Data Platform
Architecture Recommendations

Action	Author	Date	Purpose
Document created	Vasista Polali	10/12/2018	To recommend architecture designs for an enterprise level data platform

Contents

Introduction:

The data engineering landscape primarily revolves around data ingestion, data transformation and data consumption with emphasis on data integrity, security, and quality.

The data engineering landscape has almost entirely migrated towards cloud architecture and managed services for infrastructure operations and management. AWS, in particular, is the market leader which provides a mix of “Infrastructure as a Service” (IaaS), “Platform as a Service” (PaaS) and packaged “Software as a Service” (SaaS) offerings with Azure and GCP being the other players in the domain.

This document is a compilation of recommendations for the Ayzenberg Services team to build an enterprise level data platform that would fulfill their present and future data management requirements to effectively ingest, process, store, and derive value from data and build reports and present them to the business users.

The approaches outlined within this document focus on software architecture aspects such as scalability, ease of maintenance, flexibility, portability, and integration with different environments.

Purpose:

The purpose of this document is to make recommendations on the data platform architecture [redacted] should be adapting to better manage their data pipelines, automate their reports and build a robust platform that can scale up easily to suit their changing needs.

The information used to draft this document has been collected from interviews with the stakeholders from the various teams across [redacted] and by analyzing the relevant documentation provided by them.

Summary:

The information collected through interviews with the various stakeholders and analyzing the documents point to the current situation as:

- Siloed data management, wherein a data set or a data repository is under the control of one specific team and caters to the needs of only that team and is isolated from the rest of the organization.
- Data is mostly stored and managed in spreadsheets.
- A lot of manual business and software processes are being followed, that could be automated and improved.

While the prevailing system is a matter of necessity rather than choice, it can impact data integrity, data consistency and drain valuable organizational resources unnecessarily for mundane tasks.

A few things to consider before going further:

- There has to be a comprehensive IT and data management strategy to formulate enterprise level IT and data management policies.
- It should be considered that the strategy is an organic process and should be flexible and ever adapting to the changing business needs and trends.
- The final data management solution to be adapted should be decided in conjunction with the overall IT strategy, scope of work and direction that Ayzenberg wants to take.
- This document is a technical recommendation outlining different approaches and is designed to be comprehensive and user-friendly. As all users may not be interested in all parts of the document it has been abstracted as much as possible to the point that it makes sense even if individual blocks are read in isolation. So being repetitive was necessary to maintain the abstraction and is intentional.

This document is split into two parts:

Part A discusses the development and maintenance of an enterprise-level data platform to ingest, transform, retain and consume data in a holistic, integrated manner and the processes to be followed therein.

[redacted]

The most recommended architecture is analyzed first, followed by the second and third best ones.

In Scope:

The scope of this document is to analyze the data requirements, current processes followed, existing infrastructure, reporting requirements and issues faced by various business groups at Ayzenberg and recommend the best approaches to build a fully operational, integrated and a scalable enterprise-level data platform for the various groups to process data in a non-redundant and shared manner while maintaining data quality and integrity and use it to drive their respective business functions in a fully automated way.

Out of Scope:

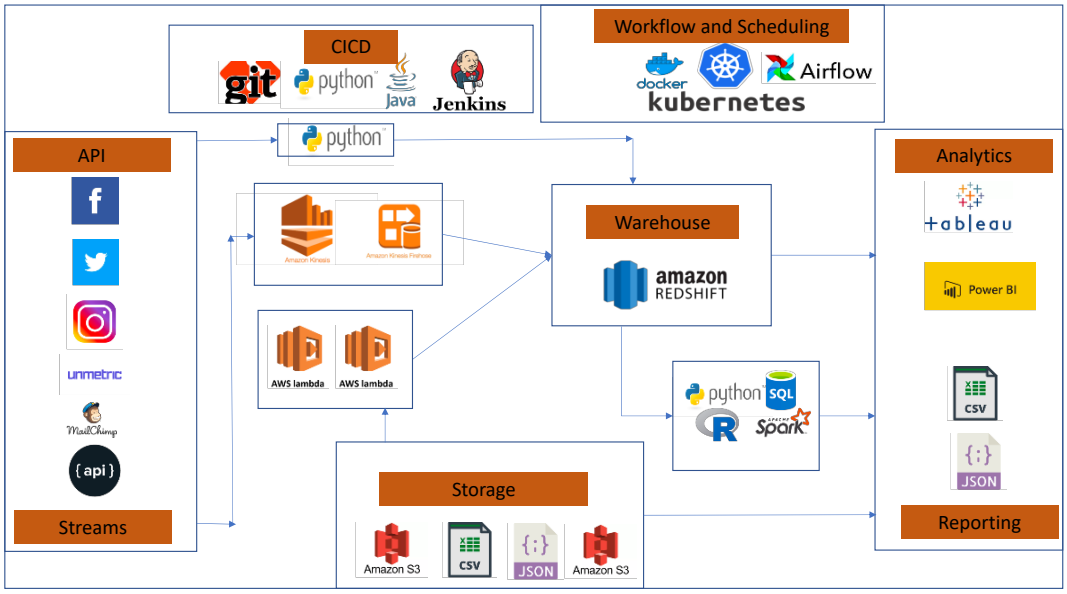
A general outline will be provided on best practices on how to structure organizations and accounts in cloud infrastructure and environments in software development cycle. This document does not deal with enterprise-level security, user administration and the nitty-gritty of IAM policies and roles in particular.

