Claudio Fahey's Big Data Blog

Real-Time Global Anomaly Detection in IoT with EMC Elastic Cloud Storage (ECS)

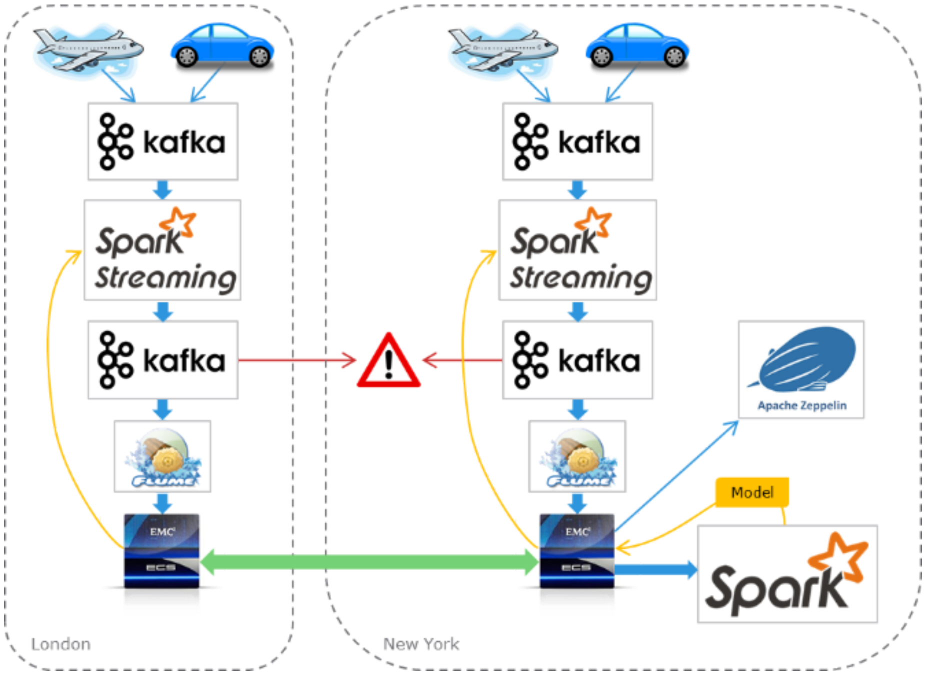
Posted by Claudio Fahey in Claudio Fahey's Big Data Blog on Mar 30, 2016 3:45:54 PM

Part 1

This is Part 1 of this blog series. Additional details are provided in Part 2.

Introduction

In this blog post, I'll describe an architecture for performing near real-time streaming anomaly detection on IoT data. We'll use Apache Kafka, Apache Spark, Apache Flume, Apache Zeppelin, and EMC Elastic Cloud Storage (ECS) to build a scalable, geo-distributed, and highly reliable system. The diagram below shows the key components of our system.



To be specific by what is meant by "near real-time" in this context, this system will generate alerts within about 5 seconds after an anomalous message is generated by the device.

The Internet of Things

We start with IoT devices such as smart phones, thermostats, cars, and industrial machines. These may be all over the world and will generally send real-time streaming data to a nearby web service accessible over the Internet. To reduce the traffic between continents (or perhaps to comply with privacy laws), we'll have data centers in multiple regions throughout the world. Each data center will host multiple instances of this web service. When the web service receives the message, it may perform some very basic tasks (e.g. authentication and authorization) and then it simply forwards the message to a Kafka topic.

Apache Kafka

Apache Kafka, in short, is a distributed and reliable message bus. Each data center will have its own Apache Kafka cluster to ensure functionality in the event of a communication failure between data centers. Apache Kafka will replicate the messages among its nodes to ensure that they are protected from hardware or other failures.

Spark Streaming - Real-Time Anomaly Detection

To perform the actual anomaly detection (or many other predictive machine learning tasks), we'll use Spark Streaming. Spark Streaming functions as a complex event processor. Unlike a traditional rules engine, it can be used to perform complex data enrichment and evaluate state-of-the-art machine learning models such as Random Forests and artificial neural networks.

