Build an ETL Pipeline on EMR using AWS CDK and Power BI

CookBook

Table of Content:

Introduction

Serverless architecture is a cloud computing model in which the cloud provider manages the infrastructure and automatically provisions, scales, and manages the resources required to run applications. With serverless architecture, developers can focus on writing and deploying code without having to manage or provision servers.

In a traditional server-based model, developers need to set up and manage servers to run their applications. This involves tasks such as capacity planning, server configuration, and scaling. In contrast, serverless architecture abstracts away the underlying infrastructure, allowing developers to execute code in a highly scalable and event-driven manner.

Key characteristics of serverless architecture include:

1. **Function-as-a-Service (FaaS):** Serverless often refers to FaaS, where developers write code in the form of functions. These functions are triggered by specific events or requests, and the cloud provider handles the execution and resource allocation.
2. **Event-driven:** Serverless applications are typically event-driven, meaning they respond to events such as HTTP requests, database changes, file uploads, or scheduled events. When an event occurs, the associated function is invoked to handle it.
3. **Automatic scaling:** Serverless platforms automatically scale the resources allocated to the functions based on the incoming workload. This ensures that applications can handle varying levels of traffic and workload without manual intervention.
4. **Pay-per-use billing:** With serverless architecture, you only pay for the actual execution time and resources consumed by your functions. The billing is typically based on the number of invocations, duration, and resources utilized.

Benefits of serverless architecture include:

1. **Reduced operational overhead:** Developers can focus on writing code and delivering functionality without managing servers or infrastructure.
2. **Improved scalability:** Serverless platforms can automatically scale your application based on demand, ensuring that it can handle traffic spikes and scale down during periods of low usage.
3. **Cost optimization:** Since you only pay for the actual usage of functions, serverless architecture can be cost-effective for applications with variable workloads or sporadic traffic.
4. **Faster time to market:** Serverless allows developers to quickly deploy and iterate on their code, reducing the time required for infrastructure setup and configuration.

However, serverless architecture may not be suitable for all types of applications. Long-running or continuously running processes may not be a good fit for the event-driven nature of serverless functions. Additionally, applications with strict latency or resource requirements may require more control over the underlying infrastructure.

Overall, serverless architecture provides a flexible and efficient approach to building and deploying applications, enabling developers to focus on writing code and delivering value.

AWS CDK (Cloud Development Kit) is an open-source software development framework provided by Amazon Web Services (AWS) that allows developers to define infrastructure resources in code. It enables you to provision and manage AWS infrastructure using familiar programming languages such as TypeScript, JavaScript, Python, Java, and C#.

The AWS CDK provides a higher-level abstraction over AWS CloudFormation, which is AWS's native infrastructure-as-code service. With CDK, you can define your AWS resources and their relationships using object-oriented programming constructs. This approach allows you to leverage the benefits of programming languages, such as code reusability, modularity, and type safety, to define and manage your infrastructure.

Key features and benefits of AWS CDK include:

- **Familiar programming languages:** CDK supports popular programming languages, allowing developers to use their existing skills and knowledge to define infrastructure. This makes it easier to create and maintain infrastructure code.
- **Infrastructure-as-code:** CDK enables you to define your infrastructure resources, including Amazon EC2 instances, Amazon S3 buckets, AWS Lambda functions, Amazon RDS databases, and more, using code. This makes your infrastructure version-controlled, repeatable, and easy to manage.
- **Higher-level constructs:** CDK provides higher-level constructs called "constructs" that encapsulate common patterns and best practices for provisioning AWS resources. These constructs abstract away the low-level details of CloudFormation, making it easier and faster to define and deploy infrastructure.
- **Resource provisioning and deployment:** CDK simplifies the process of provisioning and deploying AWS resources. It automatically generates CloudFormation templates based on your CDK code and deploys the resources using CloudFormation stack updates.
- **AWS resource discovery:** CDK offers a powerful CLI (Command Line Interface) that allows you to discover and explore AWS resources in your account. This helps you understand the current state of your infrastructure and makes it easier to incorporate existing resources into your CDK projects.
- **Ecosystem and community:** CDK has a vibrant ecosystem and community that contribute and share reusable constructs, patterns, and best practices. This enables you to leverage the collective knowledge and experience of the community to accelerate your infrastructure development.

To use CDK, you would typically define your infrastructure using the CDK constructs in your chosen programming language. Then, you can use the CDK CLI to synthesize the CDK code into CloudFormation templates and deploy those templates to create or update your AWS resources.

AWS CDK offers a powerful and efficient way to define, provision, and manage AWS infrastructure using code, enabling you to adopt infrastructure-as-code practices and leverage the benefits of modern software development for your cloud infrastructure.

AWS CDK (Cloud Development Kit) has gained popularity in the industry because it addresses several challenges and provides significant benefits in cloud infrastructure development. Here are some reasons why AWS CDK is considered the need of the hour:

- **Increased productivity:** CDK allows developers to define infrastructure using familiar programming languages. This reduces the learning curve and enables developers to leverage their existing skills and knowledge. By using high-level constructs and reusable components, developers can write less code and achieve more in less time.
- **Infrastructure as code (IaC) best practices:** CDK promotes the use of infrastructure as code, which brings numerous benefits such as version control, code review, automated deployments, and the ability to treat infrastructure changes as code changes. This leads to better collaboration, repeatability, and reliability in infrastructure provisioning.
- **Flexibility and abstraction:** CDK provides a higher-level abstraction over AWS CloudFormation, making it easier to define and manage complex infrastructure resources. It abstracts away the details of CloudFormation, allowing developers to focus on the desired state of their infrastructure rather than the underlying implementation.
- **Rapid iteration and experimentation:** CDK enables rapid iteration and experimentation by allowing developers to modify their infrastructure code and quickly deploy changes. This agility is crucial in fast-paced development environments and when responding to evolving business needs.
- **Ecosystem and community support:** CDK has a vibrant and growing ecosystem of constructs and libraries contributed by AWS and the community. These ready-to-use components accelerate development and provide best-practice patterns for common infrastructure requirements. The community support and knowledge-sharing further enhance the capabilities of CDK.
- **Multi-language support:** CDK supports multiple programming languages, including TypeScript, JavaScript, Python, Java, and C#. This language flexibility allows developers to choose the language they are most comfortable with, fostering productivity and reducing language-specific barriers.
- **Integration with AWS services:** CDK seamlessly integrates with the wide range of AWS services, enabling developers to provision and manage various resources in a unified manner. This tight integration simplifies the development and management of complex, multi-service architectures.

Overall, AWS CDK offers a modern, developer-friendly approach to infrastructure provisioning and management. It improves productivity, promotes infrastructure as code best practices, and provides flexibility and abstraction, all while integrating with the rich AWS ecosystem. With the growing demand for scalable and efficient cloud infrastructure, CDK has emerged as a valuable tool for organizations adopting AWS services.

