First exercise – Architecture

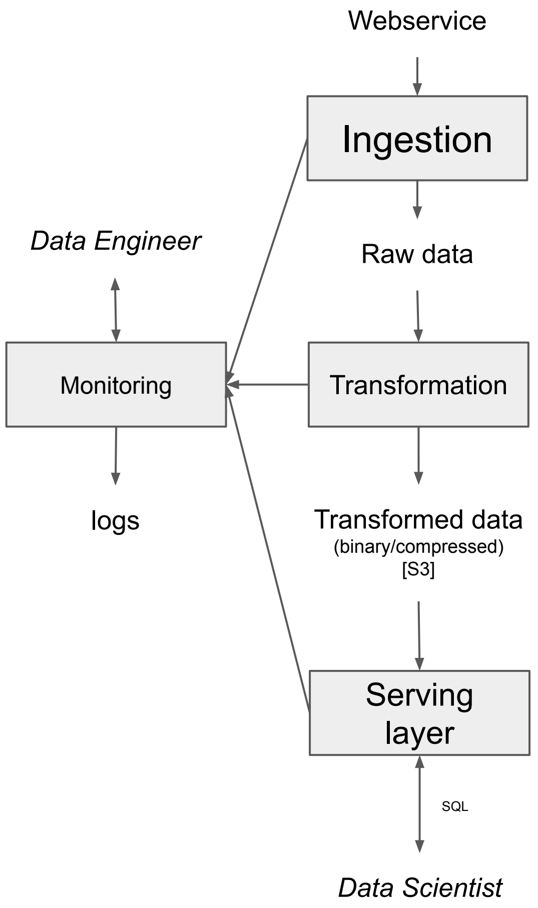
Definition

The requirement is to design a **data pipeline architecture** which will enable the **ingestion** of data from a third party tool and **dumping** it on a data lake (S3 Bucket):

- The third party tool exposes an HTTP endpoint, reachable only with a GET method.
- The data must be stored in raw format.
- A transforming layer must convert the data to a binary format (compressed).
- A serving layer through a SQL interface must be offered to the data scientists, so they can interact with the transformed data in the bucket.

Expected deliverable

The main output of this stage should be one diagram with the proposed solution and an explanation of the technological and architectural choices which were made. All the solution must be based on AWS components.

