Lambda Architecture for Batch and Real-Time Processing on AWS with Spark Streaming and Spark SQL

May 2015

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which is subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Contents

Abstract

Enterprises need to grow and manage their global computing infrastructures rapidly and efficiently while simultaneously optimizing and managing capital costs and expenses. Today, many architects and developers are looking for cloud solutions that integrate batch processing with real-time data processing. Lambda architecture is a data-processing design pattern to handle massive quantities of data and integrate batch and real-time processing within a single framework. This design pattern can be implemented on AWS.

Lambda architecture is distinct from and should not be confused with the "AWS Lambda" compute service. It is a software pattern that unifies real-time processing with batch processing within a single framework.

This white paper is intended for Amazon Web Services (AWS) Partner Network (APN) members, IT infrastructure decision-makers, and administrators. In this paper, you will learn which artifacts to use and how to configure infrastructure details, such as compute instances, bootstrap actions, storage, security, and networking. After reading it, you should have a good idea of how to set up and deploy the components of a typical Lambda architecture on AWS.

Introduction

When processing large amounts of semi-structured data, there is always a delay between the point when data is collected and its availability in dashboards. Often the delay results from the need to validate or at least identify coarse data. In some cases, however, being able to react immediately to new data is more important than being 100 percent certain of the data's validity.

The AWS tool most frequently used to deal with large volumes of semi-structured or unstructured data is Amazon Elastic MapReduce (Amazon EMR). Stream or real-time processing, the processing of a constant flux of data, in real time, is possible with a Lambda Architecture solution that includes Amazon Kinesis, Amazon Simple Storage Service (Amazon S3), Spark Streaming, and Spark SQL on top of an Amazon EMR cluster.

A Lambda Architecture approach mixes both batch and stream (real-time) data processing. It is divided into three processing layers: the batch layer, serving layer, and speed layer, as shown in the following figure.

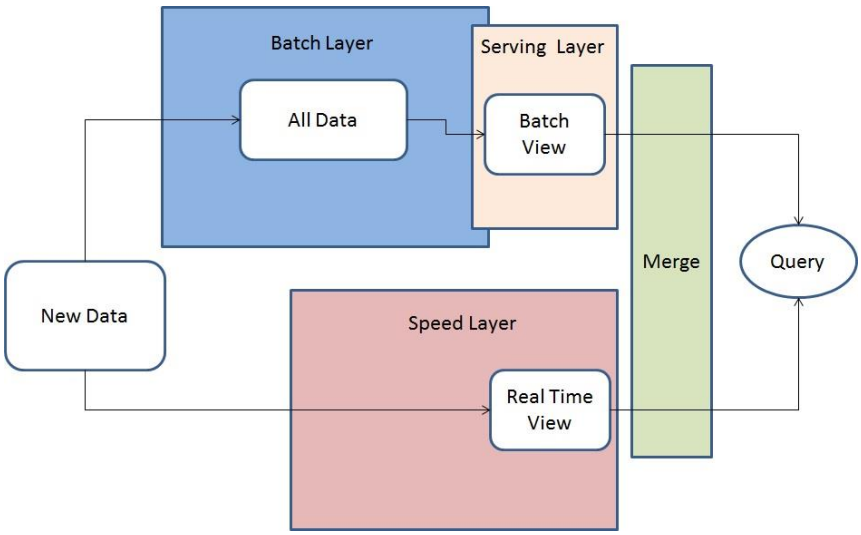

Figure 1: Lambda Architecture

All new data (in JSON format) is sent both to the batch layer (Amazon Kinesis, Amazon EMR, Amazon S3) and to the speed layer (Amazon Kinesis, Amazon EMR, Spark Streaming). In the batch layer, new data is appended to the master data set, a set of files that contains information that is not derived from any other information. It is an immutable, append-only set of data stored in an Amazon S3 bucket. The batch layer precomputes query functions continuously in a `while (true)` loop. This process is analogous to extract, transform, and load (ETL) processing performed on Amazon EMR by Spark SQL.

