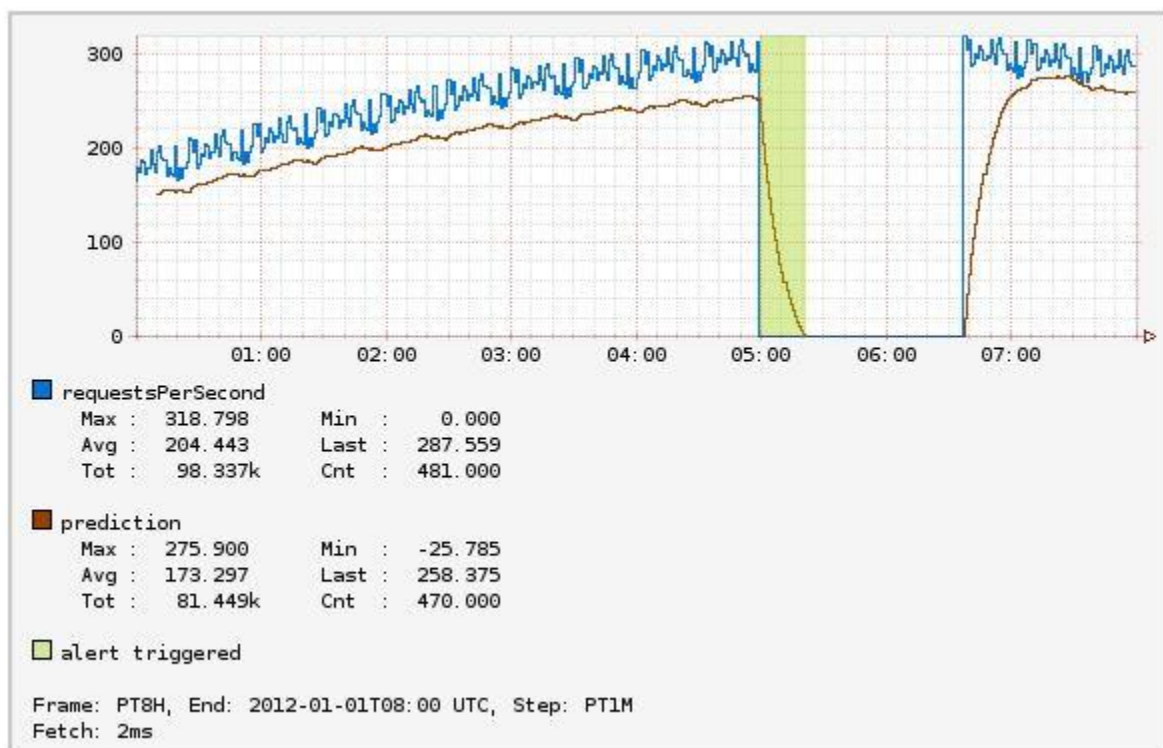
One of the key requirements was to be able to have deep links into particular charts. This could not have been possible without any simple query language thus they invested in a simpler version of query language called **Stack Language**.

The stack language is also simple to parse and interpret, allowing it to be easily consumed into a variety of tools. The core features could be listed as follows,

- Embedding and linking using a GET request
- URL friendly stack language
 - Few special symbols (comma, colon, parenthesis)
 - Easy to extend
- Basic operations
 - Query: and, or, equal, regex, has key, not
 - Aggregation: sum, count, min, max, group by
 - Consolidation: aggregate across time
 - Math: add, subtract, multiply, etc
 - Boolean: and, or, lt, gt, etc
 - Graph settings: legends, area, transparency

- **GRAPH EXAMPLE**

Following is one of the samples of a graph that I generated from the available sample java server



This graph shows the time required per user per request served in unit amount of time. We can also add prediction criteria in these graphs. The prediction can be added with **double exponential smoothing**. If the number of requests per second drops below a certain level we could add a certain trigger level in these graphs.

The essence of the use of **Stack Query Language** is that we can even add Stack Expressions to explain a bit, of what is going on...

A sample Stack Expression for the trigger we can see above in the graph example could be as follows,

```
# Create a boolean signal line that is 1
# for datapoints where the actual value is
# less than the prediction and 0 where it
# is greater than or equal the prediction.
# The 1 values are where the alert should
# trigger.
```

- **MEMORY STORAGE**

Storage for Atlas has been one of the sore points. Netflix have tried many backends and ended up moving more and more towards a model, which pretty much stores all data in memory. This is instead of storing the data on java heaps. We could list following advantages for doing so...

