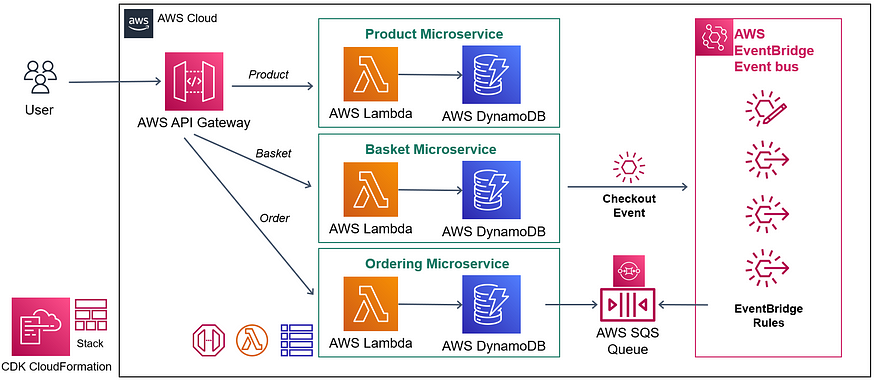
**Using AWS Lambda as a Microservice**

****

AWS Serverless Event-driven E-commerce Microservices Architecture

****Reference Architecture**** above which is a ****real-world**** ****Serverless E-commerce application**** and it includes;

* ****REST API**** and ****CRUD****endpoints with using ****AWS Lambda, API Gateway****
* ****Data persistence**** with using ****AWS DynamoDB****
* ****Decouple microservices****with ****events****using ****AWS EventBridge****
* ****Message Queues**** for cross-service communication using****AWS SQS****
* ****Cloud stack development**** with ****IaC****using ****AWS CloudFormation CDK****

1. ****Product Microservices****is basically implementing ****RESTful Microservices**** with using ****AWS Lambda, API Gateway****and ****DynamoDB****for performing ****CRUD****operations.
2. ****Basket Microservices****is implementing****RESTful Microservices****like Product microservices. But also Basket Microservices ****creates****and ****trigger Checkout Basket event****to the ****Amazon EventBridge****for ****Event-Driven asynchronous**** Communication between Microservices in order to ****Decoupling Basket****and ****Ordering Microservices****.
3. After that ****Amazon EventBridge publish Checkout event****to the ****AWS SQS**** and this ****SQS queue****messages receive from Ordering microservices in order to ****processing events asynchronously****using queues.
4. ****Ordering Microservice**** also expose ****RESTful APIs****for querying Order tables.

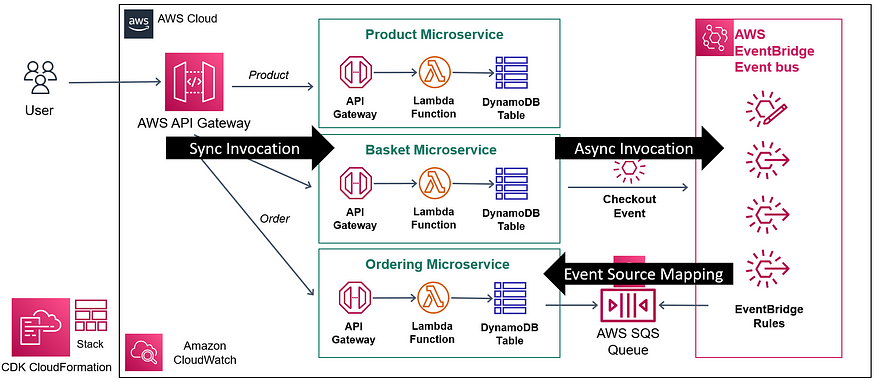
And all these****Serverless E-commerce**** application ****infrastructure****developed with using ****AWS CDK.**** We will use ****AWS Lambda****service as a ****microservices****and use power of lambda functions. And this microservices ****databases****will store No SQLAWS ****DynamoDB****databases. And also we have followed ****communication patterns****For ****synchronous****and ****asynchronous****communications with using ****AWS API Gateway, AWS Event Bridge****as a Event busand ****AWS SQS**** as a ****queue****service.

