Advances in Computational Sciences and Technology
ISSN 0973-6107 Volume 12, Number 1 (2019) pp. 1-6
© Research India Publications
http://www.ripublication.com

Cloud Based Log Analyzing and Archiving Architectural Solution

S. SATHYANARAYAN
DevOps Engineer, Amazon

Abstract

Logging is core part of an application. This paper would discuss a Cloud based serverless log analyzing and archiving solution using AWS Component or services. This system or design intends to provide the user the capability of querying or rather analyzing the logs using the power of SQL. The Architecture incorporates the capability of joining data from multiple logs sources that are ingested within minutes of the log event. This design set-up would be a single solution in spite your systems be in cloud, on-premises or hybrid and would put an end to infrastructure maintenance overhead for the service owners. Experimental Detail section of the paper show case the efficiency of this design over usual standard procedure of finding the anomaly in the logs.

INTRODUCTION

Logging is core part and parcel of an application. In general log analysis is the process of obtaining an inference from log file. The logging varies across platform's, applications, devices, and so is the importance of different type of logs like application logs, service logs, system logs etc. The use of logs varies from developers, sysadmins, security team. As developers and sysadmins intend to monitor the system performance and efficiency using logs, whereas security and vulnerability engineers intend to scan logs out for any breach. As of in legacy systems(non-cloud), analyzing and archiving is a difficult solution as service owners have to build their own application on top of their logs to process and obtain inference, at the same time have the overhead of maintaining the infrastructure. Later section of this paper discuss log analyzing, archiving architecting solution for distributed systems.

DESIGN GOALS

1. Minimizing Infrastructure maintenance overhead:

Storing logs always have been overhead out for the service owners, in many cases the service owners keep separate hardware usually storage intensive to keep the logs out for a long time.

2. **Serverless Architecture:**

The components used in the design are serverless, therefore no infra maintenance from developer or service owner side.

3. **Develop a generic architecture**

Which could be used to implement even if your application is not on cloud. This design could be used even if your systems are on-premises, hybrid or on cloud.