- **SPEED**

One of the primary goals of Atlas is to support the queries over dimensional time series databases. When the requirements is to perform aggregations over many data points even though the result data set might be small.

A simple example could be like follows,

A simple graph showing number of hits per second hitting a service for last 6 hours.

Assuming minute resolution that is 180 data points for the final output, on a typical second we would get one time series per node showing the number of requests so if we have 100 nodes on the international result is around 18k data points.

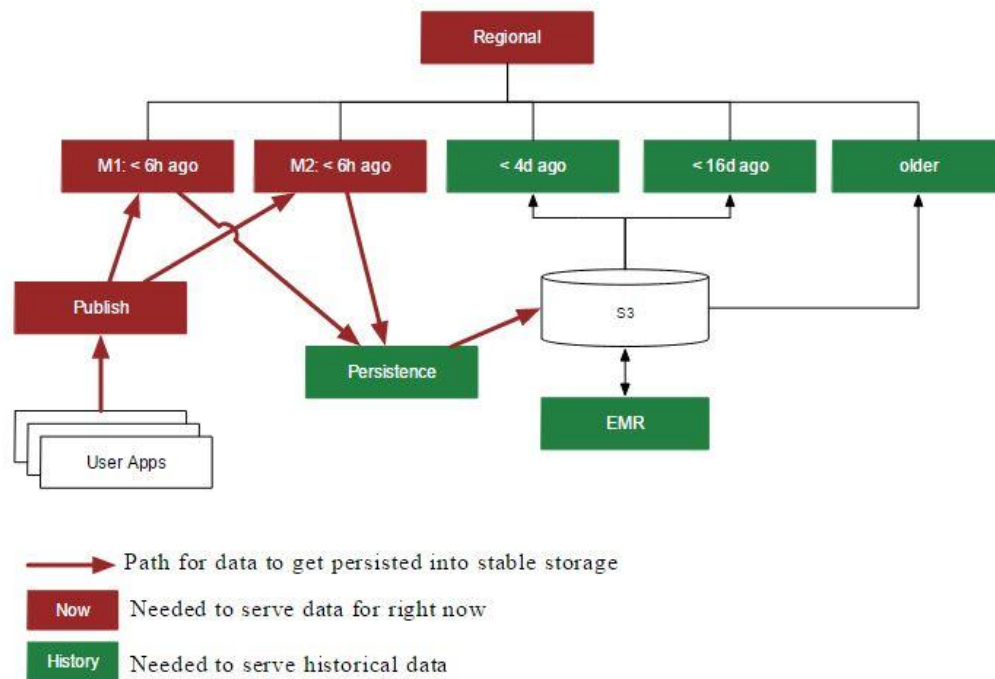
For one service, users went hog-wild dimensions breaking the down requests by device (~1000s) and country (~50s) lending for the same 6h line.

- **RESILIENCE**

When we think, about what all has to be working in order that the monitoring system to works, it falls over what is involved in getting it back up our focus is primarily operational insight so the top priority is to be able to determine what s going on right now. This leads to the following rules of thumb,

- Data becomes exponentially less important as it gets older and older
- Restoring a service is more important than preventing data loss
- Try to degrade gracefully when you must

Following is the windows of data diagram that data points could contain.



• COST

Keeping large data sets in memory is expensive also keeping in track the entire growth rate of the data storage. The combination of dimensionality and time based partitioning used for resilience also gives us a way to help manage costs.

The first way is in controlling the number of replicas. In most of the cases, Netflix is using same replicas for redundancy not to provide additional query capacity.

For historical data the can be reloaded from stable storage, Netflix typically runs only one replica as the duration of partial downtime was not deemed to be worth the cost for an additional replica.

• ECOSYSTEM

Internally Netflix has done a lot of tooling and infrastructure build up around Atlas. Many of these will subsequently be open sourced future. According to the github page of the Atlas we can expect following additional parts in near future,

- User interfaces
 - Main UI for browsing data and constructing queries.
 - Dashboards
 - Alerts
- Platform
 - Inline aggregation of reported data before storage layer
 - Storage options using off-heap memory and lucene
 - Percentile backend
 - Publish and persistence applications
 - EMR processing for computing rollups and analysis
 - Poller for SNMP, healthchecks, etc
- Client
 - Supports integrating servo with Atlas
 - Local rollups and alerting
- Analytics
 - Metrics volume report
 - Canary analysis
 - Outlier and anomaly detection