Data Description

A sales dataset typically consists of structured data that records various aspects of sales transactions and related information. It is a valuable resource for analyzing and understanding sales performance, identifying trends, making predictions, and optimizing sales strategies. The dataset can include several fields or columns, each providing specific details about the sales transactions. Here is a description of some common fields found in a sales dataset:

- **Region:** This column represents the geographical region where the sale occurred. It could include values like "North America," "Europe," "Asia," etc.
- **Country:** This column specifies the country where the sale took place, providing more specific location information.
- **Item Type:** This column describes the type or category of the item sold. It could include values such as "Electronics," "Clothing," "Home Appliances," etc.
- **Sales Channel:** This column indicates the channel through which the sale was made. It could include values like "Online," "Retail Store," "Distributor," etc.
- **Order Priority:** This column captures the priority level assigned to the order. It could include values like "High," "Medium," "Low," etc., indicating the urgency or importance of fulfilling the order.
- **Order Date:** This column records the date when the order was placed by the customer.
- **Order ID:** This column represents a unique identifier assigned to each order placed. It helps in tracking and referencing specific orders.
- **Ship Date:** This column records the date when the ordered items were shipped to the customer.
- **Units Sold:** This column indicates the number of units or items sold in the transaction.
- **Unit Price:** This column represents the price of each unit or item sold.
- **Unit Cost:** This column captures the cost associated with producing or acquiring each unit or item sold.
- **Total Revenue:** This column calculates the total revenue generated from the sale by multiplying the units sold with the unit price.
- **Total Cost:** This column represents the total cost associated with the sale by multiplying the units sold with the unit cost.
- **Total Profit:** This column calculates the total profit earned from the sale by subtracting the total cost from the total revenue.

These columns provide detailed information about the sales transactions, allowing for analysis of sales performance, profitability, and other relevant insights.

Services used in the project

AWS S3

Amazon S3 (Simple Storage Service) is a cloud-based object storage service provided by Amazon Web Services (AWS). It allows businesses and individuals to store and retrieve data from anywhere on the internet. Amazon S3 is designed to be highly scalable, reliable, and cost-effective, making it a popular choice for storing a wide variety of data, including images, videos, documents, and backups.

Some of the key features of Amazon S3 include:

1. **Scalability:** Amazon S3 is highly scalable and can handle storage needs ranging from a few gigabytes to multiple petabytes.
2. **Security:** Amazon S3 provides several security features to protect data, including encryption in transit and at rest, access controls, and multi-factor authentication.
3. **Durability and availability:** Amazon S3 store data across multiple data centers, providing high durability and availability. It also provides a service-level agreement (SLA) for availability of 99.999999999% (11 nines).
4. **Integration:** Amazon S3 integrates with a wide range of AWS services, including compute, database, analytics, and machine learning services.
5. **Cost-effective:** Amazon S3 offers a pay-as-you-go pricing model, allowing customers to pay only for the storage and data transfer they use.

Here are some disadvantages of using Amazon S3:

1. **Complexity:** The various features and configuration options of Amazon S3 can make it difficult to set up and manage for users without a technical background.
2. **Cost:** Although Amazon S3 can be cost-effective for smaller storage needs, costs can quickly add up for large volumes of data and high usage.
3. **Data transfer fees:** There are additional fees for transferring data in and out of Amazon S3, which can increase costs for businesses with high data transfer needs.
4. **Performance:** While Amazon S3 offers high durability and availability, its performance can be slower compared to other storage options for certain use cases.
5. **Vendor lock-in:** Moving data out of Amazon S3 to another storage solution can be challenging, potentially creating a vendor lock-in situation.

Overall, Amazon S3 is a powerful and flexible storage solution that can be used for a wide range of use cases, from backup and disaster recovery to big data analytics and web hosting.

Amazon Lambda

Amazon Lambda, often referred to as AWS Lambda, is a serverless computing service provided by Amazon Web Services (AWS). It allows you to run your code without provisioning or managing servers. With Lambda, you can focus on writing and deploying your code while AWS handles the infrastructure scaling and management for you.

Key features and characteristics of AWS Lambda include:

- **Event-driven computing:** Lambda functions are triggered by events such as changes to data in an Amazon S3 bucket, updates to a database, HTTP requests through API Gateway, scheduled events, or custom events from other AWS services. When an event occurs, Lambda automatically runs your code.
- **Scalability:** Lambda automatically scales your code in response to the incoming event rate. It provisions the necessary resources to handle the current workload, and you are only billed for the actual execution time of your code.
- **Pay-per-use pricing:** AWS Lambda follows a pay-per-use pricing model. You are billed for the number of requests and the duration of your code's execution, measured in milliseconds. There is no charge when your code is not running.
- **Supported languages:** Lambda supports a variety of programming languages, including Python, Node.js, Java, C#, Ruby, PowerShell, and Go. You can choose the language that best suits your application and development preferences.
- **Integration with other AWS services:** Lambda integrates seamlessly with various AWS services, allowing you to build serverless architectures and workflows. It can be used in combination with services such as Amazon S3, Amazon DynamoDB, Amazon API Gateway, Amazon SQS, and more.
- **Customizable runtime environments:** You can create custom runtime environments for Lambda functions by using AWS Lambda Layers. This enables you to package and share libraries, dependencies, and other code components across multiple Lambda functions.
- **Versioning and aliases:** Lambda provides versioning and aliasing capabilities, allowing you to manage different versions of your functions and create aliases to point to specific versions. This enables you to implement controlled deployments and rollback strategies.

AWS Lambda offers a wide range of use cases, including data processing, real-time file processing, backend services, IoT (Internet of Things) applications, chatbots, web applications, and more. It abstracts away the underlying infrastructure, allowing you to focus on writing business logic and building scalable applications.

To use AWS Lambda, you define your code as a Lambda function, configure the triggers or events that invoke your function, and deploy it to AWS Lambda. You can manage your Lambda functions and monitor their performance through the AWS Management Console, CLI (Command Line Interface), or SDKs (Software Development Kits).

AWS Lambda provides an efficient and cost-effective way to execute code in a serverless manner, eliminating the need for server management and enabling rapid scaling and event-driven application architectures.

Amazon EMR

Amazon EMR (Elastic MapReduce) is a cloud-based big data processing service provided by Amazon Web Services (AWS). It allows users to process and analyze large datasets using popular frameworks such as Apache Spark, Apache Hadoop, Apache Hive, and Apache HBase, among others.

EMR simplifies the process of setting up, configuring, and managing a big data infrastructure. It provides a managed environment where users can launch and scale clusters on-demand, enabling them to process and analyze large datasets efficiently. EMR also integrates with other AWS services, allowing users to seamlessly store data in Amazon S3, utilize Amazon EC2 instances for cluster computation, and use AWS Identity and Access Management (IAM) for access control.

