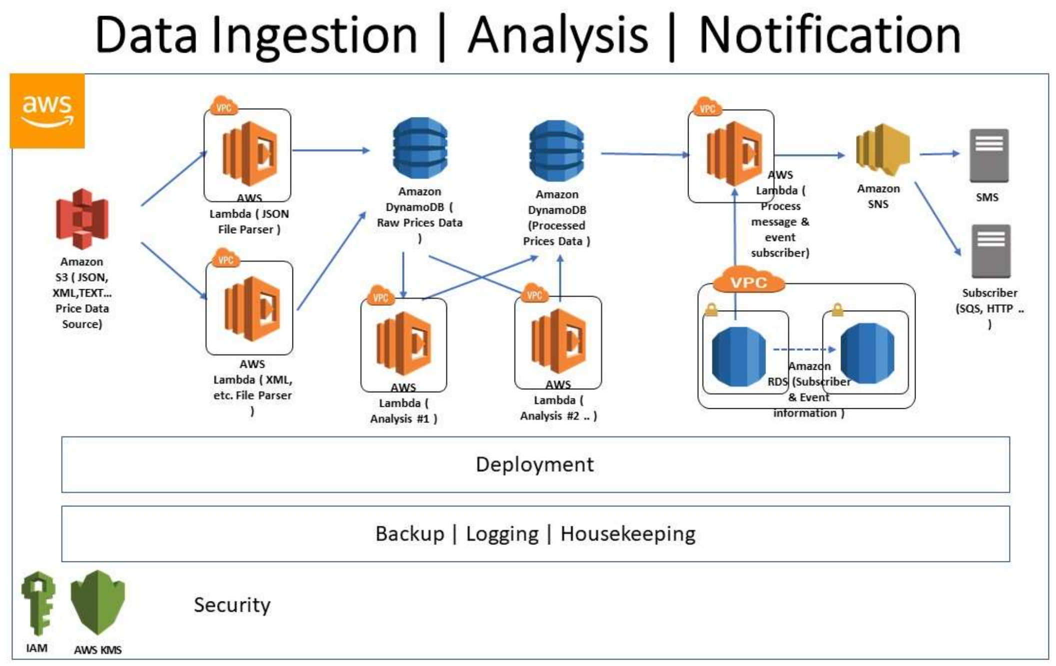
Data Ingestion | Analysis | Notification
aws
Amazon S3 (JSON, XML,TEXT... Price Data Source)
VPC
AWS Lambda (JSON File Parser)
AWS Lambda (XML, etc. File Parser)
Amazon DynamoDB (Raw Prices Data)
Amazon DynamoDB (Processed Prices Data)
AWS Lambda (Analysis #1)
AWS Lambda (Analysis #2 ..)
AWS Lambda (Process message & event subscriber)
Amazon SNS
SMS
Subscriber (SQS, HTTP ..)
Amazon RDS (Subscriber & Event information)
Deployment
Backup | Logging | Housekeeping
Security
IAM
AWS KMS

Estimated Time Taken

- Implementation (Build, Debug, Test) ~ 7 hrs.
- Research ~ 6 hrs.
- Design ~ 6 hrs.
- Documentation ~ 3 hrs.

Implementation

- Data Ingestion Module
  - Pre-requisite
    - AWS CLI configuration setup with Access Key ID and Access Key to a valid AWS account that has the right to launch AWS resources.
    - Create a role "aws-lambda-s3-execution-role" for testing and attached the following policy
      - AmazonS3FullAccess
      - AWSLambdaExecute
      - <Create an inline policy to allow Lambda to write item to DynamoDB>
    - Replace the <AWS account id> place holder in the ARN values with a valid AWS account id, in the following files
      - test-cloudformation.bat
      - test-lambda.json
  - Execute CloudFormation
    - Execute the following from Windows command prompt "test-cloudformation.bat"
  - Remark
    - The CloudFormation will create the following stack

            - test-dynamodb
            - test-lambda
            - test-s3destbucket
            - test-s3srcbucket

Data Ingestion Module

- Components
    - S3
        - Container for file drop.
        - Cost efficient scalable object storage, without required to size the storage in advance. Thus, save effort in sizing.
    - Lambda
        - To parser the raw data file from S3 bucket. The function will retrieve the files from S3, parser with reference to the format. The output will be write to DynamoDB.
        - It will be triggered whenever raw data file arrived in S3.
    - DynamoDB
        - Using as datastore, to store the raw (semi-processed) price data after Lambda parser the files.
        - The use case here does not have complex relationship between the data, thus a NoSQL DB is used. Data points are “time vs value”.
        - Store only raw (semi-processed) price data.
- Assumption
    - Files are drop individually into S3 bucket. Files are removed after picked up for processing.
    - Data are publicly available, there are no confidential information.
    - Data are informational, no complex relationship between the data required.
    - Data are not backup, as data obsolete with time.
    - Lambda function is optimized to be able to complete processing within Lambda limitation.
    - File parser will not perform any error handling.
    - Expected heavy read/write of data into DynamoDB in the future.
    - AWS services are highly available and scalable.
    - Minimum effort to maintain the platform, where the application running on.