REAL TIME ANALYSIS WITH A NETFLIX DEMO

- **Prerequisites To Study Real Time Insights**

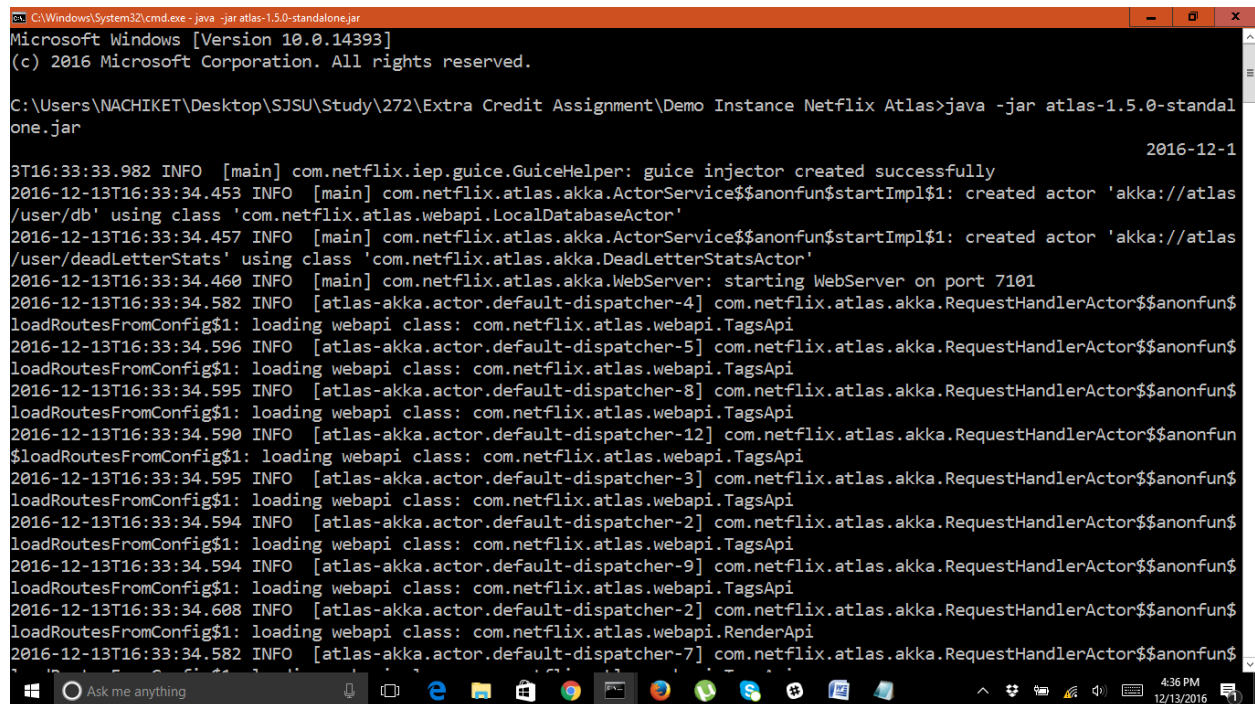
- ✓ **CURL**
- ✓ **JAVA 8 or higher**

- **RUN A DEMO INSTANCE**

1. To Start A Netflix Server Command

```
java -jar atlas-1.5.0-standalone.jar
```

✓ **Output:**



```
C:\Windows\System32\cmd.exe - java -jar atlas-1.5.0-standalone.jar
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\Demo Instance Netflix Atlas>java -jar atlas-1.5.0-standalone.jar

2016-12-13T16:33:33.982 INFO [main] com.netflix.iep.guice.GuiceHelper: guice injector created successfully
2016-12-13T16:33:34.453 INFO [main] com.netflix.atlas.akka.ActorService$$anonfun$startImpl$1: created actor 'akka://atlas/user/db' using class 'com.netflix.atlas.webapi.LocalDatabaseActor'
2016-12-13T16:33:34.457 INFO [main] com.netflix.atlas.akka.ActorService$$anonfun$startImpl$1: created actor 'akka://atlas/user/deadLetterStats' using class 'com.netflix.atlas.akka.DeadLetterStatsActor'
2016-12-13T16:33:34.460 INFO [main] com.netflix.atlas.akka.WebServer: starting WebServer on port 7101
2016-12-13T16:33:34.582 INFO [atlas-akka.actor.default-dispatcher-4] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T16:33:34.596 INFO [atlas-akka.actor.default-dispatcher-5] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T16:33:34.595 INFO [atlas-akka.actor.default-dispatcher-8] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T16:33:34.590 INFO [atlas-akka.actor.default-dispatcher-12] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T16:33:34.595 INFO [atlas-akka.actor.default-dispatcher-3] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T16:33:34.594 INFO [atlas-akka.actor.default-dispatcher-2] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T16:33:34.594 INFO [atlas-akka.actor.default-dispatcher-9] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T16:33:34.608 INFO [atlas-akka.actor.default-dispatcher-2] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.RenderApi
2016-12-13T16:33:34.582 INFO [atlas-akka.actor.default-dispatcher-7] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
```