Key features and components of Amazon EMR include:

1. **Cluster Management:** EMR allows users to create and manage clusters, which are collections of EC2 instances configured to perform big data processing tasks. Users can customize cluster settings, select different instance types, and add or remove instances dynamically as needed.
2. **Data Processing Frameworks:** EMR supports a variety of big data processing frameworks, including Apache Spark, Apache Hadoop (MapReduce), Apache Hive, Apache Pig, and Apache HBase. These frameworks enable users to perform tasks such as distributed data processing, data querying, batch processing, and real-time analytics.
3. **Data Storage:** Amazon EMR integrates with Amazon S3 for storing input and output data. Users can easily access and process data stored in S3 buckets, enabling efficient data handling and reducing data transfer costs.
4. **Scaling:** EMR allows users to scale their clusters dynamically based on workload requirements. Users can add or remove instances from the cluster, adjusting the compute resources allocated to the processing tasks.
5. **Integration with Other AWS Services:** EMR integrates with various other AWS services. For example, users can use AWS Glue for data cataloging and metadata management, Amazon Redshift for data warehousing, and Amazon Kinesis for real-time streaming data processing.
6. **Security and Access Control:** EMR provides security features such as encryption for data at rest and in transit, integration with AWS IAM for user access control, and VPC (Virtual Private Cloud) isolation for secure network connectivity.

Amazon EMR is a flexible and scalable solution for processing and analyzing large datasets in a distributed manner. It allows users to leverage popular big data frameworks and easily manage their big data infrastructure, making it a valuable tool for data processing, analytics, and machine learning applications.

AWS CDK

AWS CDK (Cloud Development Kit) is an open-source software development framework provided by Amazon Web Services (AWS). It allows developers to define cloud infrastructure resources in code using familiar programming languages such as TypeScript, JavaScript, Python, Java, and C#. CDK enables you to provision, update, and manage your AWS infrastructure in a programmatic and efficient manner.

Key features and benefits of AWS CDK include:

- **Infrastructure as code (IaC):** CDK allows you to define your cloud infrastructure using code. This brings the benefits of version control, code review, and the ability to treat infrastructure changes as code changes. It also provides repeatability and consistency in provisioning and managing infrastructure resources.
- **Familiar programming languages:** With CDK, you can use programming languages that you are already familiar with to define your infrastructure. This reduces the learning curve and enables you to leverage your existing skills and knowledge.
- **Construct library:** CDK provides a construct library that offers pre-built, reusable components representing AWS resources and services. These constructs encapsulate best practices and make it easier to define infrastructure patterns and architectures. The library is continuously growing, and there is an active community contributing new constructs.
- **Multi-language support:** CDK supports multiple programming languages, allowing developers to choose the language they prefer. You can define your infrastructure in TypeScript, JavaScript, Python, Java, or C#, depending on your needs and comfort.
- **Easy provisioning and deployment:** CDK simplifies the process of provisioning and deploying infrastructure resources. It automatically generates AWS CloudFormation templates based on your CDK code and deploys them using CloudFormation stack updates. CDK also provides convenient CLI commands for managing your infrastructure.
- **Integration with AWS services:** CDK seamlessly integrates with various AWS services, enabling you to define and provision a wide range of resources, including compute instances, storage, databases, networking components, security groups, and more. You can leverage the full power of AWS services while defining your infrastructure in code.
- **Continuous integration and delivery (CI/CD) integration:** CDK can be easily integrated into CI/CD pipelines, allowing you to automate the deployment and management of your infrastructure as part of your software development lifecycle.

Overall, AWS CDK offers a powerful and developer-friendly approach to provisioning and managing cloud infrastructure. It combines the benefits of infrastructure as code, familiar programming languages, a construct library, and easy integration with AWS services. With CDK, you can efficiently define and manage your AWS infrastructure, enabling agile development and rapid iteration.

AWS Cloud9

AWS Cloud9 is an integrated development environment (IDE) provided by Amazon Web Services (AWS). It offers a cloud-based development environment for writing, running, and debugging code. Cloud9 eliminates the need for developers to set up and manage their own development environments, as everything is provisioned and hosted in the cloud.

Key features and capabilities of AWS Cloud9 include:

- Web-based IDE: Cloud9 provides a web-based development environment accessible through a web browser. It supports various programming languages such as Python, JavaScript, Java, C++, and more.
- Collaboration: Cloud9 allows multiple developers to collaborate on the same codebase in real-time. You can share your development environment with team members, enabling them to edit code, provide feedback, and collaborate seamlessly.
- Code editing and navigation: Cloud9 offers a rich code editor with features like syntax highlighting, autocompletion, code folding, and code navigation. It provides a familiar IDE experience for developers to write and edit code efficiently.
- Terminal access: Cloud9 includes a built-in terminal with a Linux shell. Developers can execute commands, run scripts, and manage their development environment directly from the terminal.
- Code debugging: Cloud9 integrates with popular debugging tools, allowing developers to debug their code within the IDE. It supports step-by-step debugging, breakpoints, and variable inspection for various programming languages.
- Version control integration: Cloud9 seamlessly integrates with version control systems such as Git. You can clone, commit, push, and pull code directly from within the IDE, making it easy to collaborate and manage code repositories.
- AWS service integration: Cloud9 provides tight integration with other AWS services, making it easy to develop and deploy applications on AWS. You can interact with AWS resources, configure AWS CLI, and access AWS SDKs directly from the IDE.
- Customization and extensibility: Cloud9 allows customization of the IDE environment. You can install additional software, plugins, and extensions to enhance the development experience and meet specific requirements.
- Serverless development: Cloud9 provides native support for serverless development using AWS Lambda. It simplifies the creation, testing, and deployment of serverless applications, allowing you to focus on writing code.

AWS Cloud9 offers a flexible and convenient development environment for building applications in the cloud. It streamlines the development workflow, promotes collaboration among team members, and integrates seamlessly with AWS services. With Cloud9, developers can code, debug, and deploy applications with ease, leveraging the power of the cloud to accelerate development cycles.

Power BI

Power BI is a business analytics tool developed by Microsoft. It provides interactive visualizations and business intelligence capabilities, allowing users to analyze data, share insights, and make data-driven decisions. Power BI can connect to a wide range of data sources, including databases, spreadsheets, cloud services, and online services.

Key features and components of Power BI include:

- **Data Preparation:** Power BI allows users to import and transform data from various sources, perform data cleansing and shaping operations, and create data models. It provides a user-friendly interface for data preparation tasks, enabling users to merge, filter, and transform data easily.
- **Data Visualization:** Power BI offers a wide range of interactive visualizations, including charts, tables, maps, and gauges. Users can drag and drop data fields onto the canvas to create visual representations of their data. The visuals can be customized and formatted to meet specific requirements.
- **Dashboards and Reports:** Power BI allows users to create interactive dashboards and reports that provide a consolidated view of their data. Dashboards can display multiple visualizations and provide real-time updates. Reports provide more detailed insights and can include multiple pages with visuals, filters, and interactive features.
- **Sharing and Collaboration:** Power BI enables users to share dashboards, reports, and datasets with others in their organization. It supports collaboration features such as sharing permissions, comments, and version control. Users can also create and participate in data-driven discussions using Power BI's collaboration tools.
- **Data Exploration and Analysis:** Power BI provides powerful data exploration capabilities, allowing users to drill down into their data, filter data based on criteria, and create ad-hoc calculations and measures. It also supports natural language queries, where users can ask questions about their data using plain language and get instant visual responses.
- **Integration with Other Tools:** Power BI integrates seamlessly with other Microsoft products and services, such as Excel, SharePoint, and Azure. It also supports connectivity to third-party applications and services, allowing users to leverage data from different sources and create comprehensive analyses.
- **Mobile Access:** Power BI offers mobile apps for iOS and Android devices, enabling users to access and interact with their dashboards and reports on the go. The mobile apps provide a responsive and optimized experience for viewing and exploring data on mobile devices.