Our Spark Streaming job will do quite a few things.

1. First, our Spark Streaming job will pull the messages from our Kafka topic and create mini-batches of messages every few seconds.
2. For each mini-batch, our job will enrich the data as needed (e.g. join with other data sets, perform geo-location, calculate fields).
3. We will want the enriched data to be persisted to reliable storage. Although Spark can write directly to Hadoop-compatible storage, doing so would result in many small files (one per batch per partition). To avoid this, the Spark Streaming job will send the enriched data to a different Kafka topic where it will be consumed by Flume. Flume will then reliably create very large files, flushing them periodically to ensure the file system has the latest data. We'll use ECS for this purpose because it gives us protection from the failure of a single node as well as the failure of an entire data center.
4. The enriched data will then be transformed into features (e.g. convert all categorical data and words into real numbers).
5. For each message, the features will be used as the input into our machine learning model and the output will reflect whether the message is predicted to be an anomaly. (The machine learning model will be trained separately. Keep reading.)
6. If an anomaly is predicted, then the job will send an alert. This can simply be an alert message sent to a different Kafka topic that an outside alerting system monitors.

EMC Elastic Cloud Storage (ECS)

Now we have our enriched data in multiple files in a single geo-distributed ECS bucket. The primary copy of each object (file) will physically remain within the data center where it originated. Other data centers throughput the world will contain exactly enough parts of the erasure-coded objects so that an object can be rebuilt in the event of a failure of any single data center. Specifically, Reed-Solomon erasure coding is used within a data center and parity (XOR) is used between data centers. This greatly reduces the storage requirement but retains a high degree of data protection. Note that the geo-distributed protection is an asychronous process so it is assumed that a few seconds of data loss is an acceptable trade-off for having a lower write latency.

If you were to browse this ECS bucket (e.g. hadoop fs -ls), you would see files created from the Spark Streaming job running at each data center. A request to an ECS node in one data center to read a file whose primary copy is at another data center will result in a read across your WAN. However, subsequent requests for the same file will often use a locally cached copy of the object eliminating the full WAN read. In any case, however, ECS will guarantee that you are reading the latest version of the file.

These features of ECS make it extremely simple to run high-performance analytics jobs across the entire global data set. You only need to point your job to read the files in a single directory and ECS will take care of the WAN transfer, local caching, cache invalidation, consistency, disaster recovery, and high availability.