2. EXPLORE ADDITIONAL TAGS

```
curl -s "http://localhost:7101/api/v1/tags"
```

✓ OUTPUT:

```
C:\Windows\System32\cmd.exe
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -s "http://localhost:7101/api/v1/tags"
[{"name", "nf.app", "nf.asg", "nf.cluster", "nf.node", "statistic", "type", "type2"}]
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>
```

3. Show all values of the name, nf.app and type tags

```
curl -s "http://localhost:7101/api/v1/tags/name"
curl -s "http://localhost:7101/api/v1/tags/nf.app"
curl -s "http://localhost:7101/api/v1/tags/type"
```

✓ Output

```
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -s "http://localhost:7101/api/v1/tags/name"
[{"DiscoveryStatus_DOWN", "DiscoveryStatus_UP", "DiscoveryStatus_nccp_UP", "playback.startLatency", "poller.asg.instance", "requestsPerSecond", "sps", "ssCpuUser"}]
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -s "http://localhost:7101/api/v1/tags/nf.app"
[{"alerttest", "nccp"}]
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -s "http://localhost:7101/api/v1/tags/type"
[{"high-noise", "ideal", "low-noise"}]
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>
```

4. show all name tags that also have the type tag

```
curl -s "http://localhost:7101/api/v1/tags/name?q=type,:has"
```

✓ OUTPUT

```
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -s "http://localhost:7101/api/v1/tags/name?q=type,:has"
[{"sps"}]
```

5. show all name tags that have an nf.app tag with a value of nccp

```
curl -s "http://localhost:7101/api/v1/tags/name?q=nf.app,nccp,:eq"
```

✓ OUTPUT

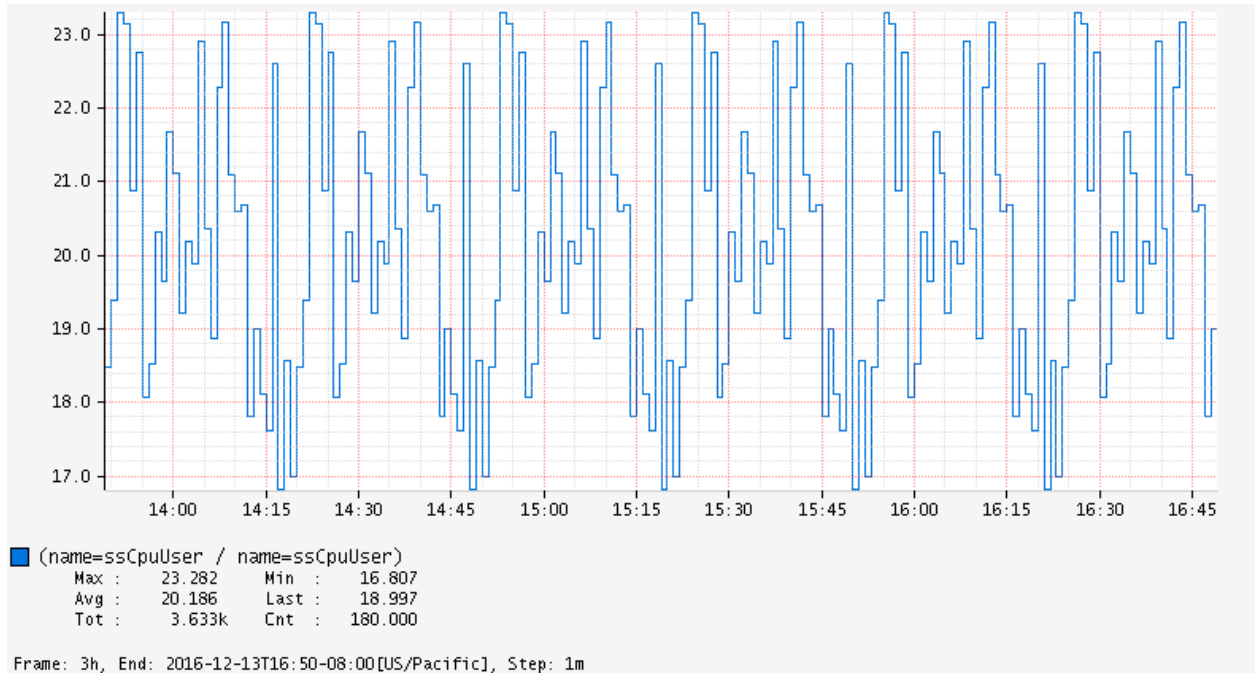
```
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -s "http://localhost:7101/api/v1/tags/name?q=nf.app,nccp,:eq"
[{"DiscoveryStatus_nccp_UP", "playback.startLatency", "poller.asg.instance", "sps"}]
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>
```