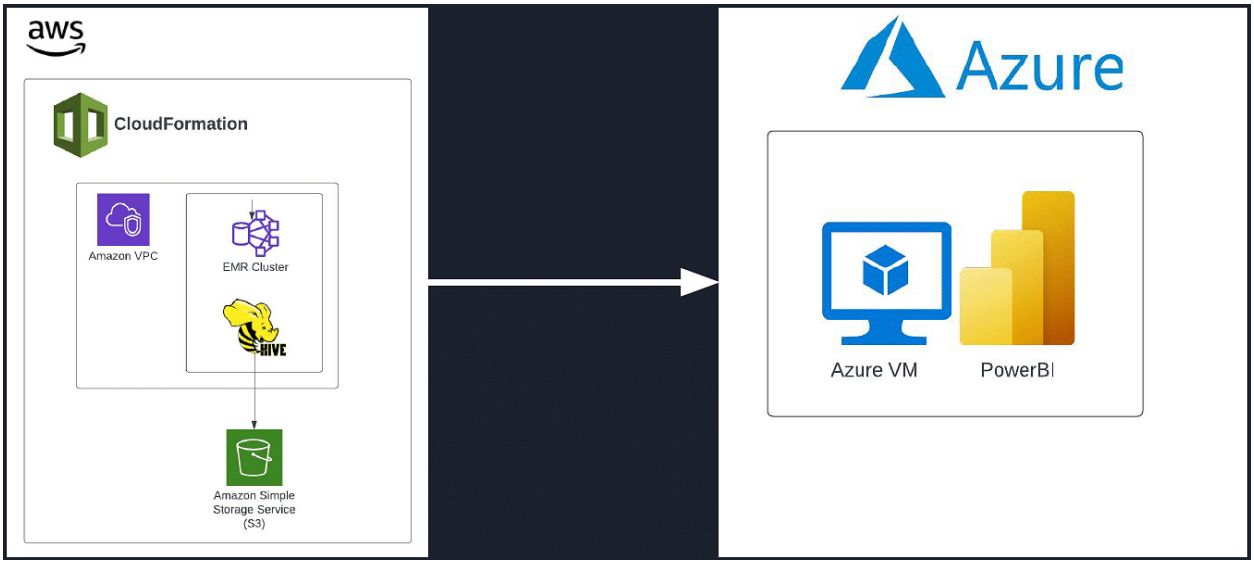
Power BI provides organizations with a powerful and intuitive tool for data analysis and visualization. It enables users to transform raw data into meaningful insights, share those insights with others, and make data-driven decisions across various departments and functions.

Use-case depicted in this project

The main goal of this project is to develop an Extract, Transform, Load (ETL) Pipeline using Amazon EMR, which will be implemented through the AWS Cloud Development Kit (CDK). The pipeline's primary purpose is to perform data analysis and transformation tasks using Apache Hive, a powerful data warehouse infrastructure running on EMR. By leveraging the capabilities of EMR, we can efficiently process and manipulate large datasets to extract valuable insights.

Furthermore, as part of this project, we will create an interactive and visually appealing dashboard using Power BI. This dashboard will serve as a platform to present the results of the data analysis in a clear and intuitive manner. With the help of Power BI's rich set of visualization tools and features, we will be able to generate informative charts, graphs, and other visual elements that effectively convey the discovered patterns, trends, and correlations within the dataset. This visualization layer will enable users to explore and interact with the analyzed data, making it easier to understand and derive actionable insights from the analysis.

The Architecture of the pipeline is as follows:



Breakdown of the Use-Case

- Setup required:
  - Create an AWS account
  - Download the dataset
  - Download the Code
  - This is an OS-independent project, as the whole pipeline is set up on AWS Cloud

- Ways to interact with the AWS services:
  - Console- Using the Web-based UI
  - CLI- Using the AWS CLI tool in Terminal/CMD
  - CDK- Using infrastructure-as-a-code, e.g., CloudFormation

  - SDK- Programmatically, e.g., 'boto3' in python
  - For this project, we will not be using the CDK(CloudFormation); instead, the rest of the three methods will be used for execution.

- Best Practices for AWS Account
  - Enable Multi-Factor Authentication(MFA)
  - Protect the account using IAM policies
  - Avoid using the root account
  - Choose the AWS region that is closest to you
  - Create a Budget and set up a notification
  - Rotate all keys and passwords periodically
  - Follow least privilege principle

AWS Cloud9

Let’s first create an AWS Cloud9 environment for our project.

To create an AWS Cloud9 environment, you can follow these steps:

1. **Open the AWS Management Console**: Go to the AWS Management Console (https://console.aws.amazon.com) and sign in to your AWS account.
2. **Navigate to Cloud9:** Use the AWS services search bar or navigate to the "Developer Tools" category and select "Cloud9."
3. **Click "Create environment":** On the Cloud9 dashboard, click the "Create environment" button.
4. **Provide environment details:**
    - Enter a name for your environment.
    - Optionally, enter a description.
    - Choose the environment type (new or clone an existing environment).
    - Select the instance type based on your requirements.
    - Choose the platform (Amazon Linux 2 or Ubuntu).
5. **Configure settings:**
    - Choose the network settings (either create a new Amazon VPC or use an existing one).
    - Select the subnet for your environment.
    - Choose the IAM role that Cloud9 will use to access AWS resources on your behalf.
    - Configure additional settings as needed.
6. **Review and create:**
    - Review the configuration details.
    - Enable the option to create an AWS CloudFormation stack if desired.
    - Click "Create environment" to start the provisioning process.
7. **Wait for the environment to be created:** The Cloud9 environment creation process may take a few minutes. You can monitor the progress on the Cloud9 dashboard.
8. **Access the Cloud9 IDE:** Once the environment is created, you can click on its name in the Cloud9 dashboard to access the Cloud9 integrated development environment (IDE) in your web browser.

That's it! You have successfully created an AWS Cloud9 environment. From the Cloud9 IDE, you can start coding, running commands, collaborating with team members, and leveraging the various features and capabilities offered by Cloud9 for your development tasks.

Cloning GitHub repository to AWS Cloud9

To link and clone GitHub to AWS Cloud9, you can follow these steps:

1. **Open your AWS Cloud9 environment:** Go to the AWS Management Console (https://console.aws.amazon.com) and navigate to the Cloud9 service. Open the Cloud9 environment you want to link with GitHub.

2. **Configure Git credentials:** In your Cloud9 environment, open a new terminal by clicking on the "Window" menu and selecting "New Terminal." Run the following commands to configure your Git credentials:

```
git config --global user.name "Your GitHub Username"
git config --global user.email "Your GitHub Email"
```

3. **Generate and add an SSH key:** To securely connect Cloud9 with your GitHub account, you need to generate an SSH key pair and add the public key to your GitHub account. Run the following command in your Cloud9 terminal:

```
ssh-keygen -t rsa -b 4096 -C "Your GitHub Email"
```

   This will generate a new SSH key pair. Press Enter to accept the default file location and optionally enter a passphrase for added security.