Batch layer: The results of the batch layer are called batch views and are stored on Amazon S3 as a tab-separated value file.

Serving layer: This layer indexes the batch views produced by the batch layer. Basically, the serving layer is a scalable database that swaps in new batch views as they become available. Due to the latency of the batch layer, the results from the serving layer are always out-of-date by a few hours.

Speed layer: The speed layer compensates for the high latency of updates to the serving layer. It uses a fast, in-memory Spark engine to process data that has not

been processed in the last batch of the batch layer. This layer produces the real-time views that are always up-to-date; it stores them in databases for both read and write operations. The speed layer is more complex than the batch layer due to the fact that the real-time views are continuously discarded as data makes its way through the batch and serving layers.

Queries are resolved by merging the batch and real-time views, which you can do with a Spark application, AWS Services like DynamoDB, RDS, Redshift, Amazon EMR running HBase, and open-source tools. Because the batch views are always recomputed completely, it is therefore possible to adjust the granularity of the data in function of its age. Another benefit of recomputing data from scratch is that if the batch or real-time views are corrupt, as the main data set is append-only, it is easy to restart and recover from the unstable state. Lastly, the end user can always query the latest version of the data, which is available from the speed layer.

One well-developed approach to merging real-time data with historical data is to use Hadoop and Apache Storm together. Each engine produces a table in the serving database; applications can issue a query, which merges those results, as shown in the following figure.

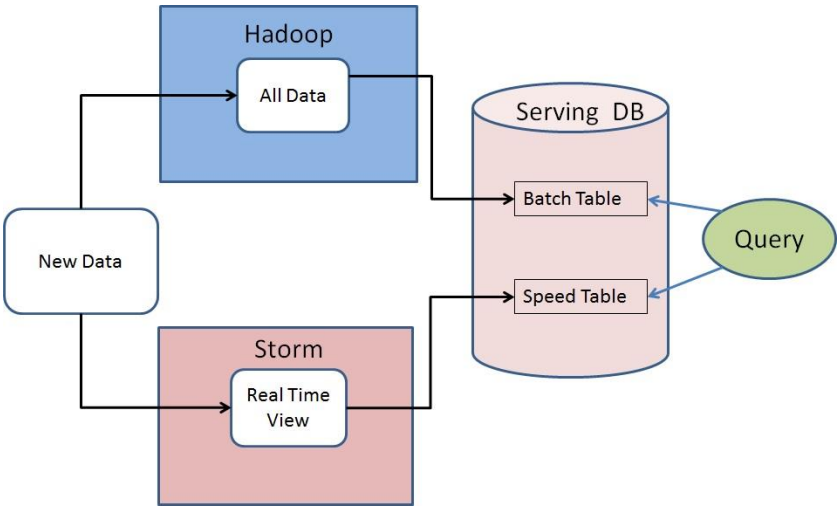

Figure 1: Traditional Approach to Lambda Architecture

The downside to traditional Lambda Architecture is that you must maintain the code required to produce the query result in two, complex, distributed systems.

Components of Lambda Architecture on AWS

Amazon EMR simplifies big data processing, providing a managed Hadoop framework that makes it easy, fast, and cost-effective for you to distribute and process vast amounts of your data across dynamically scalable Amazon EC2 instances. The application can be written in high-level programming languages like Java, Scala, or Python.

The following figure shows a Lambda Architecture on AWS.