6. GENERATE GRAPHS

graph all metrics with a name tag value of ssCpuUser, using an :avg aggregation

```
curl -Lo graph.png  
"http://localhost:7101/api/v1/graph?q=name,ssCpuUser,:eq,:avg"
```

✓ OUTPUT



Duplicate the ssCpuUser signal, check if it is greater than 22.8 and display the result as a vertical span with 30% alpha

```
curl -Lo graph.png
"http://localhost:7101/api/v1/graph?q=name,ssCpuUser,:eq,:avg,:dup,22.8,:gt
,:vspan,30,:alpha"
```

✓ OUTPUT



OUTPUT OF GRAPH GENERATION IN CMD

```
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -Lo graph.png "http://localhost:7101/api/v1/graph?q=name,ssCpuUser,:eq,:avg"
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 16953 100 16953 0 0 9119 0 0:00:01 0:00:01 --:--:-- 9119

C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>curl -Lo graph.png "http://localhost:7101/api/v1/graph?q=name,ssCpuUser,:eq,:avg,:dup,22.8,:gt,:vspan,30,:alpha"
% Total % Received % Xferd Average Speed Time Time Time Current
Dload Upload Total Spent Left Speed
100 19615 100 19615 0 0 69804 0 --:--:-- --:--:-- --:--:-- 69804

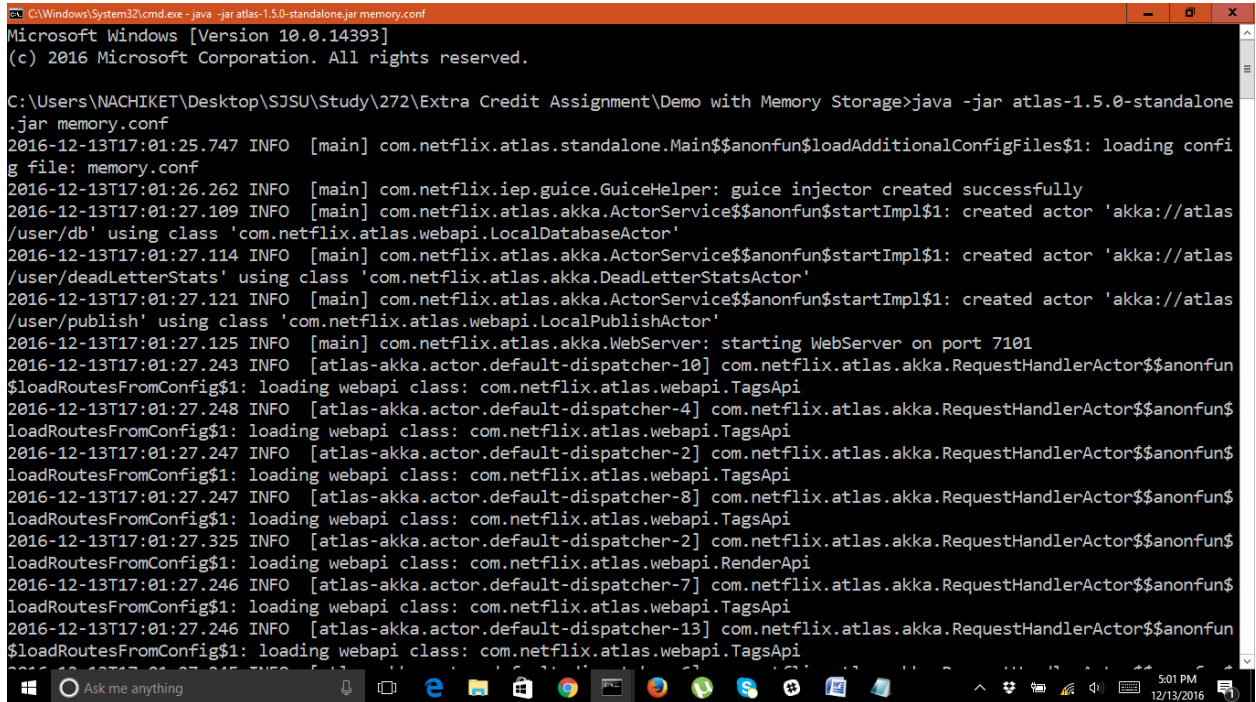
C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\test>
```