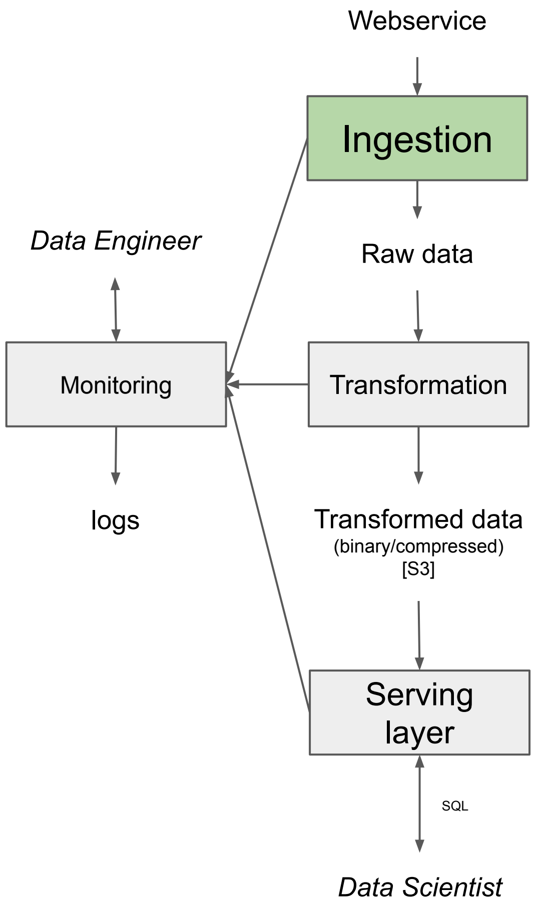


First exercise – Architecture

- Hypothesis and considerations
  - Considering a simplified data maturity model, what is our level: starting, scaling or leading (Reis & Housley, 2022). Jumping to exercise 2, I guess that we are at scaling level (establishing formal practices, creating robust architectures, adopting DevOps and DataOps etc)
  - We are part of a organization with experience with Cloud Native architectures, so the data engineer is supported by a team of Cloud Architects and Solution Architects that can share experiences and knowledge in terms of IaC for AWS.
  - “The flexibility of contemporary data pipelines allows them to be loosely coupled together to make more complex data pipelines. Loosely coupled pipelines are easier to develop, test, deploy, and reuse.” (Harvinder Atwal, 2019)

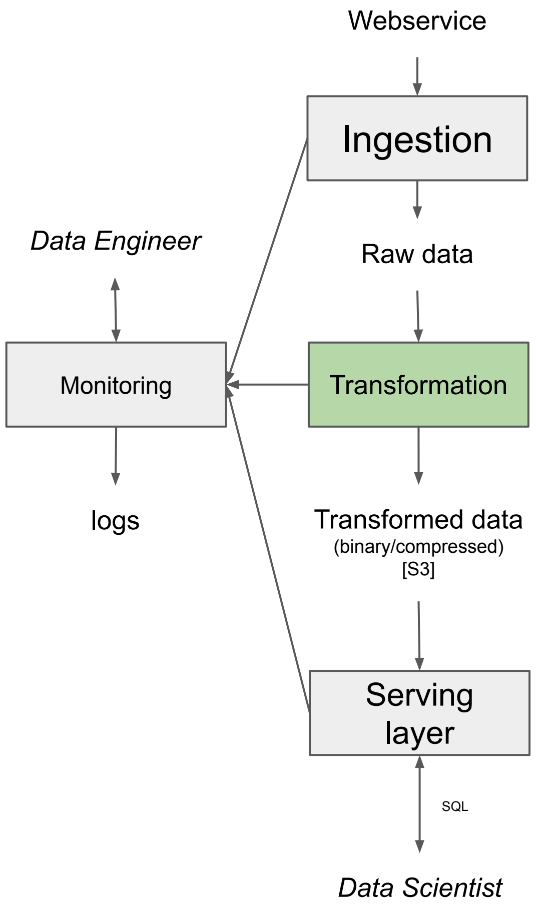
First exercise – Architecture

- Data Ingestion
  - Batch: in the proposed use case we have a batch ingestion. We can possibly run it Day-1 or Hour-1 in push mode.
  - Streaming: not considered in the use case, but it can be adopted in the future. So I recommend to decide from an architecture that can work with streaming.