Assumptions and Dependencies:

The document assumes that all the relevant information has been provided in the way of stakeholder interviews, requirement documents, answers to queries raised by [redacted] resources and any other relevant information that might have been sought by [redacted]

PART A

Hybrid Model with AWS Managed Services and Custom-built Applications



The architecture diagram above describes a data processing platform which is a hybrid model between AWS SaaS and PaaS offerings such as AWS Lambda Redshift and S3 and custom-built applications and tools that suit Ayzenberg's precise business needs. This architecture is best suited and the number one recommendation as it inculcates best of both worlds and is very flexible for future enhancements and integration with other tools and environments.

Key Points:

- A hybrid model with a combination of custom developed applications and AWS managed services.
- High performance and reliability because of granular control over the processes.
- A modular approach, where the platform components are logically separated by functionality and can be replaced or enhanced easily without affecting the overall performance of the platform.
- Very scalable and flexible as it can be scaled out to handle bigger workloads or additional functions easily without significant changes to the design.
- Portability across environments is easy as most of the components can be migrated on to other cloud platforms with minimal effort.

- The development and maintenance effort are more compared to running everything with AWS managed services.

The platform consists of a data ingestion component, a storage component, a processing component, and a reporting component.

Data Ingestion:

3rd Party APIs and Other Databases:

APIs are a quintessential part of data delivery in all systems these days and all the platforms like Facebook, Twitter, Instagram, Google Analytics and 3rd party vendors that Ayzenberg uses like Unmetric, Crimson Hexagon, and MailChimp have APIs that can be used to extract the attributes that the business users are interested in.

The following would be the key steps followed with ingestion:

- Applications would be developed to call the APIs for each unique platform and scheduled to run in intermittent time periods to collect data for predetermined attributes.
- Different instances of the application would be run for different combinations of dimensions and metrics that would be passed as arguments to the application along with the connection parameters.
- The applications would be developed in Python primarily and other programming languages like Java or GoLang may be used as required.
- The data in the API response would be parsed and transformed into a relational schema based on the business logic developed in conjunction with the business users.
- The transformed data set would be ingested into a Redshift data warehouse table in the appropriate schema.
- Data will be ingested from other databases using JDBC based applications or through “Unload to S3 and COPY” approach in case of another Redshift cluster.
- A copy of the raw dataset will be saved in an S3 location for each run for backup and historical retention.

Files:

While the goal is to automate the entire data extraction component, there may be instances where the raw data has to be collected and compiled manually.

- Data generated manually will be uploaded into a specific S3 bucket by the user in form of CSV or JSON files.
- S3 events will be configured on a Lambda function. Whenever a file is uploaded into the S3 bucket, the Lambda function would be triggered parsing the data and ingesting it into a Redshift table after appropriate transformation based on the business logic.

Data Warehouse and Data Lake:

Amazon Redshift is a fast, scalable data warehouse that makes it simple and cost-effective to store and analyze all your data. Redshift delivers ten times faster performance than other data warehouses by using machine learning, massively parallel query execution, and columnar storage on high-performance disks.