- **RUN A DEMO NETFLIX INSTANCE WITH MEMORY STORAGE**

1. **Run an instance with a configuration to use the memory storage:**

```
java -jar atlas-1.5.0-standalone.jar memory.conf
```

✓ **OUTPUT**



```
C:\Windows\System32\cmd.exe - java -jar atlas-1.5.0-standalone.jar memory.conf
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\NACHIKET\Desktop\SJSU\Study\272\Extra Credit Assignment\Demo with Memory Storage>java -jar atlas-1.5.0-standalone.jar memory.conf
2016-12-13T17:01:25.747 INFO [main] com.netflix.atlas.standalone.Main$$anonfun$loadAdditionalConfigFiles$1: loading config file: memory.conf
2016-12-13T17:01:26.262 INFO [main] com.netflix.iep.guice.GuiceHelper: guice injector created successfully
2016-12-13T17:01:27.109 INFO [main] com.netflix.atlas.akka.ActorService$$anonfun$startImpl$1: created actor 'akka://atlas/user/db' using class 'com.netflix.atlas.webapi.LocalDatabaseActor'
2016-12-13T17:01:27.114 INFO [main] com.netflix.atlas.akka.ActorService$$anonfun$startImpl$1: created actor 'akka://atlas/user/deadLetterStats' using class 'com.netflix.atlas.akka.DeadLetterStatsActor'
2016-12-13T17:01:27.121 INFO [main] com.netflix.atlas.akka.ActorService$$anonfun$startImpl$1: created actor 'akka://atlas/user/publish' using class 'com.netflix.atlas.webapi.LocalPublishActor'
2016-12-13T17:01:27.125 INFO [main] com.netflix.atlas.akka.WebServer: starting WebServer on port 7101
2016-12-13T17:01:27.243 INFO [atlas-akka.actor.default-dispatcher-10] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T17:01:27.248 INFO [atlas-akka.actor.default-dispatcher-4] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T17:01:27.247 INFO [atlas-akka.actor.default-dispatcher-2] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T17:01:27.247 INFO [atlas-akka.actor.default-dispatcher-8] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T17:01:27.325 INFO [atlas-akka.actor.default-dispatcher-2] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.RenderApi
2016-12-13T17:01:27.246 INFO [atlas-akka.actor.default-dispatcher-7] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
2016-12-13T17:01:27.246 INFO [atlas-akka.actor.default-dispatcher-13] com.netflix.atlas.akka.RequestHandlerActor$$anonfun$loadRoutesFromConfig$1: loading webapi class: com.netflix.atlas.webapi.TagsApi
```

