CS 441

COURSE PROJECT DOCUMENTATION

Real time Log files processing pipeline using Akka, Apache Kafka and Spark deployed on EC2 instances

Team Members

Sai Nadkarni (Leader)

Ankit Kumar Singh

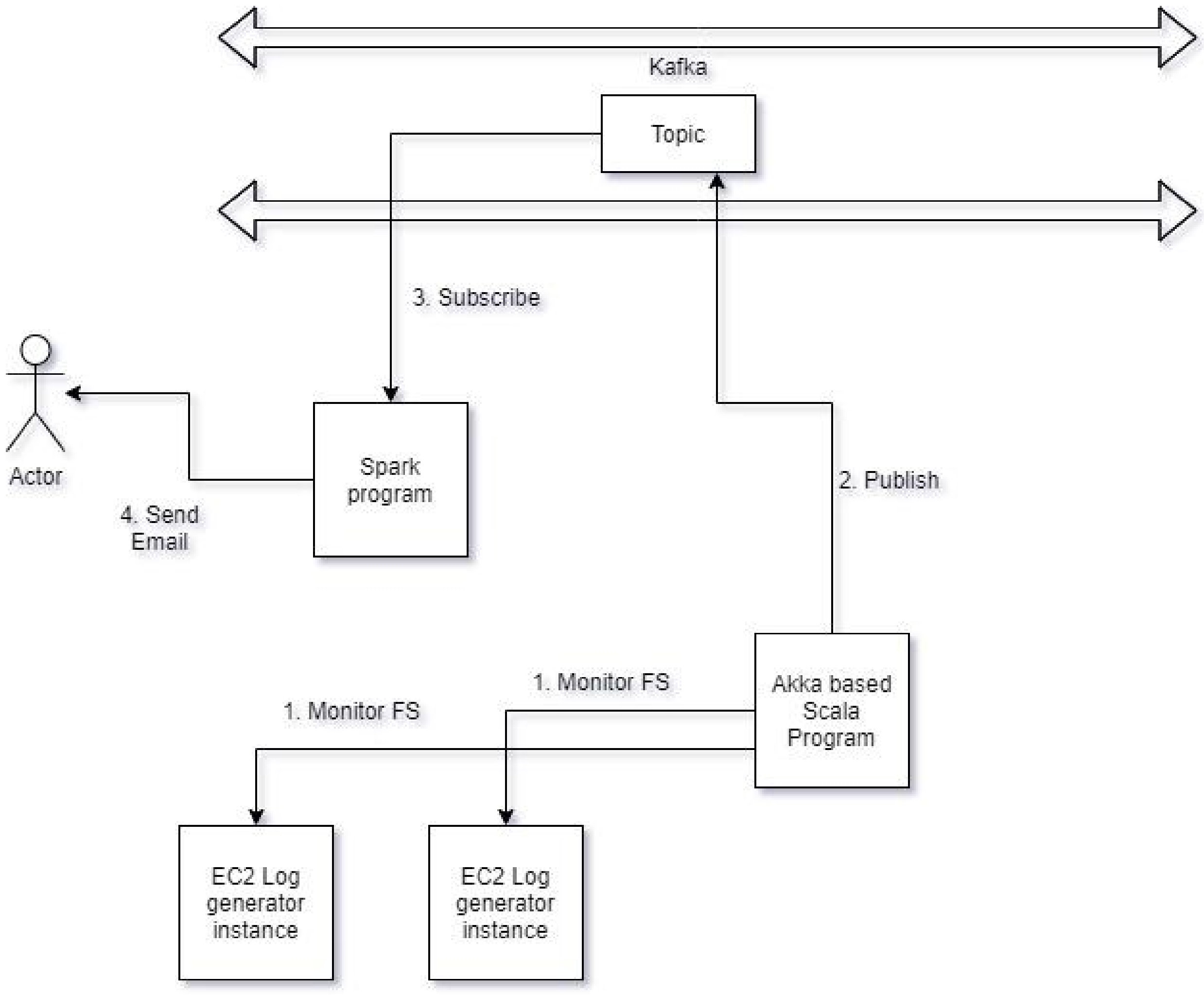
Sebastian Greczek

Sanchita Kapoor

Bhavana Nelakuditi

Basic Outline

This was our understanding going into the start of the project, and it helped outline the key areas of development and work subdivision.