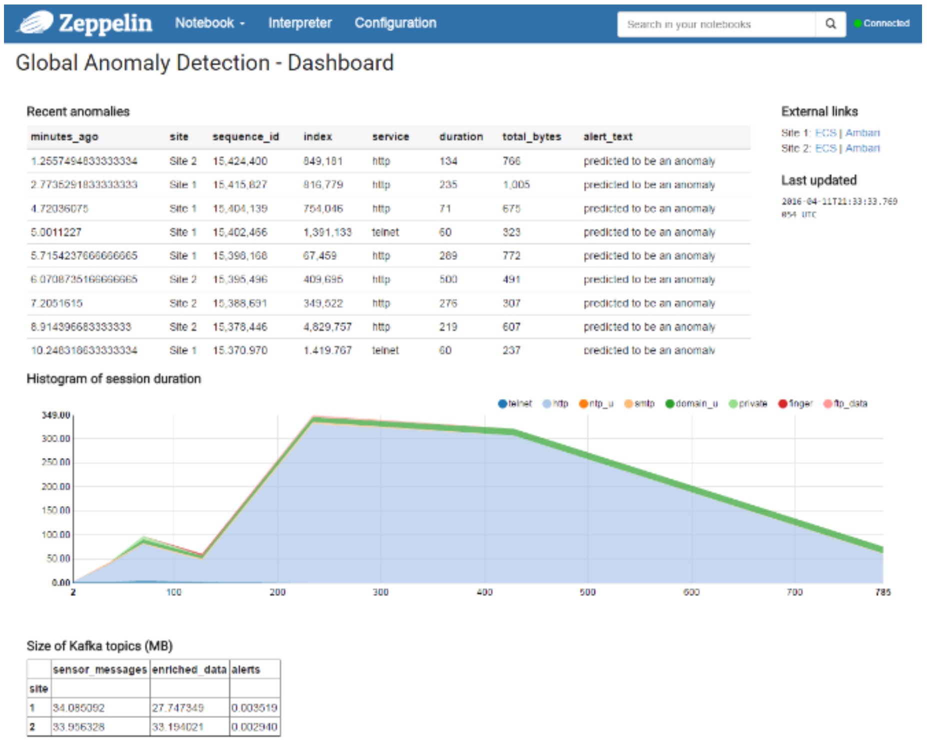
Spark - Machine Learning Model Training

Back to anomaly detection, we will use the entire global data set to train our machine learning model to detect anomalies. To do this, we'll create a Spark batch job that executes at a single data center to do the following.

1. Read the enriched data that was placed in our ECS bucket by the Flume job at each data center. Since ECS guarantees a consistent view of the data, this Spark job will see the entire data set up to the most recent flush by Flume.
2. Train our machine learning model. A single model will be created that uses the data collected by all data centers. This step may also include cross validation and hyperparameter tuning.
3. Save our model to files on ECS. Depending on the algorithm and hyperparameters we choose, the size of our model on disk may be a few KB to several GB.
4. Inform each of our Spark Streaming jobs to use the newly trained model saved on ECS.

Dashboards and Exploration with Zeppelin

Apache Zeppelin is a web-based notebook that enables interactive data analytics. It can also be used to easily create some very nice dashboards. Below we can see a list of the 1000 most recent alerts, a histogram based on the alerts, and some Kafka metrics from each site.



Global Anomaly Detection - Dashboard

Recent anomalies

minutes_ago	site	sequence_id	index	service	duration	total_bytes	alert_text
1.2557494833333334	Site 2	15,424,400	849,181	http	134	766	predicted to be an anomaly
2.7735291833333333	Site 1	15,415,827	816,779	http	235	1,005	predicted to be an anomaly
4.72036075	Site 1	15,404,139	754,046	http	71	675	predicted to be an anomaly
5.0011227	Site 1	15,402,466	1,391,133	telnet	60	323	predicted to be an anomaly
5.7154237666666665	Site 1	15,398,168	67,459	http	289	772	predicted to be an anomaly
6.0708735166666665	Site 2	15,395,496	409,695	http	500	491	predicted to be an anomaly
7.2051615	Site 2	15,388,691	349,522	http	276	307	predicted to be an anomaly
8.914396683333333	Site 2	15,378,446	4,829,757	http	219	607	predicted to be an anomaly
10.248318633333334	Site 1	15,370,970	1,419,767	telnet	60	237	predicted to be an anomaly

External links

Site 1: ECS | Ambari
Site 2: ECS | Ambari

Last updated

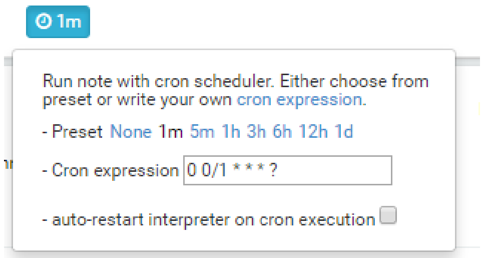
2016-04-11T21:33:33.769854 UTC

Histogram of session duration

Size of Kafka topics (MB)

	sensor_messages	enriched_data	alerts
site			
1	34.085092	27.747349	0.003519
2	33.956328	33.194021	0.002940

The dashboard can be configured to refresh itself every minute.



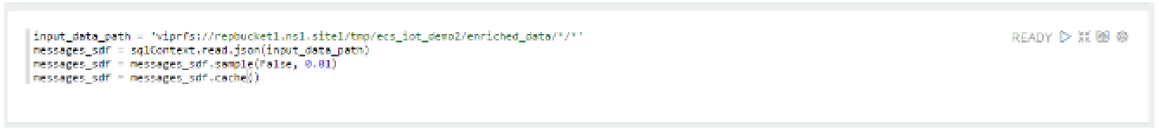
Run note with cron scheduler. Either choose from preset or write your own cron expression.