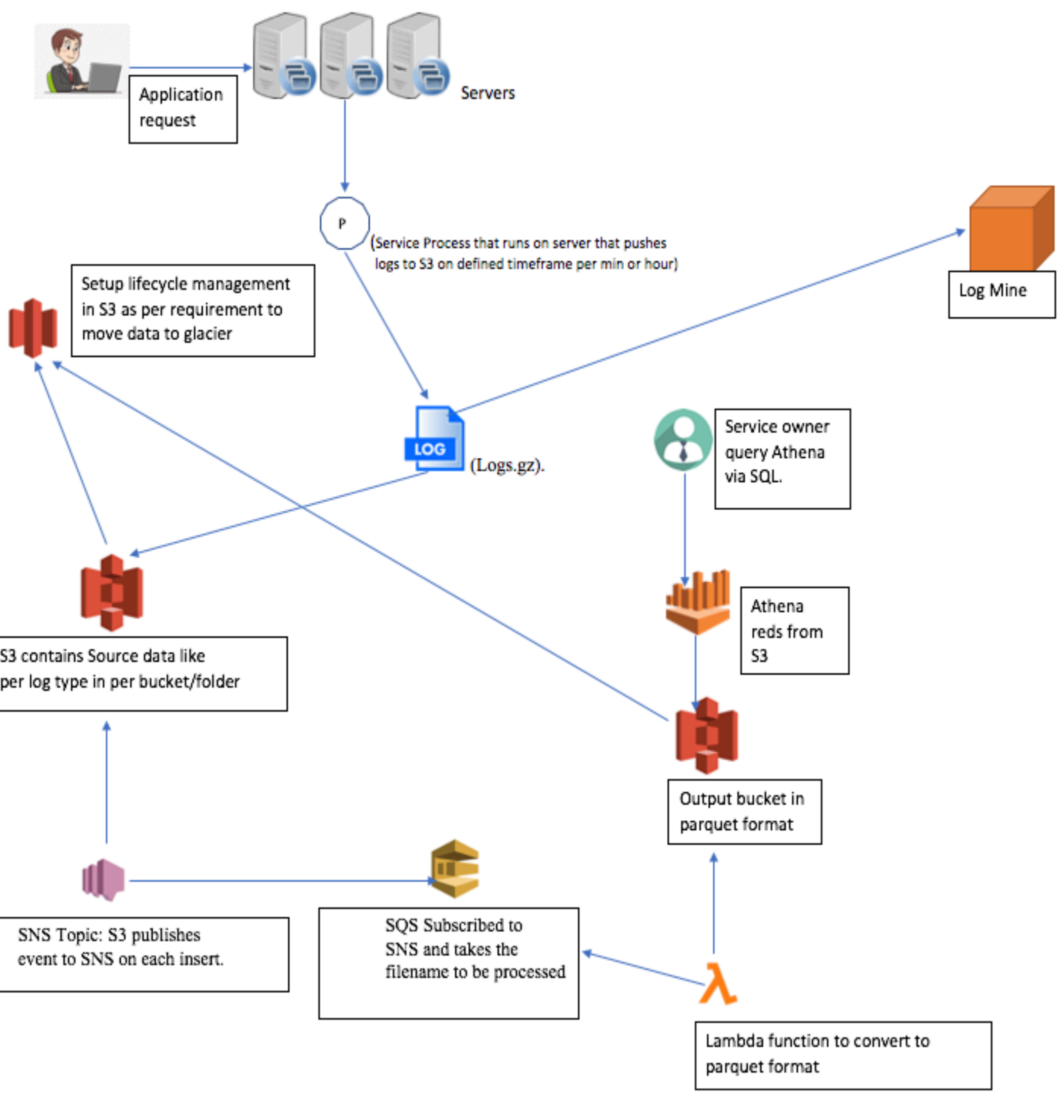
4. **Support Wide range of logs:**

As there are different kind of logs and varying format.

ARCHITECTURAL COMPONENTS

1. **User/client:** An entity from which request comes to the system.
2. **Host:** Hardware on which application is running
3. **S3**: is an object storage service that offers industry-leading scalability, data availability, security, performance provided by AWS.
4. **Broker process P or agent**: is a process that runs on the hosts and pushes logs to S3 bucket.
5. **Glacier:** is a secure, durable, and extremely low-cost cloud storage service for data archiving and long-term backup provided by AWS.
6. **Lambda:** Lets you run code without provisioning or managing servers. You pay only for the compute time you consume is a service provided by AWS.
7. **SNS:** is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and serverless applications.
8. **SQS:** is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications.
9. **Cloudwatch**: is a monitoring and management service that provides you with data and actionable insights to monitor your applications.
10. **Athena:** Athena is an interactive query service that makes it easy to analyze data in Amazon S3 using standard SQL.

PROPOSED ARCHITECTURE


Application request
Servers
P
(Service Process that runs on server that pushes logs to S3 on defined timeframe per min or hour)
Log Mine
Setup lifecycle management in S3 as per requirement to move data to glacier
LOG
(Logs.gz).
Service owner query Athena via SQL.
Athena reds from S3
S3 contains Source data like per log type in per bucket/folder
Output bucket in parquet format
SNS Topic: S3 publishes event to SNS on each insert.
SQS Subscribed to SNS and takes the filename to be processed
Lambda function to convert to parquet format

The broker process (P in diagram) which is deployed across the hosts serving the application are assigned an IAM role, providing it with the access to push the logs to S3 bucket on defined timeframe. The broker is also capable of pushing the logs on basis of size. In our experimental detail section, we intend to push the logs every minute, that is, broker process delivers one gzip-compressed file per host into an S3 bucket. It is that service owner can implements custom agents, that batches and send logs to S3. If larger is the file size, say if the object of upload is greater than 5gb multipart upload api is to be used. The only requirement is that it should delivers data into S3. The bucket split architecture can vary based on the application. In our case we intend to keep one bucket per region per log format. The sub folders are created in year/month/day/hour/minute structure.

S3 publishes event to the SNS topic for each file added to the bucket. SQS in the system must be subscribed to SNS as it is the one that queue the filename to be processed.