- A data warehouse would be created in AWS Redshift with database objects grouped as different schemas.
- Redshift specifications are outlined in Part B of the document, team wise.
- The raw data saved in S3 can be used as a data lake by processing and exposing it by using AWS tools like Glue, Athena, Redshift Spectrum etc.

Processing and Scheduling:

Airflow is a platform to programmatically author, schedule and monitor data pipelines. Workflows in Airflow are run as directed acyclic graphs (DAGs) of tasks.

- An Airflow cluster of 4-5 worker nodes will initially be provisioned to schedule and run the data ingestion and data processing applications.
- More workers will be added as the computing requirements increase. Auto-scaling can also be enabled to increase and decrease the cluster size based on the workload at that point in time.
- The Airflow server and the worker nodes would be run as containerized applications with Docker and orchestrated on a Kubernetes cluster like AWS EKS (Elastic Kubernetes Service).
- Running services as Docker containers will help to avoid provisioning of dedicated servers in form of EC2 instances to run the applications. It avoids setting up of separate monitoring infrastructure and avoids the pitfalls of clashes between the versions of the same dependencies used by different applications running on the same server. Docker images are built by pre-packaging all the dependencies for a particular application along with the code and can be run on any server once built. It avoids the ubiquitous “It runs on my machine problem”.
- Running the containers on Kubernetes will help with auto-scaling the application based on the workload and fault tolerance, as manual intervention is not needed to bring up another instance of the server to replace a dead one and also avoids resource starvation as these aspects would be taken care by the cluster. It’s cheaper as resources are shared among numerous micro-services avoiding the need for paying for dedicated servers.
- Any number of applications can be deployed as containerized apps and easily integrated into the data platform as service discovery is an essential part of Kubernetes.

CICD and Version Control:

CICD (Continuous Integration and Continuous Delivery) and version control are an essential part of software engineering. While it not only helps to build and deliver code faster by automating processes but also makes it safer by avoiding the delivery of broken code.

Version control is important as it records any changes to the code and maintains versions of the code for reference and recovery.

- Git or GitHub will be used as the version control system.
- Jenkins pipeline projects will be developed and integrated with Git for CICD.
- Terraform will be used to provision and maintain AWS tools and services.

Analysis and Reporting:

The core reason for the data processing effort at Ayzenberg is to perform descriptive, prescriptive and predictive analysis of the data and convey the results to appropriate stakeholders as reports.

- A Redshift data warehouse will be the main source of data for the analytical functions.
- Other raw files or any other sources may be used in combination along with the data in the Redshift database to generate appropriate reports.
- The analysis will be done using SQL, Python, R programming or for Big Data needs Spark applications will be developed and run on clusters provisioned using EMR, QUBOLE or Databricks.
- The results of the analysis can be written back into the data warehouse in a separate schema or saved as files in S3 locations based on the user's preference.
    - Best Practice: Save it back to the warehouse. It's convenient and fast for downstream applications to use the data and files may get corrupted depending on the storage format and file system.
- Business reports will be automated by the different groups using data visualization applications like Tableau or Power BI. Automation requirements and techniques are outlined for each business team in Part B separately.
- Redshift data warehouse will be the main source of data for business reporting.
- Where generating reports in Excel or CSV files is mandatory, command line tools or Python programs will be developed to generate automated reports in the CSV file formats and upload them to an S3 location. All the reporting logic will be handled by the programs and scheduled via Airflow. Slack notifications will be sent as part of the workflow as soon as the report has been generated or in case of any exceptions.
- Connecting to Redshift from within Excel file and populating data with SQL queries directly is another option subject to compute resources limitations and user convenience.

Scalability:

Scalability of a system is the most important aspect of a software system design that has to be thought through thoroughly to make sure the platform meets the growing demand with

increased future workloads in. With the advent of distributed architectures horizontal scalability has both become limitless and relatively easy to achieve but time and again this has been proven to be elusive.

- This design makes it easy to achieve optimal scalability with very little resource overhead.
- This design is a collection of functionalities developed by selecting the software technologies and tools best suited for the job and integrating them into one whole system that complements each other. The modular approach of separating the functionality into multiple parts makes the architecture very flexible and scalable as scaling one component is not dependent on the other modules and also won't prove detrimental to each other.
- Scalability is also inherent to the AWS services used in the architecture and can be achieved with minimum overhead.
- For example, if there is an exponential increase in workloads in the form of new clients, new APIs to ingest data from etc. all that has to be done is spin new APIs to ingest the raw data, increase the Airflow cluster size for enhanced computing needs with Kubernetes auto-scaling and the Redshift cluster can be resized based on the growing storage needs. These operations are all supported out of the box by the respective software and do not need a lot of manual labor to achieve.
- Apache Spark is a distributed processing engine with APIs for Scala, Python, and R which can be leveraged to perform data analysis on the now grown data sets.