4. **View and copy the public key:** Run the following command to display your public key:

```
cat ~/.ssh/id_rsa.pub
```

   Copy the entire contents of the public key that is displayed in the terminal.

5. **Add the public key to your GitHub account:** Go to your GitHub account settings (https://github.com/settings/profile) and navigate to the "SSH and GPG keys" section. Click on "New SSH key" and give it a descriptive title. Paste the public key you copied in the previous step and click "Add SSH key."

6. **Test the connection:** To test if the SSH connection between Cloud9 and GitHub is successful, run the following command in the Cloud9 terminal:

```
ssh -T git@github.com
```

   You should see a success message indicating that you've successfully authenticated with GitHub.

7. **Clone a GitHub repository:** In your Cloud9 environment, navigate to the directory where you want to clone the GitHub repository. Run the following command to clone the repository:

```
git clone git@github.com:username/repository.git
```

Replace username with your GitHub username and repository with the name of the repository you want to clone.

That's it! You have successfully linked GitHub to your AWS Cloud9 environment. You can now collaborate with your GitHub repositories, push and pull changes, and work on your code seamlessly within Cloud9.

AWS CDK: Bucket Deployment Stack

We will load the data and hql scripts in AWS S3 bucket.

Create_tables.hql:

The provided code creates two external tables in Hive. The sales_data_raw table is created with a CSV format, using the OpenCSVSerde SerDe, and it is located in an S3 bucket. The table is configured to skip the header line. The sales_data table is created with a Parquet format, using the ParquetHiveSerDe SerDe, and it is also located in an S3 bucket. It specifies the Parquet input and output formats and uses 'SNAPPY' compression. These tables allow querying and analyzing data stored in the respective S3 locations using Hive's query capabilities.

```
DROP TABLE sales_data_raw;
CREATE EXTERNAL TABLE sales_data_raw (
    `region` STRING,
    `country` STRING,
    `item_type` STRING,
    `sales_channel` STRING,
    `order_priority` STRING,
    `order_date` STRING,
    `order_id` STRING,
    `ship_date` STRING,
    `units_sold` STRING,
    `unit_price` STRING,
    `unit_cost` STRING,
    `total_revenue` STRING,
    `total_cost` STRING,
    `total_profit` STRING
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.serde2.OpenCSVSerde'
LOCATION
  's3://emr-pipeline-c2be3our7i/emr_pipeline/data/sales_data_raw/'
TBLPROPERTIES (
    "skip.header.line.count"="1"
);

CREATE EXTERNAL TABLE sales_data (
    `region` STRING,
    `country` STRING,
    `item_type` STRING,
    `sales_channel` STRING,
    `order_priority` STRING,
    `order_date` date,
    `order_id` STRING,
    `ship_date` date,
    `units_sold` INTEGER,
```

```
    `unit_price` DECIMAL(10,2),
    `unit_cost` DECIMAL(10,2),
    `total_revenue` DECIMAL(12,2),
    `total_cost` DECIMAL(12,2),
    `total_profit` DECIMAL(12,2)
)
ROW FORMAT SERDE
  'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
STORED AS INPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetInputFormat'
OUTPUTFORMAT
  'org.apache.hadoop.hive.ql.io.parquet.MapredParquetOutputFormat'
LOCATION
  's3://emr-pipeline-c2be3our7i/emr_pipeline/data/sales_data/'
TBLPROPERTIES (
    'parquet.compression'='SNAPPY'
);
```

Transform_data.hql:

The given code is an example of an INSERT statement in Hive. It populates the sales_data table by selecting and transforming data from the sales_data_raw table. The code includes various transformations on the columns during the SELECT statement. It converts the order_date and ship_date columns from string format to the DATE data type using conditional statements and the Unix timestamp functions. It also casts the units_sold, unit_price, unit_cost, total_revenue, total_cost, and total_profit columns to their respective data types (INTEGER and DECIMAL). The SELECT statement retrieves the desired columns from the sales_data_raw table and applies the specified transformations before inserting the transformed data into the sales_data table, overwriting any existing data.

```
INSERT OVERWRITE TABLE sales_data
SELECT
    `region`,
    `country`,
    `item_type`,
    `sales_channel`,
    `order_priority`,
    CASE
      WHEN `order_date` RLIKE '([0-9]{2}\-[0-9]{2}\-[0-9]{4})'
        THEN CAST(from_unixtime(unix_timestamp(`order_date` , 'MM-dd-
yyyy')) AS DATE)
      WHEN `order_date` RLIKE '([0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4})'
        THEN CAST(from_unixtime(unix_timestamp(`order_date` ,
'MM/dd/yyyy')) AS DATE)
      ELSE NULL
    END AS `order_date`,
    `order_id`,
```

```
    CASE
      WHEN `ship_date` RLIKE '([0-9]{2}\-[0-9]{2}\-[0-9]{4})'
        THEN CAST(from_unixtime(unix_timestamp(`ship_date` , 'MM-dd-yyyy'))
AS DATE)
      WHEN `ship_date` RLIKE '([0-9]{1,2}\/[0-9]{1,2}\/[0-9]{4})'
        THEN CAST(from_unixtime(unix_timestamp(`ship_date` , 'MM/dd/yyyy'))
AS DATE)
    END AS `ship_date`,
    CAST(`units_sold` AS INTEGER) AS units_sold,
    CAST(`unit_price` AS DECIMAL) AS unit_price,
    CAST(`unit_cost` AS DECIMAL) AS unit_cost,
    CAST(`total_revenue` AS DECIMAL) AS total_revenue,
    CAST(`total_cost` AS DECIMAL) AS total_cost,
    CAST(`total_profit` AS DECIMAL) AS total_profit
FROM sales_data_raw
```

Bucket_deployment_stack.py

```
from aws_cdk import (
    Stack,
    aws_s3_deployment as s3deploy,
    aws_s3 as s3,
)
from constructs import Construct
from decouple import config

# Deploys data to S3 which triggers the lambda
class BucketDeploymentStack(Stack):

    def __init__(self,
                 scope: Construct,
                 construct_id: str,
                 primary_bucket_name: str,
                 log_bucket_name: str,
                 **kwargs
                ) -> None:
        super().__init__(scope, construct_id, **kwargs)

        primary_bucket = s3.Bucket(
            self,
            id=primary_bucket_name,
            bucket_name=log_bucket_name
        )

        log_bucket = s3.Bucket(
            self,
            id=log_bucket,
            bucket_name=log_bucket
        )

        data_deployment = s3deploy.BucketDeployment(self, "DeployData",
                sources=[s3deploy.Source.asset("./emr_pipeline/")],
                destination_bucket=primary_bucket,
                destination_key_prefix="emr_pipeline/",
                prune=False
            )
```

The provided code is written in Python and uses the AWS CDK (Cloud Development Kit) library to deploy data to an S3 bucket, which in turn triggers a lambda function. The code defines a class called BucketDeploymentStack, which inherits from the Stack class provided by the AWS CDK. The BucketDeploymentStack class has an __init__ method that takes several parameters, including the scope, construct_id, primary_bucket_name, and log_bucket_name.

