**How To Deploy Spark Applications In AWS With EMR and Data Pipeline**

Once you create an awesome data science application, it is time for you to deploy it. There are many ways to productionise them. The focus here is deploying Spark applications by using the AWS big data infrastructure. From my experience with the AWS stack and Spark development, I will discuss some high level architectural view and use cases as well as development process flow.

AWS offers a solid ecosystem to support Big Data processing and analytics, including [EMR](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html), [S3](https://docs.aws.amazon.com/AmazonS3/latest/dev/Introduction.html#overview), [Redshift](https://docs.aws.amazon.com/redshift/latest/dg/c_redshift_system_overview.html), [DynamoDB](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html) and [Data Pipeline](https://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/what-is-datapipeline.html). If you have a Spark application that runs on EMR daily, Data Pipleline enables you to execute it in the serverless manner.

The [serverless architecture](https://en.wikipedia.org/wiki/Serverless_computing) doesn’t strictly mean there is no server. When the code is running, you of course need a server to run it. The main difference from the traditional way is that you store your codes and models in a repository, launch the server only during execution and close it as soon as it finishes. In this architecture, you only pay for the cost for the length of code execution. The architecture is often used for real-time data streaming or integration. [AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html) and [Kinesis](https://docs.aws.amazon.com/streams/latest/dev/introduction.html) are good examples.

**What is good about Data Pipeline?**

Data Pipleline is a great tool to use the serverless architecture for batch jobs that run on schedule. You can design the pipeline job to control resources, workflow, execution dependency, scheduling and error handling without the hustle of provisioning and managing servers and the cost of keeping them running all the time.

Another advantage is that you can [create a job with parameters](https://docs.aws.amazon.com/datapipeline/latest/DeveloperGuide/dp-custom-templates.html) (e.g. DB connection URLs, credentials, target schema/table). Data Pipeline jobs are basically JSON files. You can easily export and edit it. With parameters, you can easily promote jobs from the development environment to the production as it is a matter of importing the JSON file from dev to prod (which of course can be coded up for automation).

I also found that debugging and updating Spark codes or models became simpler. We can simply update the repo without touching the pipeline.

[The cost of running Data Pipeline jobs](https://aws.amazon.com/datapipeline/pricing/) is affordable. For a low frequency job (once a day), it costs 60 cent per month as of today. See the link for the pricing information. Note that [the cost of running EMR](https://aws.amazon.com/emr/pricing/) will be charged at hourly rate on top of the running cost of pipelines.

Let’s have a look at the use cases.

**Use Case 1**

* Sourcing the data from different databases (application database, data lake and data warehouse) and joining them prior to running the algorithm.
* The output needs to be presented in a BI tool.

**Prerequisite**

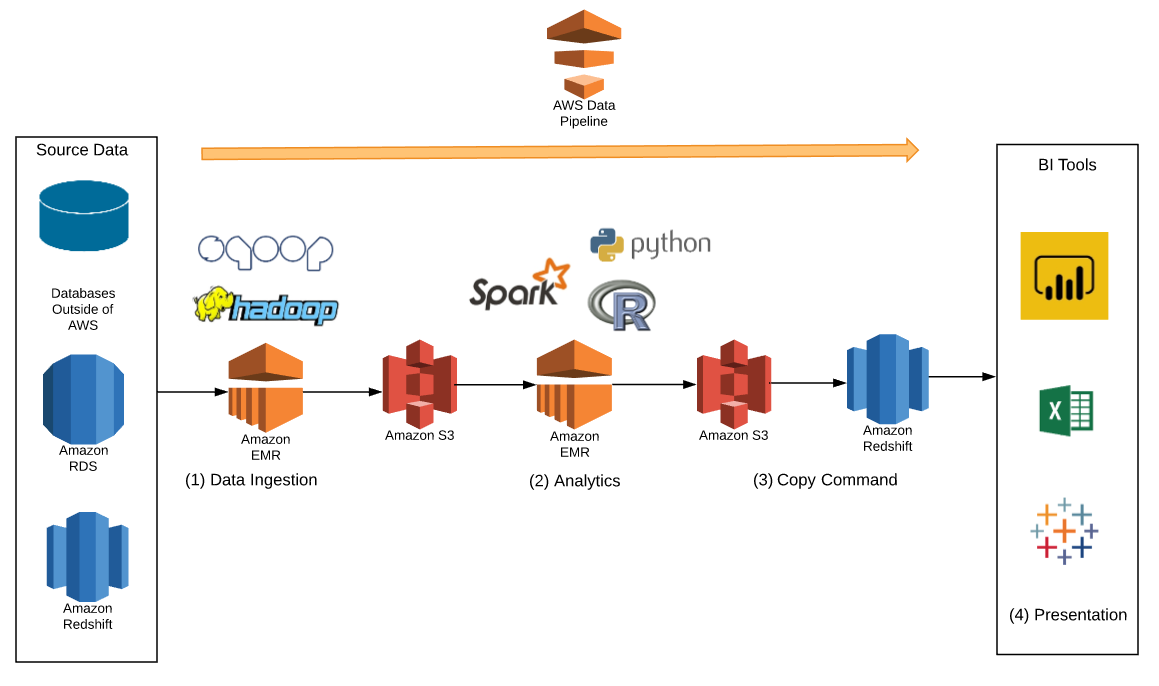
* Save your Sqoop code (as .sh), Spark code and model to Bitbucet or GitHub (or S3, which is less preferable option).