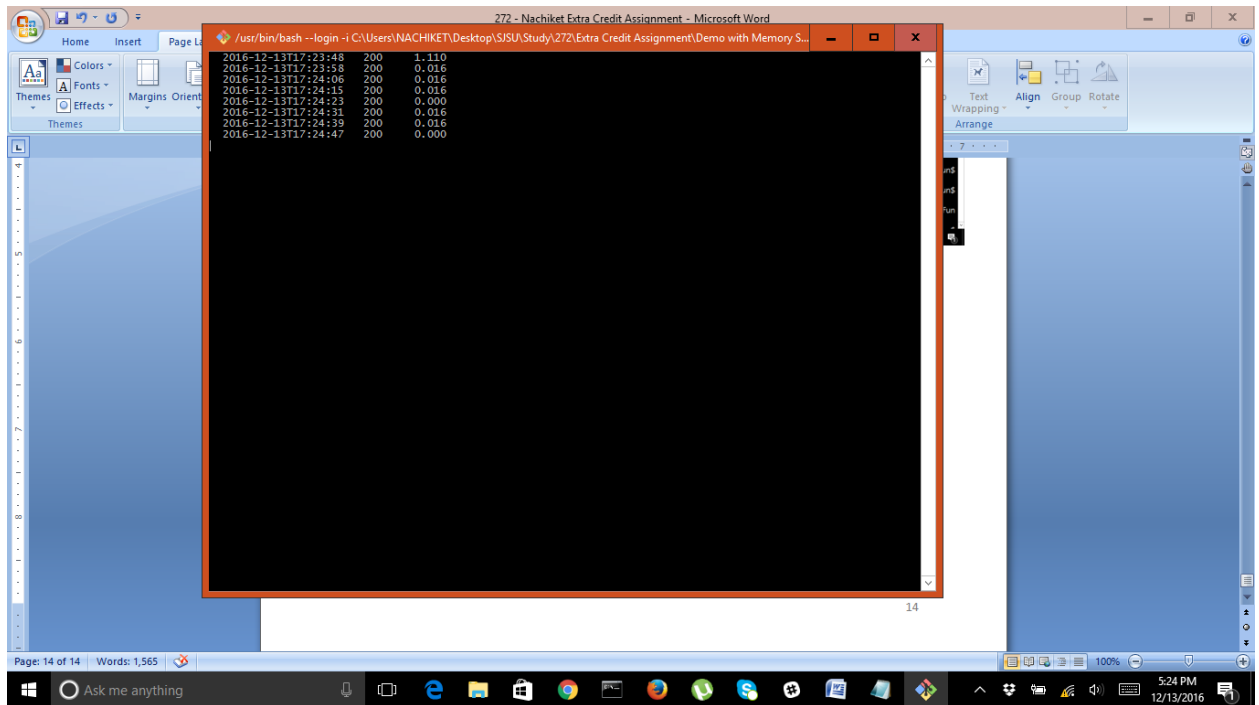
2. **This created server needs data to generate graphs. This data is stored in a sample script file.**

We have to run that shell script and publish the data.

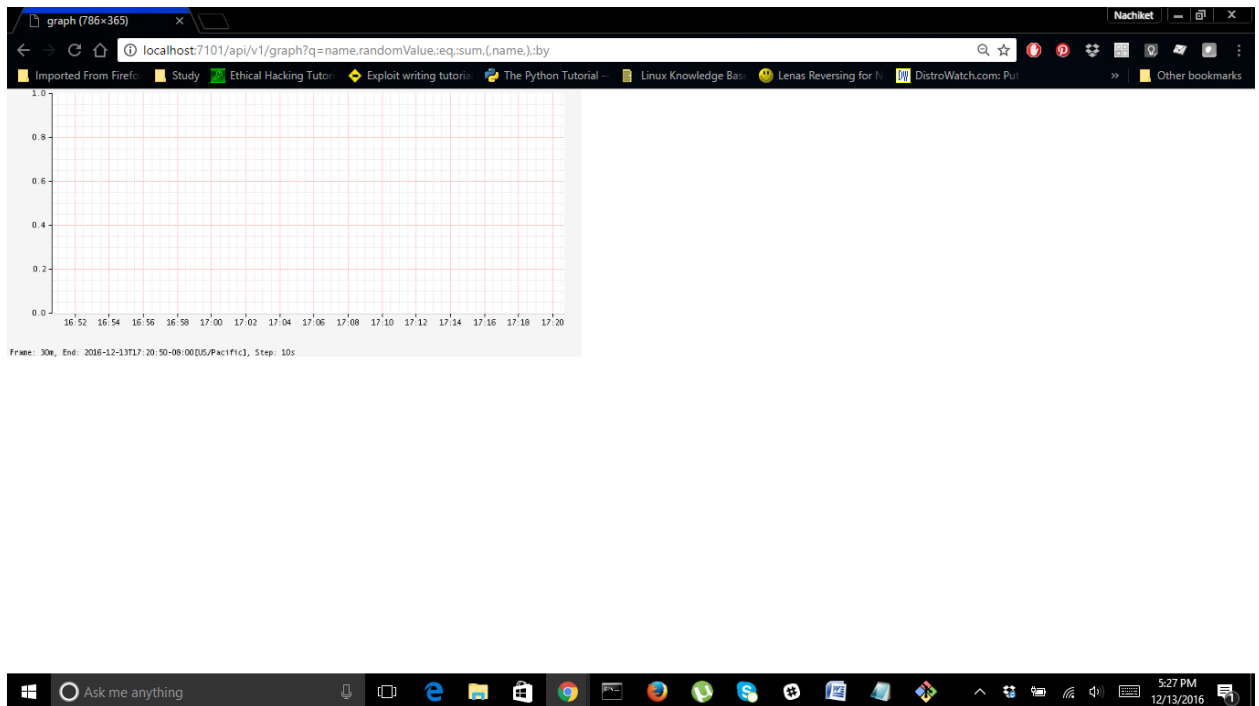
I downloaded the shell file and ran it. The screen shot of the graph before and after the shell run are as follows...

```
curl -Lo publish-test.sh
"https://raw.githubusercontent.com/Netflix/atlas/master/scripts/publish-test.sh"
chmod 755 publish-test.sh
./publish-test.sh
```