- Preset None 1m 5m 1h 3h 6h 12h 1d

- Cron expression 0 0/1 * * * ?

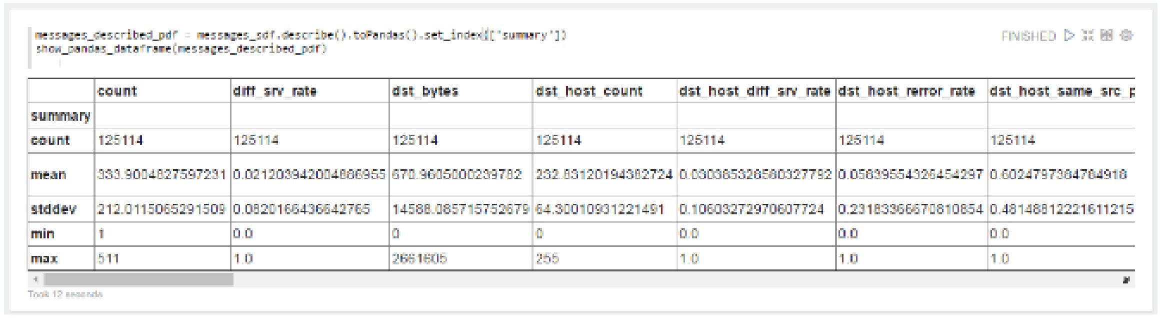
- auto-restart interpreter on cron execution

Zeppelin integrates directly with Spark and PySpark and it can be used interactively. The following will load a sample of 1% of the enriched data on ECS and cache it in Spark for fast interactive querying.



```
input_data_path = 'viprfs://repbucket1.ns1.site1/tmp/ecs_iot_demo2/enriched_data/*/*'
messages_sdf = sqlContext.read.json(input_data_path)
messages_sdf = messages_sdf.sample(False, 0.01)
messages_sdf = messages_sdf.cache()
```

I can also show some statistics for each of the numeric columns in the enriched data.

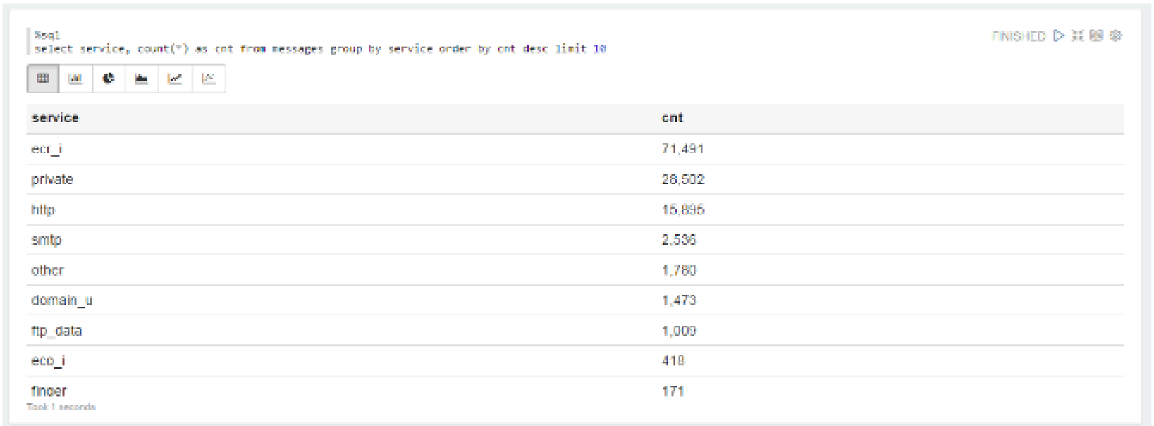


```
messages_described_pdf = messages_sdf.describe().toPandas().set_index(['summary'])
show_pandas_dataframe(messages_described_pdf)
```

	count	diff_srv_rate	dst_bytes	dst_host_count	dst_host_diff_srv_rate	dst_host_rerror_rate	dst_host_same_src_p
summary							
count	125114	125114	125114	125114	125114	125114	125114
mean	333.9004827597231	0.021203942004886955	670.9605000239782	232.83120194382724	0.030385328580327792	0.05839554326454297	0.6024797384784918
stddev	212.0115065291509	0.0820166436642765	14588.085715752679	64.30010931221491	0.10603272970607724	0.23183366670810854	0.48148812221611215
min	1	0.0	0	0	0.0	0.0	0.0
max	511	1.0	2661605	255	1.0	1.0	1.0