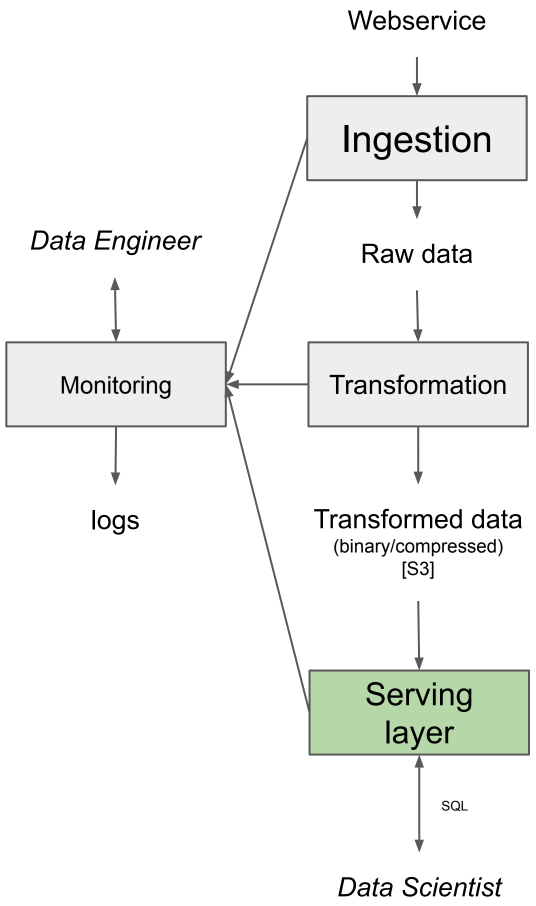
First exercise – Architecture

- Transformation
  - The error handling and retry mechanisms must be coded in a workflow. Tools that could graphically support the workflow design are highly recommended.
  - The main events of the transformation jobs must be monitored by an appropriate interface that could export this data to our own data lake. The same for logs. We can use our data pipeline to analyse itself.
  - About the binary format (compressed), we must use a Apache Parquet format (columnar storage) that will perfectly fit to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.I will bring performance, compression and lower costs to the serving layer



First exercise – Architecture

- Serving Layer
  - The usage of SQL is a tendency to the Data Scientists.
  - Nevertheless, we have to consider that during time the scientists starts asking for more complex queries.
  - The Data Scientists with highly coding skills will ask for other possibilities to code and access the data directly in the S3. For that, a table catalog is mandatory.
  - In other words, the SQL serving layer is mandatory, but if this layer also supports other approaches for future pivotations.



First exercise – Architecture

- Decision about frameworks
  - Hadoop
    - Data protection amid a hardware failure
    - Scalability from a single server to thousands of machines +1
    - Real-time analytics
  - Spark
    - SQL queries, streaming data, machine learning (ML) and graph processing +1
    - 100x faster than Hadoop for smaller workloads via in-memory processing +1
    - APIs for ease of use

First exercise – Architecture

- Decision about a non-PaaS or a PaaS data pipelines
  - nonPaaS (Amazon MWAA - Apache Airflow)
    - Job orchestration
    - Great flexibility (with great responsibility)
  - PaaS
    - ETL focused
    - It is recommended to choose a solution that can help the development, simplify the maintenance and with reduced infrastructure set up required
    - PaaS +1 and Serverless +1

First exercise – Architecture

- AWS PaaS Data Pipelines
  - Amazon EMR Serverless: “Easily run and scale Apache Spark, Hive, Presto, and other big data workloads”
    - Former Amazon Elastic MapReduce
    - Flexible, complete, higher dev cost, lower operation cost, indicated for pipelines with frequent execution.
    - Supports: Apache Spark, Apache Hive, Presto
    - Dev using EMR Notebooks and EMR Studio.
  - AWS Glue: “Simple, scalable, and serverless data integration”
    - Worker with max memory of 32 Gb
    - Team can focus in ETL job not in configuration, simple job maintenance, integrated data catalog, logs and metrics to cloudwatch
    - Supports: Apache Spark (pySpark), Python and Scala
    - Dev using AWS Glue Studio or Jupyter (running locally)
    - AWS Glue +1