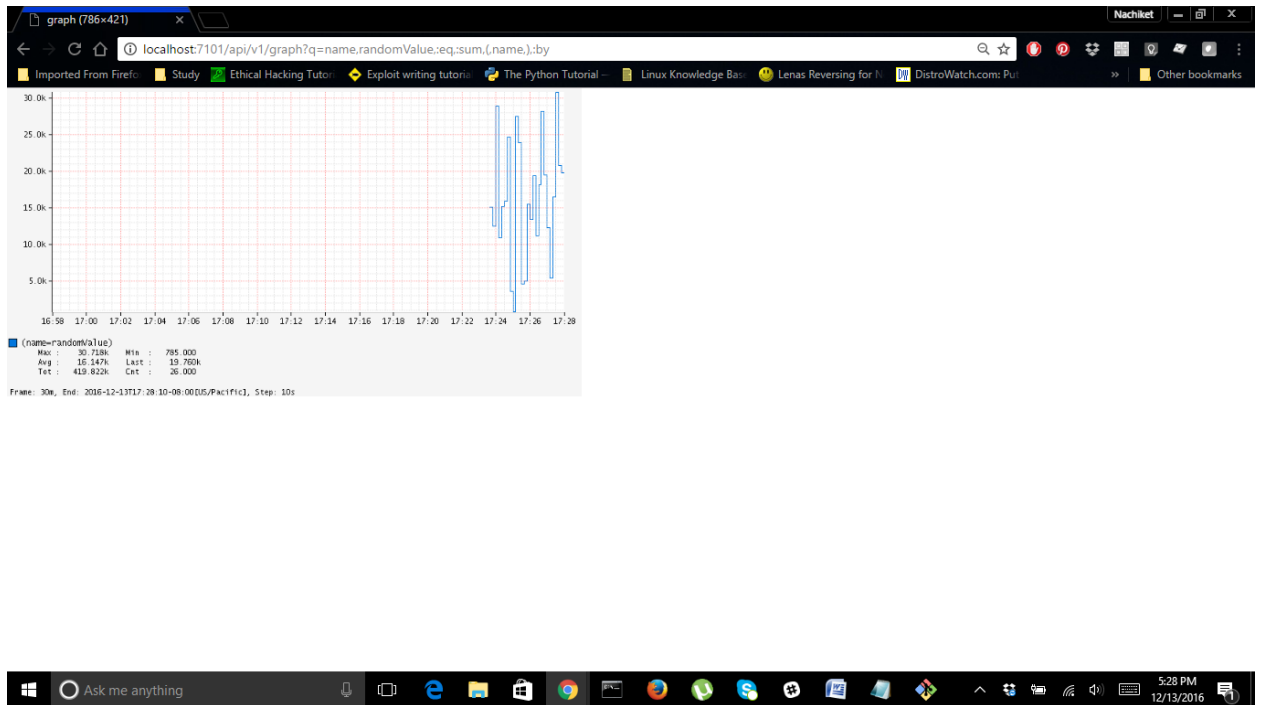
STEP 1: Publishing the data



STEP 2: LOCALHOST BEFORE SHELL RUN



STEP 2: LOCALHOST AFTER SHELL RUN



CONCLUSION

Atlas is designed to handle large quantity of data and can scale with the hardware we use to analyze and store it. This is one the emerging technologies to deal and study with operational insights of large data efficiently. Although many features of the language are yet to be made open source, this could make a large impact on the way we analyze the operational data to manage dimensional time series data for near real-time operational insights.