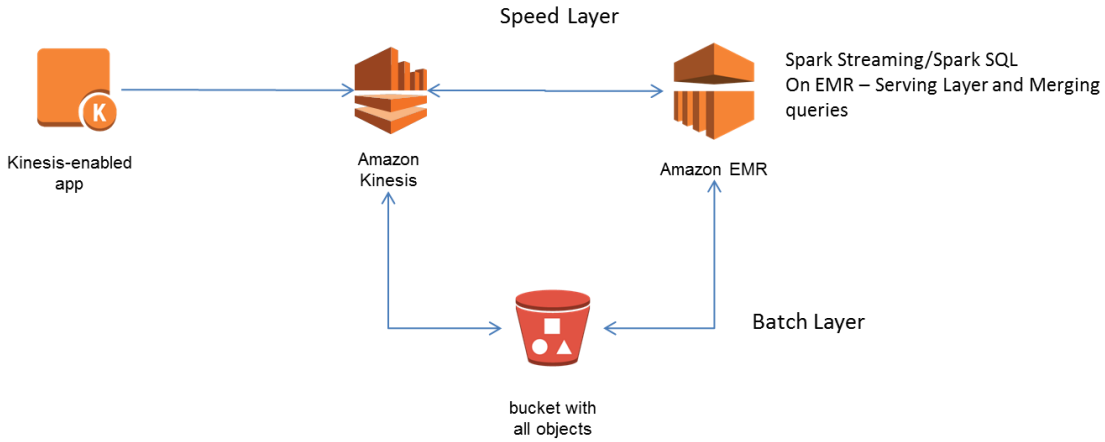

Figure 3: Lambda Architecture on AWS

Real-time data flows to the system through Amazon Kinesis. It then gets aggregated and stored in an Amazon S3 bucket. Thus, all historical data resides on Amazon S3.

This suggested architecture includes a speed layer with Spark Streaming on an Amazon EMR cluster, which consumes data from Amazon Kinesis streams. The batch layer with Spark SQL on an Amazon EMR cluster consumes data from Amazon S3. Both of these components are part of the same code base, which can be invoked as required, thus reducing the overhead of maintaining multiple code bases.

This section describes each component of the Lambda Architecture.

Amazon Kinesis and KCL

Amazon Kinesis is a fully managed service that can store and process terabytes of streaming data. With Amazon Kinesis, developers can continuously capture data from hundreds of thousands of sources, including website clickstreams, financial transaction data, social media feeds, server logs, and more. You can use the Amazon Kinesis Client Library (KCL) to develop stream-processing applications. Those applications can, in turn, consume the Amazon Kinesis streams and take action on real-time data to power real-time dashboards, implement real-time business logic, generate alerts, and ingest data to other services, such as Amazon S3, Amazon Redshift, and more.

KCL handles complex issues like adapting to changes in stream volume, load balancing streaming data, coordinating distributed worker services, and processing data with fault tolerance. KCL helps developers integrate Amazon Kinesis with services such as Amazon DynamoDB, Amazon Redshift, and Amazon S3.

Amazon EMR

Amazon EMR provides users with Hadoop, an open-source framework that you can use to distribute and process data across a resizable cluster of Amazon Elastic Compute (Amazon EC2) instances. Amazon EMR also can run distributed frameworks, such as Apache Spark, which provides an advanced execution engine for fast in-memory computing.

Amazon S3

Amazon S3 provides developers with secure and durable object storage.

Spark Streaming

Spark Streaming, an extension of the Spark API, can be installed on an Amazon EMR cluster through bootstrapping. It enables fault-tolerant stream processing of live-data streams. Data is ingested from Amazon Kinesis and processed using complex algorithms. The machine learning and graph-processing algorithms in Spark can be applied on data streams. Processed data can be pushed out to dashboards, file systems like HDFS, and Amazon S3.

Spark SQL

Like Spark Streaming, Spark SQL is also an extension of the Spark API and can be installed on Amazon EMR cluster through bootstrapping. It allows relational queries expressed in SQL or HiveQL to be executed in Spark code with integrated APIs in Python, Scala, or Java. This integration means you can run SQL queries alongside complex analytical algorithms.