Data Analysis Module

- Components
    - Lambda
        - Read the processed data off raw (semi-processed) price data DynamoDB datastore and perform the necessary analysis. And write processed price data DynamoDB datastore.
        - Triggered whenever raw data arrived into the raw (semi-processed) DynamoDB datastore.
    - DynamoDB

        - Uses as datastore, to store processed data after Lambda process the raw (semi-processed) price data.
        - The use case here does not have complex relationship between data, thus a NoSQL DB is used. Data points are “time vs value”.
        - Store only processed price data.
        - A separate DynamoDB is used to cater for future increase in the number data ingestion module, for different types of data, and data analysis module, for different types of analysis.
- Assumption
    - Data are publicly available, there are no confidential information.
    - Data are informational, no complex relationship between the data required.
    - Data backup is not critical, as data are disposable.
    - Lambda function is optimized to be able to complete processing within Lambda limitation.
    - Catered for high read/write capacity in the future.
    - AWS services are highly available and scalable.
    - Minimum effort to maintain the platform, where the application running on.

Notification Module

- Components
    - Lambda
        - To check the list of event subscriber who subscribed to the events triggered, event type, delivery methods, etc.
        - Prevent spamming of notification, by record the number of triggered by an event before dispatching a notification to subscriber.
        - Triggered whenever data are written into the processed data DynamoDB.
    - RDS (MySQL)
        - Used to store subscriber’s endpoint information, event subscriptions information, subscription events, record count before notification, etc.
        - Created in a private subnet in a VPC.
        - Backup are configured.
        - Cater for read replica, if required in the future.
    - SNS
        - Allow the expansion to other notification delivery methods example SQS, Lambda, HTTP, etc.
- Assumption
    - No records of events notification dispatched to the subscriber. Dependent on AWS CloudWatch to log all events.
    - No retry of event dispatching, whenever there is any error or failure for the notification to reach the subscriber.
    - Lambda function is optimized to be able to complete processing within Lambda limitation.

    - Subscriber’s endpoint information, event subscriptions information, subscription events, etc. are inserted by MySQL utilities. Administration module will be build as future enhancement.
    - AWS services are highly available and scalable.
    - Minimum effort to maintain the platform, where the application running on.

Security

- IAM
    - Create at least a group for Administrator and Developer
        - Administrators have the policy to create/destroy AWS services, add/remove developer or administrator.
        - Developers have the policy to read CloudWatch logs for debugging.
        - All users must be MFA enabled and password expiry policies enabled.
    - Create individual system user, attached with policies to ensure AWS resource has the required access to another AWS resource.
- S3
    - Bucket ACL configured only with “List Bucket” and “Read Bucket Permission” for Lambda service to retrieve and read raw data files drop into S3.
    - Data are publicly available, there are no confidential information. Thus, do not required versioning and encryption.
- Lambda
    - Lambda used in the ingestion and analysis module is not placed in a VPC, as all the data are publicly accessible information.
    - Only the Lambda used in the notification module will be place in a VPC, as the Lambda function required to access subscriber information from the RDS.
- DynamoDB
    - Secured with IAM.
- SNS
    - Secured by launching SNS instance in a VPC.
    - Lambda can publish message within the VPC to the SNS for dispatching to subscriber endpoint.
- RDS
    - Spin up in a VPC private subnet.
    - Create a bastion/Jump host to perform data insertion/deletion.
    - KMS enable to encrypt subscriber endpoint information, for data protection.

Deployment

- The required AWS services, will be launched using CloudFormation JSON/YAML artefacts. These artefacts will be version control with a VCS system using CodeCommit services, with corresponding release branching strategy.
- CodePipeline will be used to deploy the CloudFormation JSON/YAML artefacts for testing in a SIT/UAT environment before deployed to production.

Backup | Logging | Housekeeping

- The entire infrastructure and the Lambda functions are written in CloudFormation JSON/YAML artefact and version controlled by CodeCommit.
- RDS (MySQL) is deployed as multi-AZ setup, with a slave database acting as a backup for the master database.
- CloudWatch will be the main logging mechanism to log all activities.
- CloudTrail will be used to log all user activities, API call to resources, and where it is call. This can be used for security audit purposes if need arises.
- Housekeeping job example batch job will be used to purge obsolete data from the DynamoDB, depending on the required retention period.

Design Limitation

- Limited by Lambda time-out period, memory, and other limitation.
- No handling errors nor error recovery when notification failed to reach subscriber endpoint.
- Challenging to trace CloudWatch logs if required.
- Potential single point of failure at SNS in the Notification module.
- Limited defense against cyberattack example DOS at Lambda or S3.

Design Trade-Of

- Unable to perform complex computation analysis or parser of complex file format, as need to consider Lambda limitation when implementing Lambda function.
- Dependent on AWS Cloud Provider for HA and Fault Tolerance in their services.
- Potentially tied to AWS Cloud Provider and “expensive” to migrate to other Cloud Provider in the future.