Within the __init__ method, the code creates two S3 bucket resources using the s3.Bucket class from the AWS CDK library. One bucket is assigned to the primary_bucket variable, and the other is assigned to the log_bucket variable. The bucket names are specified using the bucket_name parameter.

The code then deploys the data to the S3 bucket using the s3deploy.BucketDeployment class from the AWS CDK library. The BucketDeployment class takes several arguments, including the sources parameter which specifies the source of the data to be deployed (in this case, the "./emr_pipeline/" directory), the destination_bucket parameter which specifies the target S3 bucket (primary_bucket in this case), and the destination_key_prefix parameter which specifies the prefix for the destination key in the bucket ("emr_pipeline/" in this case). The prune parameter is set to False, which means that any files in the destination bucket that are not present in the source will not be deleted.

Overall, the code sets up an AWS CDK stack that deploys data to an S3 bucket and triggers a lambda function.

AWS CDK: EMR Cluster Stack

We need to create an EMR cluster. The following code is useful in scenarios where you want to automate the provisioning and configuration of an EMR cluster as part of an infrastructure-as-code approach.

By using the AWS CDK, you can define your EMR cluster configuration and dependencies using a high-level programming language (in this case, Python) rather than manually configuring the cluster through the AWS Management Console or using the AWS CLI.

The code sets up the necessary resources and configurations for an EMR cluster, including IAM roles, S3 buckets for storing scripts and logs, instance types, subnet, key name, and the steps to be executed in the cluster.

Using the code, you can create, update, and manage the EMR cluster as code, enabling version control, repeatability, and automation of the cluster deployment process. It also provides flexibility to modify and customize the cluster configuration according to your specific requirements.

Overall, the code allows you to define and deploy an EMR cluster with the desired configurations programmatically, streamlining the process of setting up and managing EMR clusters in an automated and scalable manner.

Emr_cluster_stack.py:

```
from aws_cdk import aws_ec2 as ec2, aws_iam as iam, App, Aws, Stack,
aws_emr as emr
from constructs import Construct

MASTER_INSTANCE_TYPE = "m5.xlarge"
CORE_INSTANCE_TYPE = "m5.xlarge"
CORE_INSTANCE_COUNT = 2
MARKET = "ON_DEMAND"
CLUSTER_NAME = "emr-pipeline-cluster"
EMR_RELEASE="emr-6.10.0"
SUBNET_ID ="subnet-0d0f61ccb5bd3ca19"

class EMRClusterStack(Stack):
    def __init__(
        self,
        scope: Construct,
        id: str,
        s3_log_bucket: str,
        s3_script_bucket: str,
        vpc_name: str,
        **kwargs,
    ) -> None:
        super().__init__(scope, id, **kwargs)
```

```
        scripts_location = f"{s3_script_bucket}/emr_pipeline/scripts"

        # enable reading scripts from s3 bucket
        read_scripts_policy = iam.PolicyStatement(
            effect=iam.Effect.ALLOW,
            actions=["s3:GetObject",],
            resources=[f"arn:aws:s3:::{scripts_location}/*"],
        )
        read_scripts_document = iam.PolicyDocument()
        read_scripts_document.add_statements(read_scripts_policy)

        # emr service role
        emr_service_role = iam.Role(
            self,
            "emr_service_role",
            assumed_by=iam.ServicePrincipal("elasticmapreduce.amazonaws.com
"),
            managed_policies=[
                iam.ManagedPolicy.from_aws_managed_policy_name(
                    "service-role/AmazonElasticMapReduceRole"
                )
            ],
            inline_policies={
                "read_scripts_document": read_scripts_document
            },
        )

        # emr job flow role
        emr_job_flow_role = iam.Role(
            self,
            "emr_job_flow_role",
            assumed_by=iam.ServicePrincipal("ec2.amazonaws.com"),
            managed_policies=[
                iam.ManagedPolicy.from_aws_managed_policy_name(
                    "service-role/AmazonElasticMapReduceforEC2Role"
                )
            ],
        )
        # emr job flow profile
        emr_job_flow_profile = iam.CfnInstanceProfile(
            self,
            "emr_job_flow_profile",
            roles=["EMR_EC2_DefaultRole"],
            instance_profile_name="emrJobFlowProfile_",
        )

        # create emr cluster
        emr_cluster = emr.CfnCluster(
```

```
            self,
            "emr_cluster",
            instances=emr.CfnCluster.JobFlowInstancesConfigProperty(
                core_instance_group=emr.CfnCluster.InstanceGroupConfigPrope
rty(
                    instance_count=CORE_INSTANCE_COUNT,
                    instance_type=CORE_INSTANCE_TYPE,
                    market=MARKET
                ),
                ec2_subnet_id=SUBNET_ID,
                ec2_key_name="ProjectProAlexClark",
                keep_job_flow_alive_when_no_steps=True,
                master_instance_group=emr.CfnCluster.InstanceGroupConfigPro
perty(
                    instance_count=1,
                    instance_type=MASTER_INSTANCE_TYPE,
                    market=MARKET
                ),
            ),
            steps=[
                {
                    "actionOnFailure": "CANCEL_AND_WAIT",
                    "hadoopJarStep": {
                        "jar": "command-runner.jar",
                        "args": [
                            "hive",
                            "-f",
                            f"s3://{scripts_location}/create_tables.hql",
                        ],
                    },
                    "name": "create_tables",
                },
                {
                    "actionOnFailure": "CANCEL_AND_WAIT",
                    "hadoopJarStep": {
                        "jar": "command-runner.jar",
                        "args": [
                            "hive",
                            "-f",
                            f"s3://{scripts_location}/transform_data.hql",
                        ],
                    },
                    "name": "transform_data",
                }
            ],
            job_flow_role="EMR_EC2_DefaultRole",
            name=CLUSTER_NAME,
            applications=[emr.CfnCluster.ApplicationProperty(name="Hive")],
            service_role=emr_service_role.role_name,
```

```
            configurations=[],
            log_uri=f"s3://{s3_log_bucket}/{Aws.REGION}/elasticmapreduce/",
            release_label=EMR_RELEASE,
            visible_to_all_users=False,
        )
```

The provided code is written in Python and uses the AWS CDK (Cloud Development Kit) library to deploy data to an S3 bucket, which in turn triggers a lambda function. The code defines a class called BucketDeploymentStack, which inherits from the Stack class provided by the AWS CDK. The BucketDeploymentStack class has an __init__ method that takes several parameters, including the scope, construct_id, primary_bucket_name, and log_bucket_name.

Within the __init__ method, the code creates two S3 bucket resources using the s3.Bucket class from the AWS CDK library. One bucket is assigned to the primary_bucket variable, and the other is assigned to the log_bucket variable. The bucket names are specified using the bucket_name parameter.

The code then deploys the data to the S3 bucket using the s3deploy.BucketDeployment class from the AWS CDK library. The BucketDeployment class takes several arguments, including the sources parameter which specifies the source of the data to be deployed (in this case, the "./emr_pipeline/" directory), the destination_bucket parameter which specifies the target S3 bucket (primary_bucket in this case), and the destination_key_prefix parameter which specifies the prefix for the destination key in the bucket ("emr_pipeline/" in this case). The prune parameter is set to False, which means that any files in the destination bucket that are not present in the source will not be deleted.