Deploying Lambda Architecture on AWS

The first deployment step is to set up an Amazon EMR cluster with the Spark API. Run the following command, which will create a three-node Amazon EMR cluster, install Spark, and create an SBT package.

```
./elastic-mapreduce --create --alive --name SparkCluster \
--hive-interactive --instance-type m3.xlarge \
--instance-count 3 --ami-version 3.2.1 \
--bootstrap-action
"s3://support.elasticmapreduce/spark/install-spark" \
--bootstrap-name "Install Spark"
--jar s3://elasticmapreduce/libs/script-runner/script-
runner.jar \
--args "s3://elasticmapreduce.bootstrapactions/sbt/install-
sbt"

Or

aws emr create-cluster --name SparkCluster \
--ami-version 3.2 --instance-type m3.xlarge --instance-
count 3 \
--ec2-attributes KeyName=<<MYKEY>> \
--bootstrap-actions
Path=s3://support.elasticmapreduce/spark/install-
spark,Name=Install_Spark \
--steps
Name=Install_Sbt,Jar=s3://elasticmapreduce/libs/script-
runner/script-
runner.jar,ActionOnFailure=CONTINUE,Args=s3://elasticmapred
uce.bootstrapactions/sbt/install-sbt \
```

```
--termination-protected
```

Application code can be found here. Download the code and build the package on the Amazon EMR master node.

```
mkdir ~/workspace
cd ~/workspace
wget https://s3.amazonaws.com/chayel-
public/LambdaArchitecturePattern.zip
unzip LambdaArchitecturePattern.zip
sbt package
```

Streaming Data to Amazon Kinesis

An Amazon Kinesis producer is generating JSON data and pushing it to Amazon Kinesis using the KCL.

```
// Download JAR:
wget http://chayel-
public.s3.amazonaws.com/KinesisProducer.jar
java –jar KinesisProducer.jar //Publishes 10K events to
Kinesis stream myStream
```

Here are the JSON streams generated by the producer:

```
{"zipcode":95126,"ProductName":"product2","price":16,"times
tamp":"2014-11-01 19:35:41.158"}
{"zipcode":98029,"ProductName":"product4","price":51,"times
tamp":"2014-11-01 19:35:41.323"}
{"zipcode":96194,"ProductName":"product40","price":11,"time
stamp":"2014-11-01 19:35:41.438"}
```

```
.....
```

Real-Time Processing Using Spark Streaming

To connect to the master node, open a terminal window and run this command:

```
/home/hadoop/spark/bin/spark-submit --master local[3] --
class "RealTime" /home/hadoop/workspace/target/scala-
2.10/simple-project_2.10-1.0.jar myStream
https://kinesis.us-east-1.amazonaws.com
```

The **RealTime** streaming object listens to the Amazon Kinesis stream for batch intervals of one second and, using SQL syntax, queries the Resilient Distributed Dataset (RDD). It then writes the data to a file (in HDFS or Amazon S3). Data is moved to your historical location (shown in the following example as `myS3Bucket`) by using the Hadoop API.

Batch Processing Using Spark SQL

Open a terminal window and run the following command:

```
/home/hadoop/spark/bin/spark-submit --master yarn-client --
class "Historical" /home/hadoop/workspace/target/scala-
2.10/simple-project_2.10-1.0.jar myS3Bucket
```

You have now configured and run all of the Lambda Architecture components. This provides you with starting point to develop a unified application that can integrate batch processing with real-time processing under the single code base.

Cleaning up the Software Stack

After you have run queries using Spark SQL, you should shut down your cluster to avoid incurring further charges.

- Terminate your SSH session to disconnect from the master node.
- On your local machine, run the following command to terminate your Amazon EMR cluster. Replace `j-xxxxxx` with the identifier of your cluster.

```
elastic-mapreduce --terminate -j j-xxxxxx
```

- Delete any log files stored in your Amazon S3 bucket, s3://yours3bucket, where *yours3bucket* is the name of the bucket you specified when you launched the job flow. For more information, see Deleting an Object.

Conclusion

The Lambda Architecture described in this paper is a unified architectural pattern that unifies stream (real-time) and batch processing within a single code base. Through the use of Spark Streaming and Spark SQL APIs, you implement your business logic function once, and then reuse the code in a batch ETL process as well as for real-time streaming processes. In this way, you can quickly implement a real-time layer to complement the batch-processing one. In the long term, this architecture will reduce your maintenance overhead. It will also reduce the risk for errors resulting from duplicate code bases.

Contributors

The following individuals and organizations contributed to this document:

- Vadim Astakhov, Amazon Web Services (AWS)
- Manjeet Chayel, Amazon Web Services (AWS)