Flexibility:

In the ever-changing landscape of software tools and technologies, as new ones are being invented and introduced into the market incessantly, it is essential for a software system to be flexible enough to adapt to the changing trends without much effort.

- This design achieves flexibility because of its modular approach where one component can be completely replaced in the future with minimum overhead.
- For example, if in the future it were decided to replace the warehouse with a then top performing alternative, all that has to be done would be to change the driver details and connection parameters in the applications.
- For example, if Airflow is being replaced with another workflow engine, the new platform has to be spun up as containers on Kubernetes and the existing application code simply migrated to the new workflow.

Portability and Vendor Lock:

Since the majority of the components are built with open source technologies and the cloud vendor is used only for infrastructure to run them, these can be easily moved onto other cloud providers based on cost-benefit ratio considerations. Also, different parts of the system can be implemented on different cloud solutions based on cost and efficiency needs.

Things to Consider:

While this design is robust, scalable and flexible, there are a couple of things to consider.

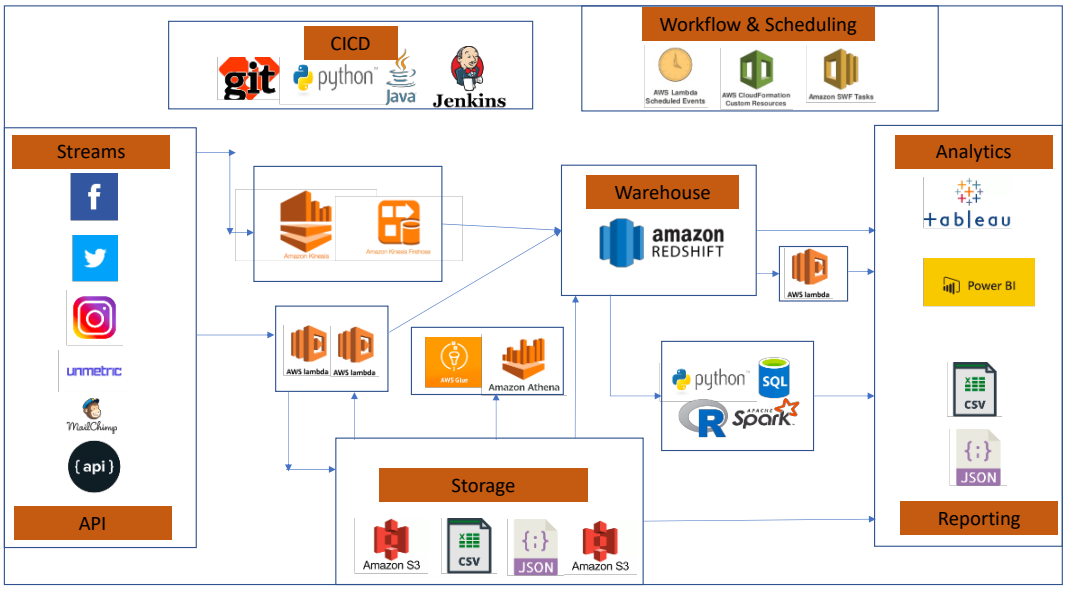
- This design gives granular control over the functionality and implementation due to the high level of customization. The same application can be iteratively enhanced to accommodate changing business requirements.
- At the same time, it also calls for a certain degree of skills to be present in the developers and operational people.
- The required initial development effort is higher compared to building the solution using managed services and tools offered by the cloud providers like AWS.
- More maintenance and monitoring effort is needed because of the custom applications developed, compared to out of box solutions like using managed services and tools offered by cloud providers like AWS, definitely not more than any midsized platform that sees decent traffic.

Resources Needed for Operations:

A minimum set of resources is needed to maintain this architecture, not counting the report developers and business users:

- Software engineer(s) focused on data engineering with broad-based knowledge of software systems and their integration to maintain and enhance the data pipelines.
- DevOps engineer(s) familiar with AWS services, CICD pipelines, and general system administration tasks.