Overview

In this project we have implemented a streaming data pipeline using cloud computing technologies like Amazon Elastic Compute Cloud (EC2) and by designing and implementing an actor - model service using Akka that is monitoring directories to which log files are added in real time and sends it to Spark using an event based service, Kafka. The project is deployed on AWS.

Development Environment

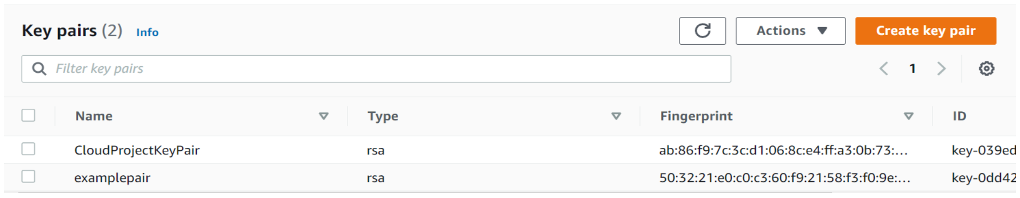
- Language: Scala v2.13.7
- IDE: IntelliJ IDEA 2021.3
- Build Tool: SBT v1.3
- Framework Used:
    - Akka 2.5.26
    - Akka HTTP 10.1.11
    - Kafka 2.8.1
    - Spark Core 3.0.3

Setting up the Environment

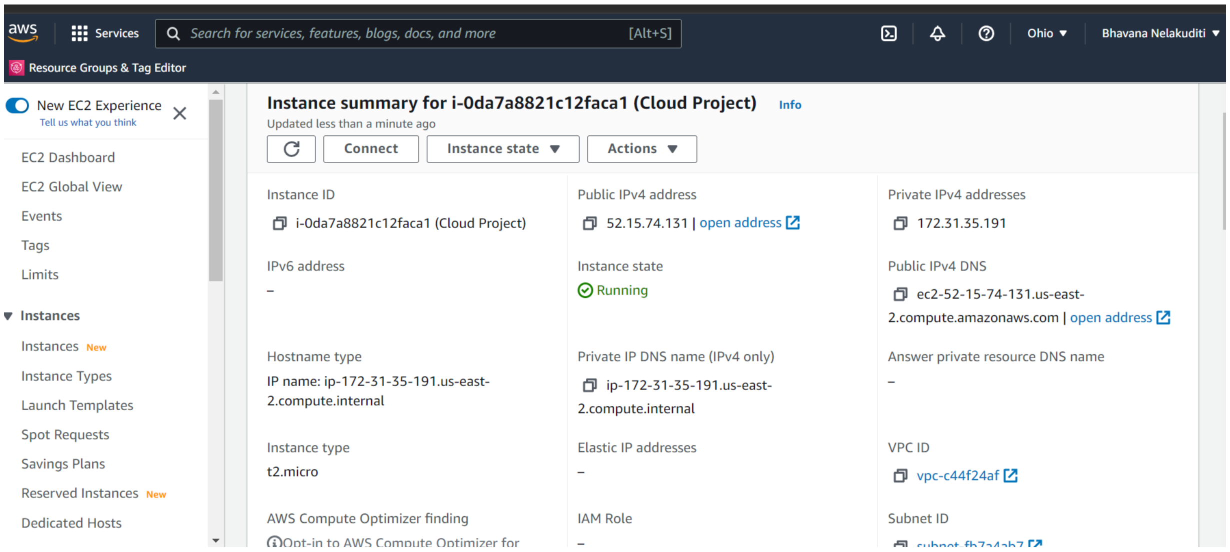
The Development Environment mentioned above is required to be set up in the EC2 instance before running the program. Different versions of the environment may lead to inconsistencies.

Setting up EC2:

Create EC2 instances with Amazon Linux 2, Key pairs and Security groups using AWS console. For our execution set up is as below.