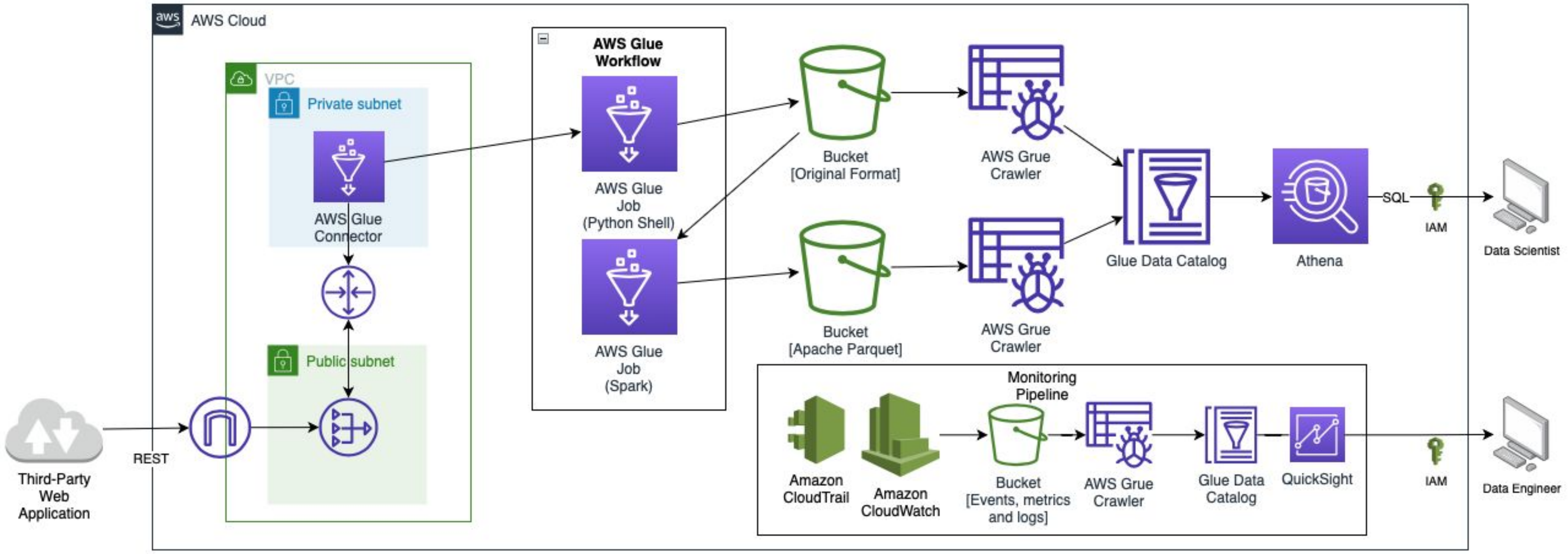
First exercise – Architecture

- Decision about ETL Workflow
  - Amazon Glue Workflow
    - It is not as complete as Amazon Step Function but works fine for this simple case.
    - If in the future the third-party interface evolutes, Glue Workflow can be adapted to use Apache Kinesis.
    - Can be created using AWS Glue blueprint

First exercise – Architecture

- Decision about serving layer
  - Amazon Athena: Start querying data instantly. Get results in seconds. Pay only for the queries you run.
    - Serverless
    - Pay as you go
    - ANSI SQL queries are available in Apache Athena and it can be connected using JDBC. Considering that the data scientists can have role and permissions to Athena, the IAM authenticator must be used to permit the connection.
    - If the data scientists use Tableau, Athena also provides specific connectors.
    - Query S3 Using SQL: We cannot lose the scalability of a noSQL ETL, and we have to continue using “the language of data”: SQL

First exercise – Architecture



Second exercise – Architecture

Definition

Managing the data flows like the ones requested in the goals above requires a strong DevOps component.

Please discuss what strategies, technologies and tools you would use to simplify and automate this day to day management, namely:

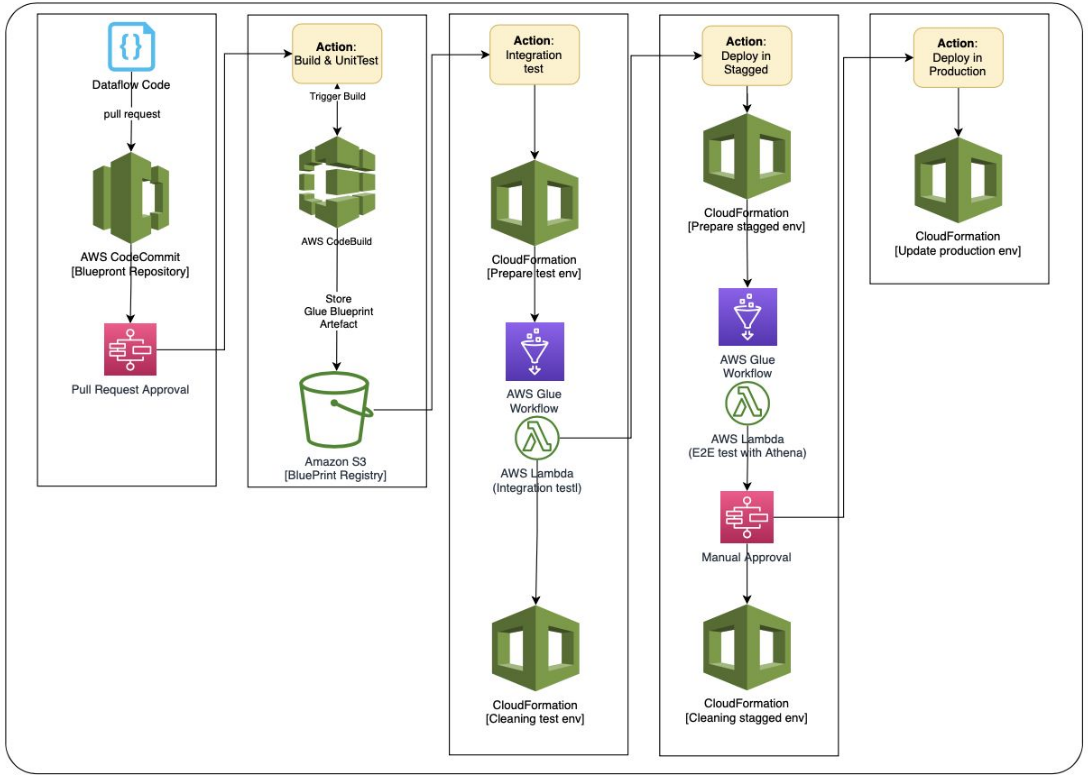
- metrics and monitoring for the running dataflows
- recovering from a failed dataflow
- testing a dataflow in a development environment and then deploying it to a production environment
- modifying code on the dataflow without stopping it

Expected deliverable

The main output of this goal is a discussion of the above topics and proposed approaches on how to tackle them. All the solution must be based on AWS components.

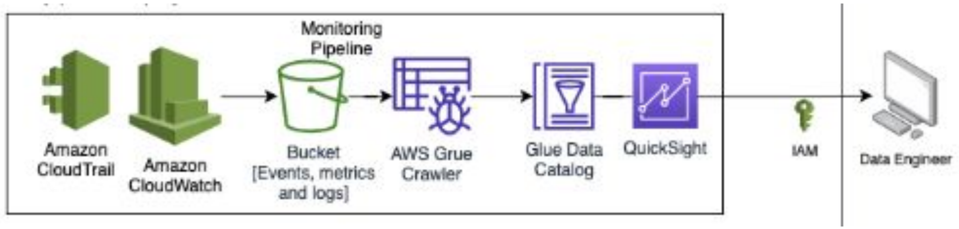
Second exercise – Architecture

Proposed DataOps



Second exercise – Architecture

- Metrics and monitoring for the running dataflows
  - The AWS Glue is integrated to:
    - Amazon CloudWatch for a unified logging base.
    - Amazon CloudWatch (using AWS Glue job profile) to collect jobs metrics
    - Amazon CloudTrail for the workflow events.
  - The access to S3 Bucket must be also monitored using Amazon CloudTrail.
  - For near real-time analysis, CloudWatch Logs Insights can be used.
  - Amazon CloudTrail and CloudWatch can export the log data to S3 with 12 hours delay. This bucket can be catalogued by AWS Glue Crawler and accessed via AWS Athena, where Amazon QuickSight can directly access and deliver BI dashboards.



Second exercise – Architecture

- (1) Recovering from a failed dataflow
  - Interpreting recovering term as a ETL Workflow that do not successfully complete, this means that the workflow only partially ran.
    - "After you find the root causes and make corrections, you can select one or more nodes to resume the workflow run from, and then resume the workflow run. The selected nodes and all nodes that are downstream from those nodes are then run." (link here)
  - Interpreting recovering as a systematic failure of the of the ETL Workflow code,
    - A) Failed before load (transformed data was not affected): it must be solved directly by the CI/CD pipeline; manually trigger the previous release, Deploy to Production Environment and run the healthcheck.
    - B) Failed after load: stop the downstream AWS Glue Crawler; If possible remove all affected data; If not possible, return a backup; if not possible re-execute dataflow with the previous release to reconstruct all the transformed data.