Data Platform with AWS Managed Services



The unique selling proposition of cloud providers is to provide infrastructure as a service so that the user can focus more on developing the application rather than setting up the infrastructure, run and maintain it. In a similar way, PaaS and SaaS provide the users with software components that can be used out of the box with little or no customization to quickly bootstrap an application and get it running. A combination of these tools and services can be integrated to build this version of a data platform. This architecture focusses on using AWS services to build the Ayzenberg data platform.

Key Points:

- Entirely built with AWS managed services.
- Performance and reliability are as advertised by AWS and subject to resource limitations applied by AWS. There is little control over the resource allocation or underlying runtime. For example, any logic in a Lambda function has to be limited to be able to process data that is not more than 3GB in size and can be processed within 5 minutes. Beyond that multiple Lambda functions have to be invoked concurrently or consecutively skewing the cost-benefit ratio and impose restrictions on file sizes and types in the S3 storage layer.
- A modular approach, where the platform components are logically separated by functionality and can be replaced only by other AWS tools and services or enhanced easily without affecting the overall performance of the platform.
- Scalability and flexibility are depending on the limitations placed by AWS on individual services and adding additional functionality may require design changes based on the functionality of the AWS services chosen or that are available.
- Portability across environments is not possible without significant design changes and development effort as the components are entirely developed in AWS and cannot be migrated on to other cloud platforms.
- The development and maintenance effort are lower compared to the hybrid model.
- The development resources need AWS specific skills to operate and maintain the platform.

The platform consists of a data ingestion components, a storage component, a processing component, and reporting components.

Data Ingestion:

3rd Party APIs and Other Databases:

The following would be the key steps followed with ingestion:

- AWS Lambda, a serverless event-based computing platform, would be used to call the APIs for each unique platform and scheduled to run in intermittent time periods using “scheduled events” to collect data for predetermined attributes and dump the raw data into an S3 bucket, similar to what is implemented today.
- Different instances of Lambda functions would be run for different combinations of dimensions and metrics that would be passed as arguments along with the connection parameters.

- The functions will be primarily developed in Python and other programming languages like Java or GoLang may be used as needed.
- The raw data stored in the S3 bucket will be parsed, transformed into a relational schema based on the business logic developed in conjunction with the business users, using another set of a Lambda function. S3 events will be configured to trigger the functions.
- The transformed data set will be ingested into a Redshift data warehouse table with the appropriate schema.
- Data will be ingested from other databases using Lambda functions, implementing JDBC, or through “Unload to S3 and COPY” approach, in case of a Redshift cluster.
- Redshift Spectrum can be used to create external tables on the raw S3 data. This is a simple abstraction providing schema on reading capabilities allowing the same data to be parsed with different schemas as the storage and metadata are separated.
    - **Pain Factor:** While this is an effective method to process data, it is not flexible enough to handle dynamic schema in data sets from social media and does not scale up enough to support nested data structures like JSON. Customization of the parsing logic is not possible.
- AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it easy to prepare and load data for analytics. AWS Glue uses a combination of crawlers and a catalog to automatically infer schema, generate ETL code and catalog the metadata of the data set. Typically, AWS Athena is used to running SQL queries and analytics on the transformed data.
    - **Pain Factor:** Glue works on Spark, generating compatible PySpark code. While the user has access to the generated code and can modify it to a certain extent, as long as it is using the Glue DSL, complex transformations to wrangle data to support customization is not possible. Importing external Spark packages, other than those provided by Glue, is also not supported, limiting the customization further.
- Data streams will be ingested through Kinesis and consumed using Kinesis Firehose to land data in Redshift.
    - **Pain Factor:** Firehose uses an S3 and COPY command to consume data from Kinesis streams and load it into Redshift. This is a fully managed service and any custom transformation is not possible. Doesn’t scale up to work with nested data. A custom application has to be developed to handle dynamic schema and complex data structures on top of the ingested raw data separately, causing redundancy.
- The raw dataset will be retained in the S3 location for backup and historical retention.

Files:

While the goal is to automate the entire data extraction component, there may be instances where the raw data has to be collected and compiled manually.

- Data generated manually will be uploaded into a specific S3 bucket by the user in form of CSV or JSON files.

- S3 events will be configured using Lambda functions. Whenever a file is uploaded into the bucket, the Lambda function would be triggered parsing the data and ingesting it into a Redshift table, after appropriate transformation based on the business logic.