Name	Type	Fingerprint	ID
CloudProjectKeyPair	rsa	ab:86:f9:7c:3c:d1:06:8c:e4:ff:a3:0b:73:...	key-039ed
examplepair	rsa	50:32:21:e0:c0:c3:60:f9:21:58:f3:f0:9e:...	key-0dd42


aws Services Search for services, features, blogs, docs, and more [Alt+S] Ohio Bhavana Nelakuditi

Resource Groups & Tag Editor

New EC2 Experience
Tell us what you think

EC2 Dashboard
EC2 Global View
Events
Tags
Limits

Instances
Instances New
Instance Types
Launch Templates
Spot Requests
Savings Plans
Reserved Instances New
Dedicated Hosts

Instance summary for i-0da7a8821c12faca1 (Cloud Project) Info
Updated less than a minute ago

Connect | Instance state | Actions

Instance ID	Public IPv4 address	Private IPv4 addresses
i-0da7a8821c12faca1 (Cloud Project)	52.15.74.131 | open address	172.31.35.191
IPv6 address	Instance state	Public IPv4 DNS
–	Running	ec2-52-15-74-131.us-east-2.compute.amazonaws.com | open address
Hostname type	Private IP DNS name (IPv4 only)	Answer private resource DNS name
IP name: ip-172-31-35-191.us-east-2.compute.internal	ip-172-31-35-191.us-east-2.compute.internal	–
Instance type	Elastic IP addresses	VPC ID
t2.micro	–	vpc-c44f24af
AWS Compute Optimizer finding	IAM Role	Subnet ID
Opt-in to AWS Compute Optimizer for	–	subnet-fb7a4ab7

Connect to the instances with SSH client and your private Key file using the command:

ssh -i <"PrivateKey.pem"> <Public DNS of the instance>

Setting up Git:

1. *sudo yum update -y*
2. *sudo yum install git -y*

Setting up SBT

SBT can be set up along with JDK using SDKMAN

1. *curl -s "https://get.sdkman.io" |bash*
2. *source "$HOME/ .sdkman/bin/sdkman-init.sh"*
3. *sdk install java $(sdk list java | grep -o "8\.[0-9]*\.[0-9]*\.hs-adpt" | head -1)*
4. *sdk install sbt*

Setting up Kafka

Download Kafka Binaries from here and unpack them:

1. *tar -xzf kafka_2.13-2.8.1.tgz*
2. *cd kafka_2.13-2.8.1*
3. *bin/zookeeper-server-start.sh config\zookeeper.properties*

4. *bin/kafka-server-start.sh config\server.properties*
5. *bin/kafka-topics.sh --create --topic logfilescraper --bootstrap-server localhost:9092*

Setting up Spark

Make sure Java and Python are set up since Apache Spark requires Java 8.

Download Spark from [here].

(Windows) Download [winutils.exe] file.

Choose a Spark release and the Pre-built for Apache Hadoop 2.7 package.

(Windows) Add the winutils.exe file to the /hadoop/bin folder after extracting spark.

Once Spark is installed go to the spark directory bin folder and run *spark-shell*

(Optional) For ensuring the email is sent out through the spark program if there are any errors during runtime about SMTP host, please look for java.security file in your java installation and comment this line of code out,