Overall, the code sets up an AWS CDK stack that deploys data to an S3 bucket and triggers a lambda function.

AWS CDK: Security Stack

Creating a security stack in the AWS CDK (Cloud Development Kit) serves several purposes and offers a range of benefits for managing security-related resources in your AWS infrastructure.

- **Modularity and Reusability:** By creating a separate security stack, you can encapsulate security-related resources such as VPCs, security groups, and network configurations into a self-contained module. This promotes code reusability and allows you to easily incorporate security components into different parts of your infrastructure.
- **Consistent and Auditable Security:** A security stack enables you to define security configurations consistently across multiple environments or projects. By leveraging infrastructure-as-code principles, you can review and audit the security settings defined in the stack, ensuring compliance with organizational or industry-specific security requirements.
- **Automation and Version Control:** With the CDK, you can define and manage your security stack as code. This enables automation of security provisioning and configuration processes, reducing manual errors and ensuring consistent deployment of security resources across different environments. Additionally, version control allows you to track changes, rollbacks, and collaborate with others on security-related code.
- **Scalability and Flexibility:** The CDK's programming model allows you to dynamically configure security resources based on variables or conditions. You can parameterize the security stack to accommodate different environments or deployment scenarios, such as adjusting the number of subnets or configuring security groups based on specific requirements.
- **Integration with Other Stacks:** By separating security-related resources into a dedicated stack, you can easily integrate it with other stacks that require secure networking, such as application stacks or data processing stacks. This promotes loose coupling between different components of your infrastructure while maintaining a clear separation of concerns.

Security_stack.py code:

```
from aws_cdk import aws_ec2 as ec2, Stack
from constructs import Construct

class SecurityStack(Stack):
    def __init__(
            self,
            scope: Construct,
            id: str,
            vpc_name: str,
            **kwargs,
        ) -> None:
        super().__init__(scope, id, **kwargs)

        # VPC
        self.vpc = ec2.Vpc(
            self,
            vpc_name,
            nat_gateways=0,
```

```
            subnet_configuration=[ec2.SubnetConfiguration(
                name=vpc_name,
                subnet_type=ec2.SubnetType.PUBLIC
                )
            ]
        )
```

The provided code is written in Python and uses the AWS CDK (Cloud Development Kit) library to define and create a VPC (Virtual Private Cloud) with a single public subnet. The code defines a class called SecurityStack that inherits from the Stack class provided by the AWS CDK.

Within the __init__ method of the SecurityStack class, the code creates a VPC using the ec2.Vpc class from the AWS CDK library. The VPC is configured with a specified vpc_name and has no NAT (Network Address Translation) gateways. It also specifies a single subnet configuration using the ec2.SubnetConfiguration class, which sets the subnet_type to be public.

By creating this stack, you can define and deploy a VPC with a single public subnet programmatically. This approach allows you to version control, automate, and manage your network infrastructure using the AWS CDK and a high-level programming language (Python). It provides flexibility to customize the VPC configuration, such as adding additional subnets or modifying the subnet type, according to your specific needs.

AWS CDK: APP.PY

The app.py file is of significant importance in AWS CDK (Cloud Development Kit) as it serves as the entry point for your CDK application. It plays a crucial role in defining and deploying the infrastructure resources in your AWS environment.

Python code:

```
#!/usr/bin/env python3
import os

import aws_cdk as cdk
from aws_cdk import Tags
from decouple import config

from bucket_deployment_stack.bucket_deployment_stack import
BucketDeploymentStack
from emr_cluster_stack.emr_cluster_stack import EMRClusterStack
from security_stack.security_stack import SecurityStack

# Set global variables
ID = config('id')
PRIMARY_BUCKET = f"emr-pipeline-{ID}"
LOG_BUCKET = f"emr-logs-{ID}"
SCRIPT_LOCATION = f"{PRIMARY_BUCKET}/scripts/"
VPC_NAME = f"emr-pipeline-vpc-{ID}"

env_USA = cdk.Environment(account="143176219551", region="us-west-2")
app = cdk.App()

# Create Stacks

security_stack = SecurityStack(
    scope=app,
    id="SecurityStack",
    vpc_name=VPC_NAME,
    env=env_USA
)


bucket_deployment_stack = BucketDeploymentStack(
    scope=app,
    construct_id=f"BucketDeploymentStack",
    primary_bucket=PRIMARY_BUCKET,
    log_bucket=LOG_BUCKET,
    env=env_USA
)
```

```
emr_cluster_stack = EMRClusterStack(
    scope=app,
    id=f"EMRClusterStack",
    s3_log_bucket=LOG_BUCKET,
    s3_script_bucket=PRIMARY_BUCKET,
    vpc_name=f"SecurityStack/{VPC_NAME}",
    env=env_USA
)


# Add dependencies
# emr_cluster_stack.add_dependency(bucket_deployment_stack)
# emr_cluster_stack.add_dependency(security_stack)


# Add tags
Tags.of(app).add("ProjectOwner", "Alex-Clark")
Tags.of(app).add("Project", "EMR-ETL-Pipeline")

app.synth()
```

The provided code is a Python script that defines and deploys various AWS CDK stacks to set up an EMR ETL (Extract, Transform, Load) pipeline.
The script begins with the necessary imports and environment configuration. It defines global variables such as ID, PRIMARY_BUCKET, LOG_BUCKET, SCRIPT_LOCATION, and VPC_NAME, which are used to customize the deployment.
Next, the script creates the CDK application and initializes the AWS environment for the specific region and AWS account.
The code then proceeds to create three CDK stacks: SecurityStack, BucketDeploymentStack, and EMRClusterStack.

- SecurityStack sets up the VPC (Virtual Private Cloud) with a specified name.
- BucketDeploymentStack handles the deployment of data to S3 buckets, which triggers the associated lambda function.
- EMRClusterStack creates an EMR (Elastic MapReduce) cluster for processing data, including the necessary roles, profiles, and configurations.

After defining the stacks, there are commented lines that indicate possible dependencies between the stacks.
Lastly, the code adds tags to the CDK application, specifying project-related metadata. The app.synth() function synthesizes the CDK application into a CloudFormation template that can be deployed to AWS.

In summary, the code sets up an EMR ETL pipeline by deploying the required CDK stacks, configuring security, deploying data to S3, and creating an EMR cluster for data processing.

AWS CDK: Execution

To execute a CDK (Cloud Development Kit) project, you typically follow these steps:

1. **Install CDK:** Ensure that you have CDK installed on your local machine. You can install it globally using a package manager like npm (Node Package Manager) or pip (Python Package Installer). For example, to install CDK using npm, you would run:

   ```
   npm install -g aws-cdk
   ```

2. **Set up your CDK project:** Create a new directory for your CDK project and navigate into it. Initialize your project using the CDK CLI command **`cdk init`** followed by the desired programming language. For example, to initialize a CDK project using Python, you would run:

   ```
   cdk init app --language python
   ```

   This command sets up the necessary project structure and creates an initial **`app.py`** file.