Data Warehouse and Data Lake:

Amazon Redshift is a fast, scalable data warehouse that makes it simple and cost-effective to analyze data. Redshift delivers ten times faster performance than other data warehouses by using machine learning, massively parallel query execution, and columnar storage on high-performance disks.

- A data warehouse will be created in AWS Redshift with database objects grouped as different schemas.
- Redshift specifications are outlined in Part B of the document, team wise.
- The raw data saved in S3 can be used as a data lake by processing and exposing it by using AWS tools like Glue, Athena, Redshift Spectrum etc.

Processing and Scheduling:

- All the applications are run using AWS services. No separate runtime environment is needed.
- Scheduling is done using custom AWS tools like scheduled events, CloudWatch events, and Simple Workflow tasks.

CICD and Version Control:

CICD (Continuous Integration and Continuous Delivery) and version control are essential aspects of software engineering. While it not only helps to build and deliver code faster by automating processes but also makes it safer by avoiding the delivery of broken code.

Version control is important as it records any changes to the code and maintains versions of it for reference and recovery.

- Git or GitHub will be used as the version control system.
- Jenkins pipeline projects will be developed and integrated with Git for CICD.
- Terraform will be used to provision and maintain AWS tools and services.

Analysis and Reporting:

The core reason for this effort is to perform descriptive, prescriptive and predictive analysis on the data and convey the results to appropriate stakeholders as reports.

- The Redshift data warehouse will be the main source of data for the analytical functions.
- Other raw files or any other sources may be used in combination of the data in the Redshift database to generate appropriate reports.

- The analysis will be done using SQL, Python, R programming and for Big Data needs Spark applications will be developed and run on clusters provisioned using EMR, QUBOLE or Databricks.
- The results of the analysis can be written back into the data warehouse into a separate schema or saved as files in S3, based on the user's preference.
- Business reports will be automated by the various teams using data visualization applications like Tableau or Power BI.
- The Redshift data warehouse will be the main source of metrics for business reporting.
- For reports where generating them in Excel or CSV files is mandatory, Lambda functions will be developed to generate automated reports in the CSV file formats and uploaded to an S3 location. All the reporting logic will be handled by the Lambda functions and scheduled via "Cron" expressions in the Lambda configuration. Slack notifications will be sent as soon as the report has been generated, or in case of any exceptions.
- Connecting to Redshift from within an Excel file and populating data with SQL queries directly is another option, subject to compute resources limitations and user convenience.

Scalability:

Scalability of a system is the most important aspect of a software system design that has to be thought through thoroughly to make sure the platform meets the growing demands with increased future workloads in the future. With the advent of distributed architectures horizontal scalability has both become limitless and relatively easy to achieve but time and again has proven to be elusive.

- While scalability is inherent to individual AWS services and can be achieved with minimum overhead, it may not be the case when trying to achieve the same with the entire data platform. The platform is developed using out of the box managed services that are designed to deliver a set functionality with resource limits placed on them. For example, if Redshift Spectrum cannot handle nested data properly, custom solutions should be developed to process and analyze the data structure instead. While this can be done using AWS Lambda, the resource limits placed on Lambda functions may restrict it from being used for complex and resource intensive computations.
- Scaling up AWS tools and services comes with a cost attached to them and should be looked at in the perspective of cost-benefit ratio.

Flexibility:

In the ever-changing landscape of software tools and technologies, as new ones are being invented and introduced into the market incessantly, it is essential for a software system to be flexible enough to adapt to the changing trends without much effort.

- This managed services design is flexible to the extent that it can be integrated with any services that are part of the AWS catalog and can be safely assumed to be integrated or replaced with any other solutions AWS may introduce in the future. Of course, subject to the functional and resource limitations placed on them.

- Integrating it into other custom developed applications or services provided by other cloud vendors may need design changes to be accomplished introducing the overhead of increased development effort every time something has to be done. If not handled carefully or if done in an ad-hoc manner, such efforts, in the long run, may introduce a lot of redundancy, performance deterioration and spaghetti code.

Portability and Vendor Lock:

Since all the components are developed with AWS services, it introduces into the architecture a characteristic of getting “vendor locked”. The applications and the data cannot be easily moved to other environments or onto other cloud platforms. Such a move entails major development effort in terms of resources and time to achieve it.