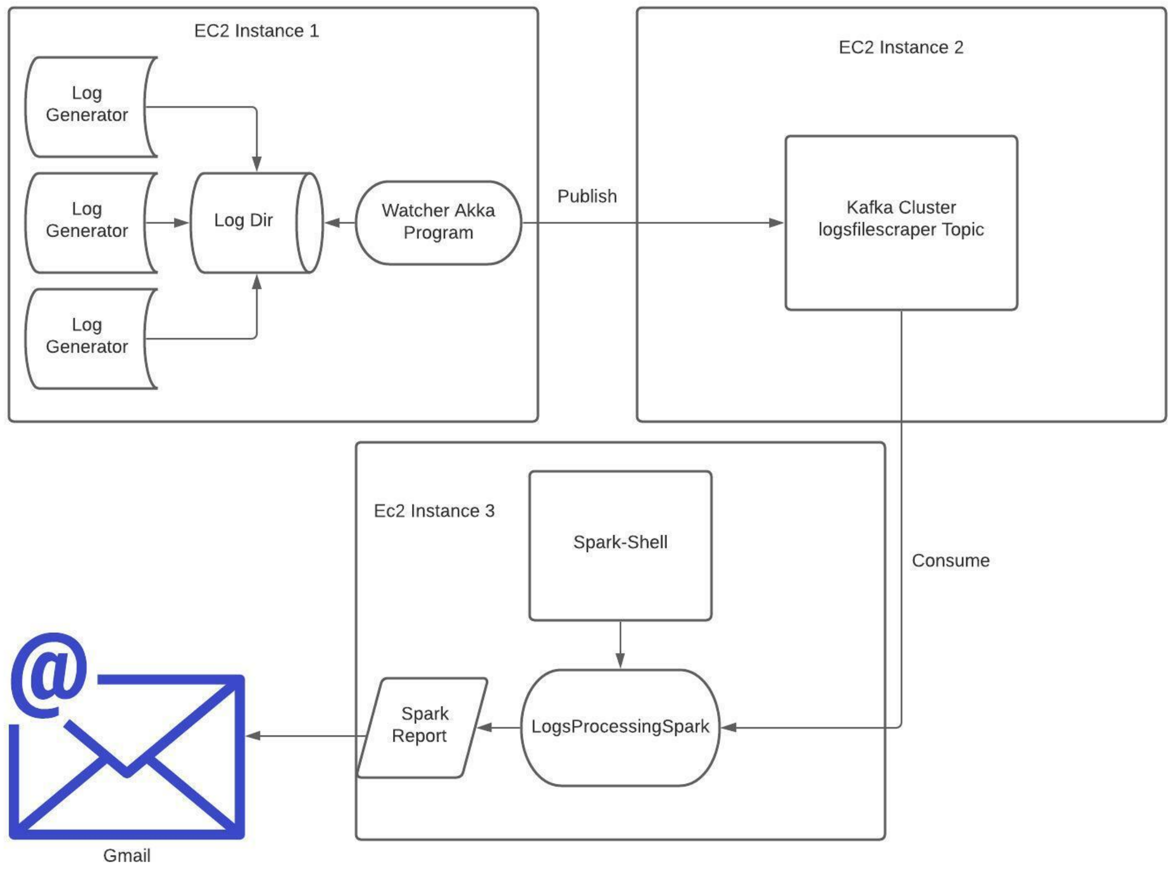
```
jdk.tls.disabledAlgorithms=SSLv3, TLSv1, TLSv1.1, RC4, DES, MD5withRSA, \
    DH keySize < 1024, EC keySize < 224, 3DES_EDE_CBC, anon, NULL, \
```

Steps to Run

- Clone this repository and navigate to the directory its been stored in each of the EC2 instances and run one module in each.
- Go to Watcher Directory and run
  - *sbt clean compile*
  - *sbt test*
  - *sbt run*
- Go to LogFileGenerator Directory and run
  - *sbt clean compile*
  - *sbt test*
  - *sbt run*

Code Structure and Flow:

Basic Pipeline:



Akka:

The Akka modules provide Akka Actors Model which allow a higher level of abstraction for writing concurrent and distributed programs. Akka Actors which are implemented in the Watcher directory are looking out for changes in the Log files generated by the Log File Generator. Java NIO is implemented in the Watcher program to watch for the changes in the Log Directory. Akka Actor Model checks the logs generated in real time and if it receives more than four (configurable amount) ERROR or WARN messages, these logs are sent to the Kafka Cluster.

Kafka:

Kafka is a distributed event streaming platform that lets us process, read, write and store events i.e messages across machines. Log messages are sent as events from the Akka actor system to the kafka cluster. We have Zookeeper running on this instance that keeps track of a node's status and maintains a list of topics. Zookeeper is used for service synchronisation in distributed systems and in Apache Kafka, Zookeeper is used to track the status of nodes in the Kafka cluster and maintain the list of Kafka topics and messages.

In Kafka, Events are stored and organised in topics which is like a file system with topics being the folders and events being the files.

Our events are under the topic "logfilescraper". This has been kept configurable in every scala program in the project which uses Kafka.

Kafka Broker, which works as a controller, allows consumers to fetch messages by topic and can create a kafka cluster by sharing information between each other directly or indirectly using zookeeper. We have our kafka broker running on this instance where the "Watcher" program acts as the producer and the "LogsProcessingSpark" Program acts as a consumer. "KafkaConsumer" and "KafkaProducer" directories were our starting point into writing a full fledged implementation, and experimenting with Kafka.

Spark:

Spark, which is an extension of the map/reduce computational model with considerable scheduling and performance improvements and works on the concept of datasets that can be reused several times, unlike key-value pairs in map/reduce. Apache Spark is a distributed processing system for big data workloads that makes use of in-memory caching and optimized query execution.