Second exercise – Architecture

- (1) Recovering from a failed dataflow
  - Best Practices:
    - Use events to highlight important actions of the workflow
    - Use S3 bucket with Versioning in order rollback versions.
    - Use modified date to search for files modified in order to delete or rollback to previous version.
    - It can be achieved using appropriate versioning of the build phase in order to permit the redeploy.
    - Decouple ETL process in order to avoid malformed data propagation.
    - Automatic backups to S3 Glacier must be implemented and used to recover the bucket to a previous state.

Second exercise – Architecture

- (2) Testing a dataflow in a development environment and then deploying it to a production environment
    - Shift-Left Testing
        - The ETL job must be tested during the development in local machines.
        - Tests are added to version control and runned as part of an automated CI process.
        - AWS Glue jobs can be tested locally using a Docker container.
    - Test data management (TDM)
        - More critical for data analytics than software development
        - Tests are complex because we have two parts: data and code. Data in production continually changes..
        - There is often more complexity in data than code.
        - Automated testing requires the rapid and secure provision of fit-for-purpose test data.

Second exercise – Architecture

- (2) Testing a dataflow in a development environment and then deploying it to a production environment
  - Choosing test data
    - Ideally, test data should be a full copy of the production dat.
    - If it is not possible to load a large copy into stagged environments, a large sample is the next best option.
    - Fake data is not ideal except for sensitive data.
  - Don’t forget non-functional tests
    - Latency KPIs, security tests for password creation and authentication, access control policies, scans for open ports, exposure of cloud storage or sensitive data, code analysis for vulnerabilities etc.
  - Proposed Approach:
    - Automate unittest in build phase using containers
    - Automate Integration tests with lightweight and fake data
    - Deploy to Pre-Production (stagged) with data similar to production and running E2E tests based on different Athena SQL queries. Manual approved required for the next step.
    - Deploy to production activates the healthcheck for a quality assurance.

Second exercise – Architecture

- (3) Modifying code on the dataflow without stopping it
  - Best Practices
    - Parameterization and templates allow DataOps teams to reduce the complexity (DRY - Don’t Repeat Yourself). Use of AWS Glue Custom Blueprint is recommended
    - Configuration orchestration: AWS CloudFormation allow infrastructure to be created, modified, or destroyed with configuration files (Infrastructure as Code (IaC))
    - Source Code repository must be used for job scripts files, unittest scripts files, integration tests scrips files, CloudFormation configuration files (about ll devices required to construct the integration test environment and the stagged environment), Dockerfiles, CloudBuild configuration files, AWS Glue blueprint files etc
    - Appropriate use of pull request.

Second exercise – Architecture

- (3) Modifying code on the dataflow without stopping it
  - Almost zero downtime scenario
    - Considering the proposed approach for a batch ingestion dataflow, the appropriate use of a CI/CD process as described in the previous slides are enough.
    - As the serving layer and the ETL are decouple, it will not be relevant if the ETL will have a certain downtime during the CloudBuild execution (when necessary).
  - Zero downtime scenario
    - Considering a streaming ingestion and a necessity of real time data analysis, a more complex (and expensive) approach must be adopted.
    - A blue-green deployment at the production environment is a possible approach. In this case we have two completely identical parallel environment. When the blue is in production the green is waiting for new releases (and vice-versa). The switch between blue and green will require an orchestration of the Glue Data Catalog using an expert Lambda Function, as this feature is not available yet in Apache Athena.

References

- Saurabh Gupta, Venkata Giri. Practical Enterprise Data Lake Insights: Handle Data-Driven Challenges in an Enterprise Big Data Lake. June 2018 (Apress)
- Joe Reis, Matt Housley. Fundamentals of Data Engineering. June 2022 (O'Reilly Media, Inc.)
- Harvinder Atwal. Practical DataOps: Delivering Agile Data Science at Scale. December 2019 (Apress)