3. **Define your CDK stacks:** Open the **`app.py`** file and define the stacks that make up your infrastructure. Use the CDK constructs specific to your chosen programming language to define AWS resources, their properties, and any desired configurations.

4. **Install dependencies:** If your CDK project has any external dependencies, install them using the package manager specific to your programming language. For example, if you're using Python and have dependencies listed in a **`requirements.txt`** file, you can install them using **`pip install -r requirements.txt`**.

5. **Bootstrap AWS environment:** If you are deploying your CDK project to an AWS account for the first time or using a new AWS region, you need to bootstrap your environment. This is a one-time setup process that ensures the necessary resources are created in your AWS account to support CDK deployments. Run the command **`cdk bootstrap`** to perform the bootstrapping process.

6. **Deploy the CDK stacks:** Once your CDK project is set up and the AWS environment is bootstrapped, you can deploy your stacks. Use the CDK CLI command **`cdk deploy`** followed by the stack name or the **`–all`** flag to deploy all stacks defined in your **`app.py`** file. For example:

   ```
   cdk deploy --all
   ```

   The CDK CLI will package and deploy your infrastructure resources according to the definitions in your stacks.

7. **Review and confirm changes:** The CDK CLI will display a summary of the changes that will be made to your AWS environment. Review the changes and enter "y" to confirm and proceed with the deployment.

8. **Monitor the deployment:** The CDK CLI will provide progress updates as it creates or updates resources. You can monitor the deployment status and any errors or warnings that may arise during the process.

Once the deployment is complete, your CDK infrastructure will be provisioned and ready to use. You can make changes to your stacks in the **`app.py`** file, run the **`cdk deploy`** command again to update your resources, and continue iterating on your infrastructure as needed.

Setting up Virtual Machine in Azure for Power BI (Mac users)

To enable Mac users to utilize Power BI, we will establish a virtual machine in Azure dedicated to running Power BI. This approach circumvents the limitation of directly running Power BI on Mac systems. By setting up the virtual machine in Azure, Mac users can access and utilize Power BI through the virtual environment, providing them with the full functionality of Power BI on their Mac computers.

To set up a Windows virtual machine (VM) in Azure, you can follow these steps:

1. Sign in to the Azure portal (https://portal.azure.com) using your Azure account.
2. Click on "Create a resource" or the "+" button on the left-hand side of the portal.
3. In the search bar, type "Virtual Machine" and select "Virtual Machine" from the search results.
4. Click on the "Create" button to start the virtual machine creation process.
5. Provide the necessary details for the VM, such as the subscription, resource group, and virtual machine name.
6. Choose the appropriate region where you want to deploy the VM.
7. Select the desired operating system image for the VM, such as Windows, Linux, or a specific distribution.
8. Choose the VM size based on your requirements, such as the number of CPU cores, memory, and disk space.
9. Configure the networking settings, including virtual network, subnet, and public IP address.
10. Set up authentication and access options, such as username and password or SSH key.
11. Configure additional options, such as disk types, availability options, monitoring, and tags.
12. Review the summary of your configuration and click on the "Create" button to create the virtual machine.
13. Azure will start provisioning the VM, which may take a few minutes.
14. Once the deployment is complete, you can access and manage your virtual machine from the Azure portal.

It's important to note that setting up a virtual machine in Azure may involve additional configuration steps depending on your specific requirements, such as configuring network security groups, attaching storage disks, setting up load balancing, or configuring other advanced features.

To install Power BI in a virtual machine in Azure, you can follow these steps:

1. Sign in to the Azure portal (portal.azure.com) using your Azure account.
2. Connect to the virtual machine using Remote Desktop Protocol (RDP) or any other remote access method provided by Azure.
3. Once connected to the virtual machine, open a web browser and navigate to the Microsoft Power BI download page (powerbi.microsoft.com).
4. Download the Power BI Desktop installer suitable for your virtual machine's operating system. Choose the version (32-bit or 64-bit) based on your VM's architecture.
5. Locate the downloaded installer file on the virtual machine and run it.
6. Follow the on-screen prompts to complete the installation process. Review and accept the license terms, choose the installation location if prompted, and select any additional installation options as needed.

7. Once the installation is complete, launch Power BI Desktop from the Start menu or desktop shortcut.
8. Sign in to Power BI Desktop using your Power BI account credentials or the appropriate authentication method.
9. Power BI Desktop is now installed and ready to use in the virtual machine. You can start creating reports, visualizations, and dashboards using Power BI's features and functionalities.

Remember to ensure that your virtual machine has sufficient resources and storage capacity to handle Power BI workloads effectively. Additionally, consider any licensing requirements for Power BI usage in your Azure environment.

Data Visualizations in Power BI

To connect Power BI to data stored in Hive within an EMR cluster, you can follow these steps:

1. Launch Power BI Desktop on your local machine.
2. Click on "Get Data" in the Home tab of the Power BI Desktop ribbon.
3. In the "Get Data" window, search for "Hive" and select "Hive (Beta)" from the available data connectors.
4. In the "Hive" window, provide the necessary connection details:
    - Server: Enter the hostname or IP address of the EMR cluster's master node where Hive is running.
    - Port: Specify the port number on which HiveServer2 is listening. By default, it is 10000.
    - Database: Specify the name of the Hive database containing the data you want to access.
    - Authentication: Choose the appropriate authentication method based on your EMR cluster setup.
    - Username and Password: Provide the credentials for authentication if required.
5. Click on "Connect" to establish the connection to the Hive data.
6. Power BI will retrieve the metadata from Hive and display a navigator window showing the available tables and views in the selected database.
7. Select the desired table or view that you want to import into Power BI and click on "Load" or "Transform Data" to start importing the data.
8. Power BI will load the selected data into the Power Query Editor, where you can perform any necessary transformations or data shaping operations.
9. Once you are satisfied with the data transformations, click on "Close & Apply" to load the data into Power BI.
10. You can now start building visualizations, reports, and dashboards in Power BI using the data sourced from Hive in the EMR cluster.

Note: Make sure that you have the necessary network connectivity and permissions to access the EMR cluster and Hive from your local machine running Power BI Desktop.

Summary

- In this training, we built a serverless pipeline using AWS CDK.
- Understood the importance of AWS CDK in today's world.
- The reusability of the proposed pipeline in the various industries was discussed.
- We understood each service used in this project in detail, their advantages, disadvantages and use cases.
- Various AWS services were leveraged to implement the concepts with the most cost-optimized configurations that can even serve 100x the amount of demo data used.
- The Sales dataset was understood, and data analysis and transformation were performed using tools like Hive and Spark in an EMR cluster.
- AWS CDK was installed, and its various commands were learned for infrastructure as code. Different CDK stacks, such as the S3 Bucket deployment stack and the Security stack, were deployed.
- The advantages of Serverless technologies were explored, offering scalability and cost-effectiveness for solutions.
- The differences between AWS and Azure cloud platforms were understood and compared.
- A Virtual Machine was set up in Azure, and connections to Hive tables were established using Power BI for data visualization and analysis.