Took 12 seconds

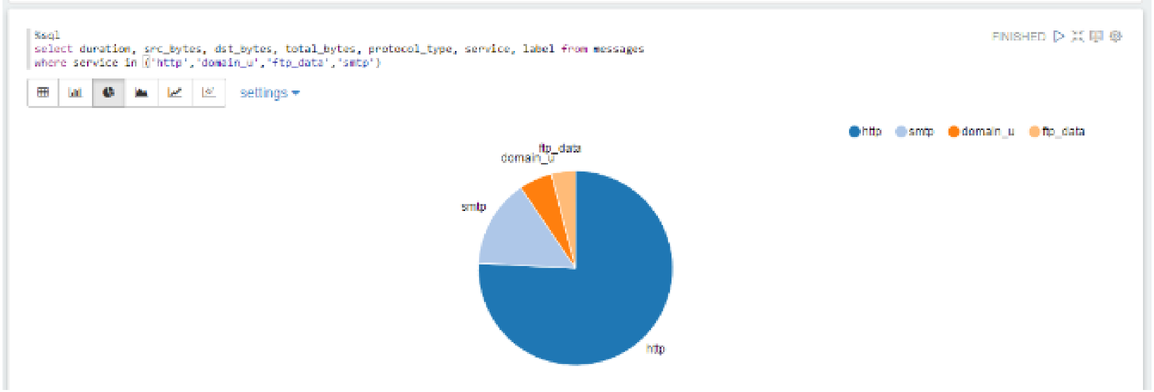
I can also write SQL and view the results in a table or in a variety of charts.



```
%sql
select service, count(*) as cnt from messages group by service order by cnt desc limit 10
```

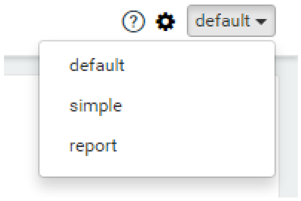
service	cnt
ecr_i	71,491
private	28,502
http	16,896
smtp	2,536
other	1,780
domain_u	1,473
ftp_data	1,009
eco_i	418
finger	171

Took 1 seconds



```
%sql
select duration, src_bytes, dst_bytes, total_bytes, protocol_type, service, label from messages
where service in ('http','domain_u','ftp_data','smtp')
```

And finally, it's extremely simple to convert any interactive notebook into a dashboard by changing the view to "report."



Conclusion

That's it for the core components of a typical anomaly detection pipeline for global IoT data. You may have noticed that there wasn't very much to deal with to make it "global." There was no need to develop scripts and processes around using DistCp to copy files from one data center to another (and back). We didn't need to worry about caching data locally or the harder problem of cache invalidation. We didn't need build complex disaster recovery / high availability processes, as least as far as the data storage is concerned. ECS handled these tasks for us allowing us to focus on more interesting things.

Although this blog post is focused on anomaly detection, most of this architecture is general purpose and can be used for a wide variety of machine learning and other global Big Data use cases.

Part 2

If you want to learn more, there are a lot more details in Part 2 of this blog series.

Part 3

See Part 3 of this blog series.

4389 Views Tags: flume , streaming , machine_learning , hadoop , analytics , kafka , spark , iot , ecs , zeppelin

Average User Rating

(8 ratings)

My Rating:

1 Comment

B. Scott Cassell Apr 14, 2016 11:26 AM

Wow! This is amazing Claudio. Excellent blog and research. This type of practical solutions to a real world business application really helps clarify the power of ECS in leveraging geo-distributed data. More please!

Like (0)