Things to Consider:

While this design is robust, scalable and flexible, there are a couple of things to consider.

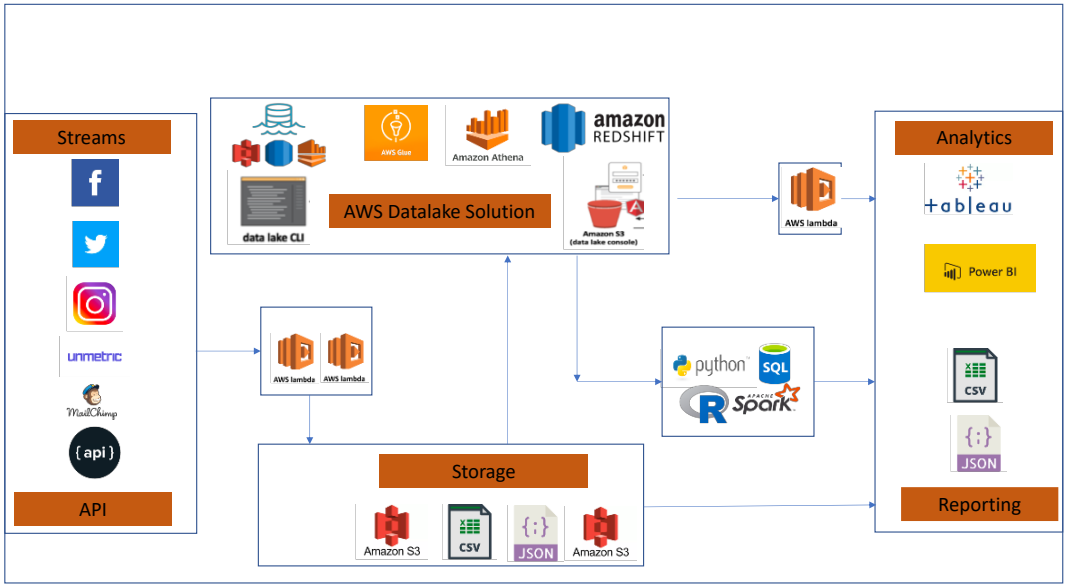
- The applications cannot be iteratively enhanced to accommodate changing business requirements and new ones have to be developed due to external restrictions that may have been introduced in the current ones like compute resource limits or dependency injection-related, for example in AWS Lambda.
- The resource developing the application can focus entirely on the application logic rather than worry about performance optimization based on the runtime environment as performance is guaranteed by AWS. The resources need not have a broad-based knowledge in software engineering but can get started with mid-level application development experience.
- Bootstrapping applications is quick and overall development effort is lower compared to building customized solutions.
- Maintenance and monitoring effort is lower compared to customized solutions as applications and pipelines are developed on AWS provided platforms, which have maintenance and monitoring tools well integrated into them.

Resources Needed for Operations:

Resources needed to maintain this architecture, not counting the report developers and business users.

- Data engineer(s) to maintain and enhance the data pipelines.
- DevOps engineer(s) familiar with AWS services, CICD pipelines, and general system administration tasks.

Data Platform with AWS Data Lake Solution



AWS offers a data lake solution that automatically configures the core AWS services necessary to easily tag, search, share, transform, analyze, and govern specific subsets of data across a company or with other external users. The solution deploys a console that users can access to search and browse available datasets for their business needs. This architecture focusses on using AWS data lake solution to build Ayzenberg's data platform.

Key Points:

- Entirely built with AWS data lake solution and managed services.
- Performance and reliability are as advertised by AWS and subject to resource limitations applied by AWS. There is no control over the resource allocation or underlying runtime.
- Some services are available only in certain AWS regions.
- Not a modular approach as the data lake solution has a set of pre-determined AWS services and processes it has to work with and uses AWS Cloud Formation to spin up services and cannot be separated into logical functional units.
- Scalability and flexibility are limited as the data lake solution is very rigid and has a set of AWS services and processes it has to work with.
- Portability across environments is not possible as it is an AWS custom solution.
- Use cases are based around using S3 and Glue for ETL and Athena and Redshift for analytical functions.
- This is the least flexible of all three solutions. There is no customization possible on the configuration front and minimal flexibility on the data wrangling side. External applications have to use the data lake CLI or console interfaces to interact with the data packages or use the SQL interface.