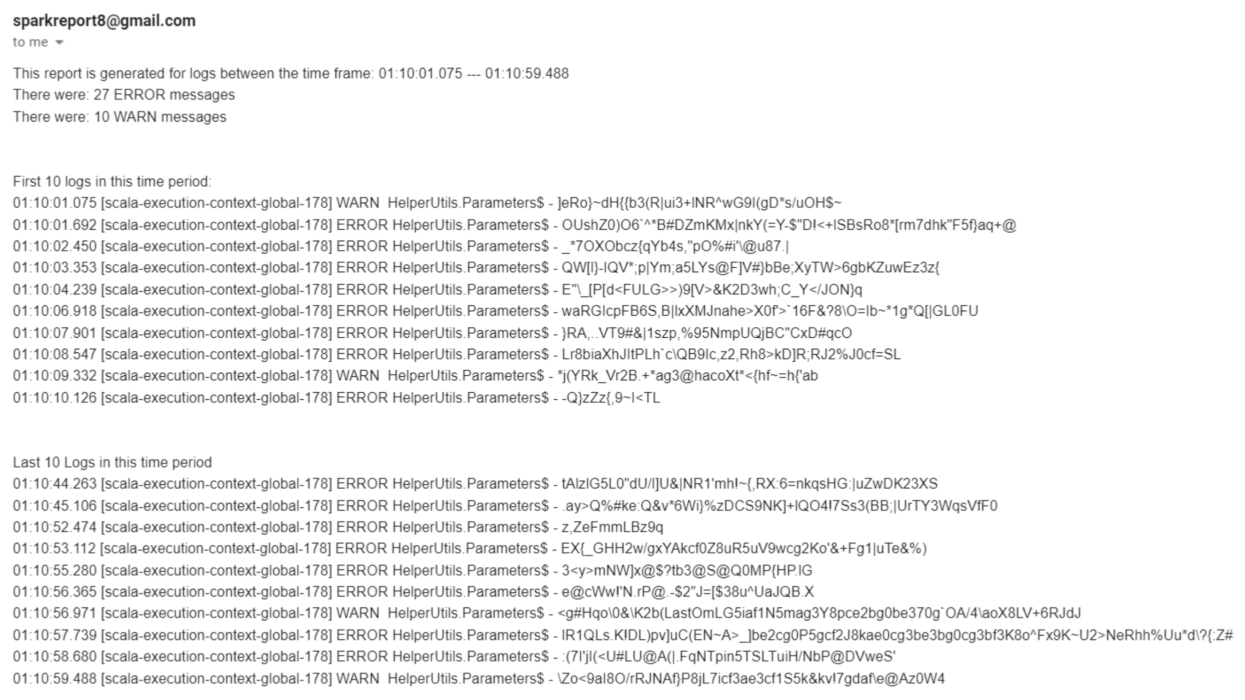
Spark shell running in the background provides the spark context that connects to the spark cluster. In the LogsProcessingSpark module, we utilize the spark context to process our logs. We count the number of ERROR and WARN log messages, provide the time frame of the logs and provide a message of the first (n) log messages and last (n) log messages.

This processing is bundled into a report and is sent as an email to the address “sparkreport8@gmail.com” created for the project using Javax Mail API.

Results:

To start off, we kept our LogGenerator instance configured to output messages for 2 minutes with an interval of between 200ms to 800ms. The probabilities of ERROR and WARN messages were kept a little higher than default, about 15-20% each. As soon as the execution finishes and the log file is updated, the pipeline is kicked into progress through the “Watcher” directories actor which is monitoring the log file directory. If the threshold is met, parsed log messages binned into 1 minute time windows are sent to Kafka, and the job here is done for “Watcher”. “LogsProcessingSpark” is constantly polling Kafka for updates, and it receives the log messages. They are processed and exported to a file, as well as a report sent out to any concerned stakeholders highlighting the key details.

The email sent to “sparkreport8@gmail.com” which contains the processed report of the log files generated in a time frame with ERROR and WARN messages:-