* ****AWS API Gateway**** -for-****Restful API-Driven Development****and Synchronous Event Sources
* ****AWS EventBridge**** -for- ****Event-Driven Asynchronous**** ****Communication****between Microservices
* ****AWS SQS**** -for- ****Decouple Microservices****and processing events asynchronously using queues

That means we design our architecture rely on ****asynchronous connections**** that gives us much more ****flexible****and ****resilient****application.

| **Term** | **Description** |
| --- | --- |
| ***Product Microservices*** | A Microservice responsible for implementing Restful APIs using AWS Lambda, API Gateway, and DynamoDB to perform CRUD operations related to products in the e-commerce application. |
| ***Basket Microservices*** | Another Microservice with Restful APIs, similar to Product Microservices, but it also triggers a Checkout Basket event to Amazon Event Bridge for Event-Driven asynchronous communication between microservices. It aims to decouple Basket and Ordering Microservices. |
| ***Ordering Microservices*** | A Microservice that processes the events triggered by Basket Microservices. It exposes Restful APIs for querying the Order tables. |
| ***AWS Lambda*** | A serverless compute service provided by AWS that allows you to run code without managing servers. It is used to implement the microservices in the architecture. |
| ***API Gateway*** | In our microservices architecture, the entry point for client applications is the API Gateway. It serves as a bridge between clients and our microservices, offering a Restful API-Driven Development approach, allowing clients to make requests and receive responses in a synchronous manner. When a client sends a request to the API Gateway, it acts as a synchronous event source, meaning it expects an immediate response from the microservices. The API Gateway then acts as a serverless API proxy, directing the requests to the appropriate internal microservices responsible for handling CRUD operations. This setup enables us to manage the communication between clients and microservices efficiently while providing a seamless experience for request and response interactions. |
| ***DynamoDB*** | A fully managed NoSQL database service provided by AWS. It is used to store the databases for the Product and Basket Microservices. |
| ***Amazon Event Bridge*** | A serverless event bus service provided by AWS. It allows you to connect application data from different sources and send it to targets such as AWS Lambda, SQS, etc. It is used for Event-Driven Asynchronous Communication between Microservices in the architecture. |
| ***AWS SQS and Ordering Lambda Microservices*** | After the Checkout event is published to the Event Bridge, the Ordering Microservice comes into play. The Event Bridge sends this event to an AWS SQS queue, leveraging the benefits of SQS's queuing capability. The Ordering Lambda Microservice then consumes this event using a polling mechanism. In this event source mapping communication type, the Ordering Lambda sends periodic requests to the AWS SQS queue to check for new events. When an event is available, the Lambda Microservice retrieves it from the queue.Once the event is consumed from the AWS Queue, the Ordering Lambda Microservice processes it by creating an order record in its DynamoDB table. All these operations are performed using Lambda functions developed with the help of the AWS SDK. The use of AWS SQS enables a decoupled and asynchronous flow, allowing the Ordering Microservice to efficiently handle events and manage the order creation process in the e-commerce application. |
| ***AWS SQS*** | Amazon Simple Queue Service (SQS) is a fully managed message queuing service. It enables decoupling of microservices and helps in processing events asynchronously using queues. It is used to handle the Checkout events published by Amazon Event Bridge in this architecture. |
| ***AWS CDK*** | AWS Cloud Development Kit (CDK) is an open-source software development framework to define cloud infrastructure in code and provision it through AWS Cloud Formation. It is used to develop the serverless e-commerce application infrastructure in this architecture. |