**Solution**

Within the Data Pipeline, you can create a job to do below:

1. Launch a ERM cluster with Sqoop and Spark. Source the Sqoop code to EMR and execute it to move the data to S3.
2. Source the Spark code and model into EMR from a repo (e.g. Bitbucket, GitHub, S3). Execute the code, which transform the data and create output according to the pre-developed model.
3. Move the output of the Spark application to S3 and execute copy command to Redshift.
4. BI tools to fetch the output from Redshift for presentation.

[Sqoop](https://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html#_introduction) is a command line tool to transfer data between Hadoop and relational databases. EMR uses Hadoop for file management. So, it is the best tool to move the data from relational databases through Hadoop in EMR to S3. It is fast and easy to learn. I learned everything about Sqoop from a cookbook which you can download for free [here](https://pdfsecret.com/download/learn-how-to-turn_5a1319cdd64ab24772a3daf3_pdf).



**Use Case 2**

* Ingesting data into Data Lake with an ETL tool.
* Transforming data with ETL or ELT within the Redshift.
* All the data required for the Spark application is in the data warehousing layer.

In the enterprise environment, it is common to have an ETL tool that manage the data ingestion and transformation. Accessing the application databases directly for analytics is not the best architectural practice, either. The first use case is suitable when you need to do data ingestion in an ad-hoc manner or cannot wait for ETL development for the sake of speedy delivery.

**Prerequisite**

The same as Use Case 1.

**Solution**

The figure shows Informatica as an ETL tool. There are heaps of options out there and any tool that suits your use case is fine. I compared DataStage, Informatica and Talend in the past and found Informatica best suited for the particular situation I was in. I especially liked the Redshift connector and I wrote a [small review](https://www.mydatahack.com/informatica-cloud-compatibility-with-aws-redshift/).

The workflow has two parts, managed by an ETL tool and Data Pipeline.

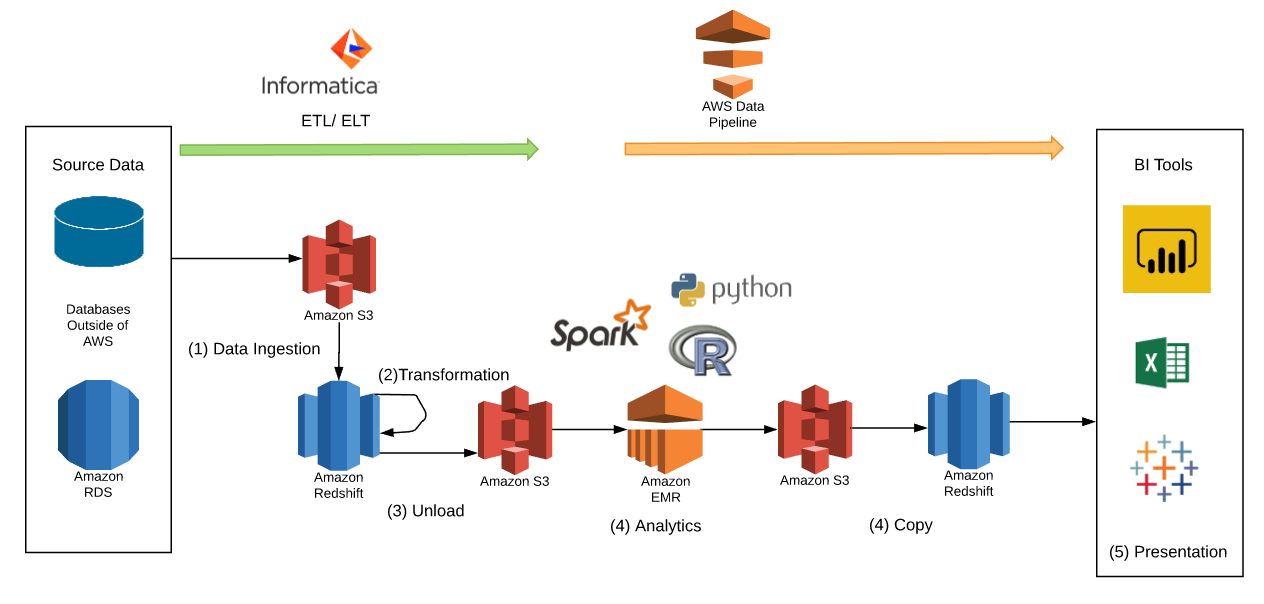
ETL Tool manages below:

1. ETL tool does data ingestion from source systems.
2. Do ETL or ELT within Redshift for transformation.
3. Unload any transformed data into S3.

Data Pipeline manages below:

1. Launch a cluster with Spark, source codes & models from a repo and execute them. The output is moved to S3.
2. Copy data from S3 to Redshift (you can execute copy commands in the Spark code or Data Pipeline).

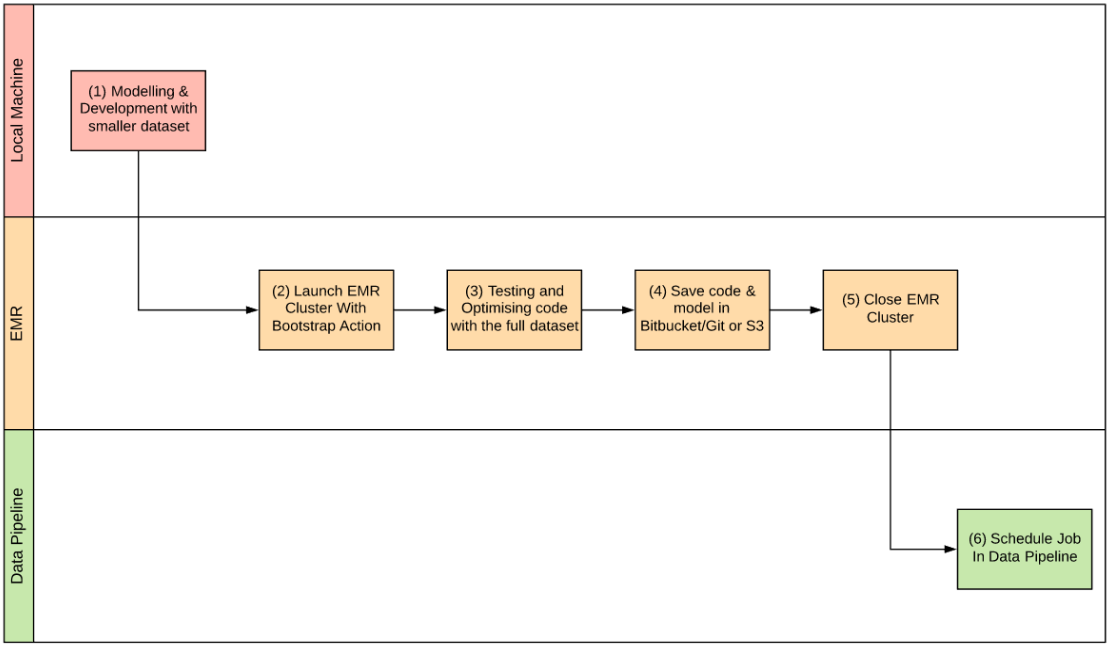
Then, you can source the output into a BI tool for presentation.



**Development Process Workflow**

Finally, let’s have a look at development process workflow. Prior to Spark application deployment, we still need to develop and test the application in an EMR cluster. In this workflow, we only launch the cluster after prototyping on the local machine with a smaller dataset. This will save money as running an EMR cluster is expensive.

Once the code and models are developed, we can close the EMR cluster and move onto the serverless execution in batch. Codes and models can be source from S3 in the Data Pipeline. It is a standard practice to version control them in a git type repository. Sourcing them from a repo in Data Pipeline makes more sense.

****

Let us know your experience with data science application deployment!