sparkreport8@gmail.com
to me

```
This report is generated for logs between the time frame: 01:10:01.075 --- 01:10:59.488
There were: 27 ERROR messages
There were: 10 WARN messages


First 10 logs in this time period:
01:10:01.075 [scala-execution-context-global-178] WARN  HelperUtils.Parameters$ - ]eRo}~dH{{b3(R|ui3+lNR^wG9l(gD*s/uOH$~
01:10:01.692 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - OUshZ0)O6`^*B#DZmKMx|nkY(=Y-$"D!<+ISBsRo8*[rm7dhk"F5f}aq+@
01:10:02.450 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - _*7OXObcz{qYb4s,"pO%#i'\@u87.|
01:10:03.353 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - QW[l}-lQV*;p|Ym;a5LYs@F]V#}bBe;XyTW>6gbKZuwEz3z{
01:10:04.239 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - E"\_[P[d<FULG>>)9[V>&K2D3wh;C_Y</JON}q
01:10:06.918 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - waRGIcpFB6S,B|lxXMJnahe>X0f'>`16F&?8\O=Ib~*1g*Q[|GL0FU
01:10:07.901 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - }RA,..VT9#&|1szp,%95NmpUQjBC"CxD#qcO
01:10:08.547 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - Lr8biaXhJltPLh`c\QB9Ic,z2,Rh8>kD]R;RJ2%J0cf=SL
01:10:09.332 [scala-execution-context-global-178] WARN  HelperUtils.Parameters$ - *j(YRk_Vr2B.+*ag3@hacoXt*<{hf~=h{'ab
01:10:10.126 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - -Q}zZz{,9~I<TL


Last 10 Logs in this time period
01:10:44.263 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - tAlzIG5L0"dU/l]U&|NR1'mh!~{,RX:6=nkqsHG:|uZwDK23XS
01:10:45.106 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - .ay>Q%#ke:Q&v*6Wi}%zDCS9NK]+lQO4!7Ss3(BB;|UrTY3WqsVfF0
01:10:52.474 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - z,ZeFmmLBz9q
01:10:53.112 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - EX{_GHH2w/gxYAkcf0Z8uR5uV9wcg2Ko'&+Fg1|uTe&%)
01:10:55.280 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - 3<y>mNW]x@$?tb3@S@Q0MP{HP.IG
01:10:56.365 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - e@cWw!'N.rP@.-$2"J=[$38u^UaJQB.X
01:10:56.971 [scala-execution-context-global-178] WARN  HelperUtils.Parameters$ - <g#Hqo\0&\K2b(LastOmLG5iaf1N5mag3Y8pce2bg0be370g`OA/4\aoX8LV+6RJdJ
01:10:57.739 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - lR1QLs.K!DL)pv]uC(EN~A>_]be2cg0P5gcf2J8kae0cg3be3bg0cg3bf3K8o^Fx9K~U2>NeRhh%Uu*d\?{:Z#
01:10:58.680 [scala-execution-context-global-178] ERROR HelperUtils.Parameters$ - :(7l'jl(<U#LU@A(|.FqNTpin5TSLTuiH/NbP@DVweS'
01:10:59.488 [scala-execution-context-global-178] WARN  HelperUtils.Parameters$ - \Zo<9al8O/rRJNAf}P8jL7icf3ae3cf1S5k&kv!7gdaf\e@Az0W4
```

In retrospect, we are pleased with our pipeline and how quick and efficient it was in dealing with our set thresholds. The challenges faced along the way helped guide our understanding towards the difficulties faced in setting up a cohesive distributed system.

Limitations:-

1. Our implementation of Javax Mail in some cases does not work due to some SMTP host error. To workaround this issue, we had to comment a certain line of code out in the java.security file in our java installation on our machines. This error only happened on our Windows machines, this was not needed on our Amazon Linux 2 Image on EC2.
2. On Linux instances, our LogGenerator would update the log file with each new log message instead of doing it after a certain time interval or after execution. This led to message repetition and duplication.

AWS Deployment video is hosted on youtube at **https://youtu.be/vj4cN4khn8A**

Future Work:

1. Address our implementation limitations.
2. Use Machine Learning in our Spark processing framework to deliver tailored results and generate meaningful insights.
3. Implement batching in Spark program to gather multiple time windows together instead of separate reports for each bin.

References:

https://www.linuxtutorial.co.uk/install-sbt-on-amazon-linux/

https://cloudaffaire.com/how-to-install-git-in-aws-ec2-instance/

https://www.scala-sbt.org/1.x/docs/Installing-sbt-on-Linux.html

https://sdkman.io/

https://phoenixnap.com/kb/install-spark-on-windows-10

https://github.com/cdarlint/winutils

https://doc.akka.io/docs/akka/current/typed/guide/introduction.html

https://jaceklaskowski.gitbooks.io/apache-kafka/content/kafka-brokers.html

https://databricks.com/spark/about

Mark Grechanik, Cloud Computing Theory and Practice, 2020