Lambda in architecture above is triggered by each entry in SQS. The lambda so generated intend to convert the raw log data into parquet format. As of now lambda function can run up to 15 mins, the conversion lambda intent to start processing as many raw files as possible and writing the content to single parquet file, as large parquet files are more optimal. Then the file is move to output S3 bucket. As the data is now available in S3 destination bucket in parquet format, Athena is used to query the logs. The S3 destination bucket split can be of one bucket per log format. Setting up S3 lifecycle to move data to glacier intend to reduce the storage cost, based on the retention needed by your service.

EXPERIMENT DETAILS

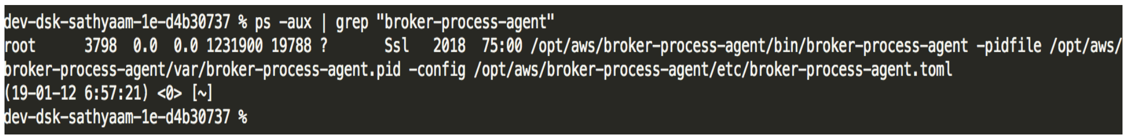
Below is sample of our log entry:

```
Mon Dec 10 05:00:09 2018 GMT  MyService      253.189.89.50  8739   172.45.30.48   80
        0.001596       0.001106       0.001332       200     200     0       3132    GET
        https://www.example.com/articles/556  HTTP/1.1       "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2
Safari/601.3.9"        DHE-RSA-AES128-SHA     TLSv1.2
[WARN]com.MyService.environment.platform.runtime.CoalescingQueryLogDataSourceDescripto
r: No PageType was set for this request. It is recommended the PageType is set for all
requests.
```

The broker agent to push the files to S3, the inputs required out for the agent in present in the toml file where we define

1. path towards the log folder
2. the interval within which data has to be pushed
3. credential details.

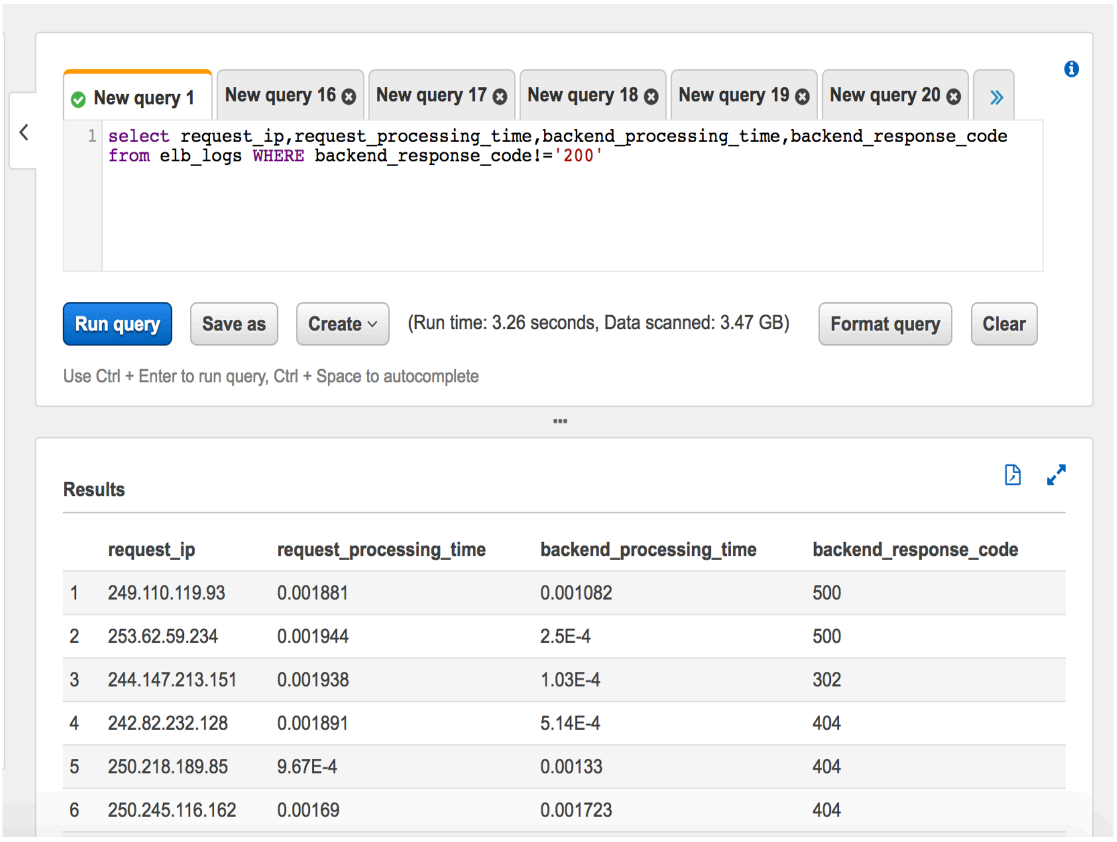
Below is the screenshot of Broker process running in the system:



```
dev-dsk-sathyaam-1e-d4b30737 % ps -aux | grep "broker-process-agent"
root      3798  0.0  0.0 1231900 19788 ?      Ssl   2018  75:00 /opt/aws/broker-process-agent/bin/broker-process-agent -pidfile /opt/aws/
broker-process-agent/var/broker-process-agent.pid -config /opt/aws/broker-process-agent/etc/broker-process-agent.toml
(19-01-12 6:57:21) <0> [~]
dev-dsk-sathyaam-1e-d4b30737 %
```

The Conversion lambda intend to use the defined schema for the processing the logs. The schema is build using the Technique mentioned in [1]LogMine, making the system capable of automatically recognizing the log pattern and hence reducing to the pain to create schema. We used Log Mine to handle the heterogeneity of logs in distributed system. In the below screenshot is sample of querying over all requests, where **backend_response_code** wasn't 200.

Sample Screenshot:



```
select request_ip,request_processing_time,backend_processing_time,backend_response_code
from elb_logs WHERE backend_response_code!='200'
```

(Run time: 3.26 seconds, Data scanned: 3.47 GB)

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

	request_ip	request_processing_time	backend_processing_time	backend_response_code
1	249.110.119.93	0.001881	0.001082	500
2	253.62.59.234	0.001944	2.5E-4	500
3	244.147.213.151	0.001938	1.03E-4	302
4	242.82.232.128	0.001891	5.14E-4	404
5	250.218.189.85	9.67E-4	0.00133	404
6	250.245.116.162	0.00169	0.001723	404

to obtain the output it took 3.26 seconds to scan over a 3.47 GB.

If we intend to search or analyze the logs say over in legacy way of scanning the file using grep/awk intend to take more time. As a sample below to obtain the same result from the log, that is to get the records where **backend_response_code** wasn't 200 using awk, it took 14.576s to scan 303MB file. Through above example we intend to say that it took 4x time more, out for a 11.45x time less data size, this is just over the case of scanning the small file and when it comes to real use case of searching or analyzing the Tera bytes of file size, the proposed architecture would be better.

```
[acbc32b81145:Desktop sathyaam$ du -ah log.csv
303M    log.csv
[acbc32b81145:Desktop sathyaam$ time awk -F, '$11 != 200' log.csv >> result_log.txt

real    0m14.576s
user    0m13.852s
sys     0m0.422s
acbc32b81145:Desktop sathyaam$ 
```

CONCLUSION

We have Proposed an architecture for log analyzing and archiving which is a serverless solution. It's capable of handling millions of heterogenous logs making it perfect solution for distributed system. As the components used in the system automatically takes care of its scalability and availability, reducing the infrastructure overhead.

This design also would provide better upper hand when building a customized analyzing solution like kibana on top of it. In the next phase we intend to make this system automatically detect anomalies in the log using machine learning.

REFERENCES

[1] https://www.cs.unm.edu/~mueen/Papers/LogMine.pdf

[2] https://aws.amazon.com/blogs/big-data/analyzing-data-in-s3-using-amazon-athena/

[3] https://static.googleusercontent.com/media/research.google.com /en//pubs/archive/36632.pdf

[4] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8416368

[5] https://docs.aws.amazon.com/aws-technical-content/latest/aws-overview/aws-overview.pdf