# **Microservices Communications with AWS Lambda Invocation Types:**



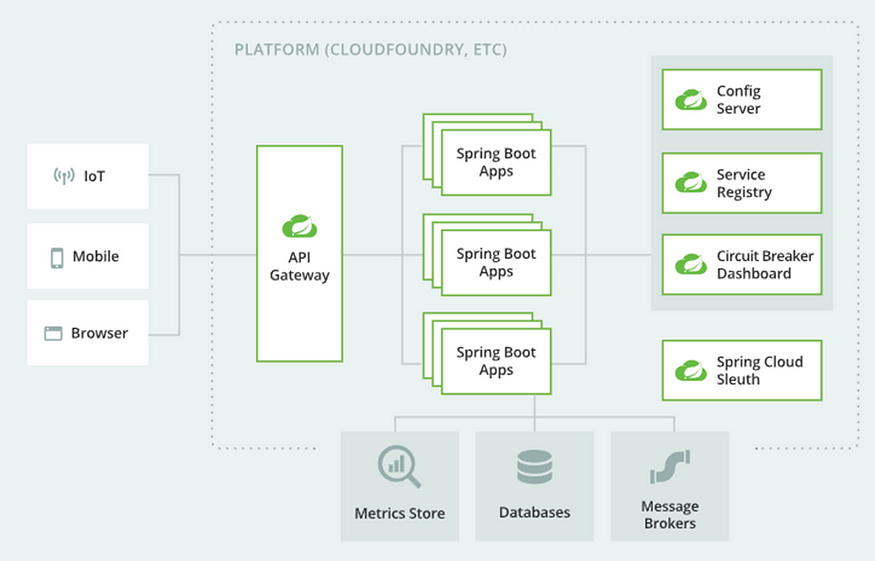
| **Term** | **Explanation** |
| --- | --- |
| Synchronous Communication with AWS API Gateway | AWS API Gateway is a service that acts as a bridge between client applications and microservices. It receives incoming requests from clients and forwards them to the appropriate Microservice for processing. The client waits for a response from the Microservice before proceeding. This is known as synchronous communication. |
| Asynchronous Communication with AWS Event Bridge | AWS Event Bridge (formerly known as the Event Bus) is a service that enables communication between different parts of an application using events. Instead of direct request-response interactions, events are published to the event bus, and other services can listen for and respond to these events asynchronously. This is called asynchronous communication. |
| Event Source Mapping Communication with AWS SQS | AWS Lambda can be configured to listen to events from different sources. In this case, we are using Event Source Mapping to poll and receive messages from AWS SQS (Simple Queue Service). When a message is available in the SQS queue, Lambda processes it asynchronously. This approach helps decouple microservices and allows processing events without a direct, real-time connection. |

# **AWS is a Cloud Provider: AWS**is a**cloud provider** that provide infrastructure to our applications, it is a platform service that provide **virtualization to** our **servers**, **infrastructures**and **databases**requirements. It is not only an infrastructure provider, but also an **Application Development Framework.**

# **What is Application Development Framework ?**

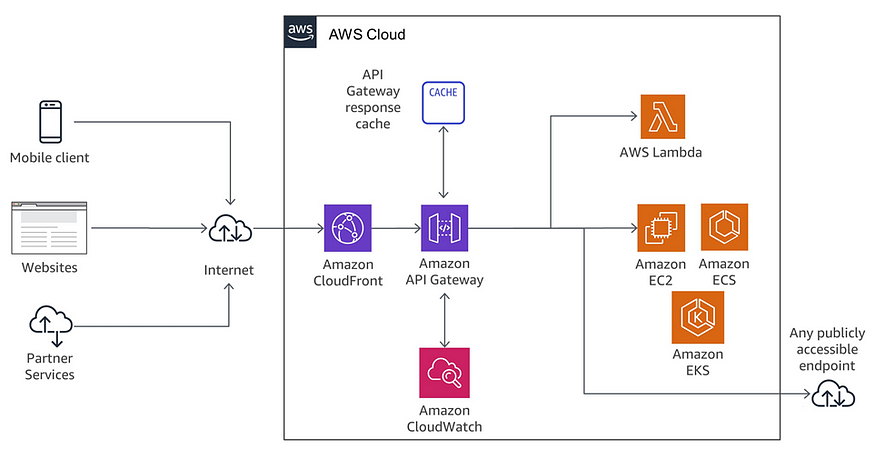
Application Development Framework is a ****set of code**** that is responsible for calling your actual business logic based on its defined architecture. It provides the following features :

* To Create Restful Apis With Underlying Web Application
* To Connect Databases With Orm Tools
* To Provide Centralized Logging Features
* To Provide Identity Management
* To Collect And Visualize Monitoring And Metrics
* To Facilitate To Connect 3rd Party Services Like Kafka, Redis, Rabbitmq So On..
* It provides all those features in order to focus on actual business logic.
* Examples are **Java Spring Boot or .Net ecosystem**



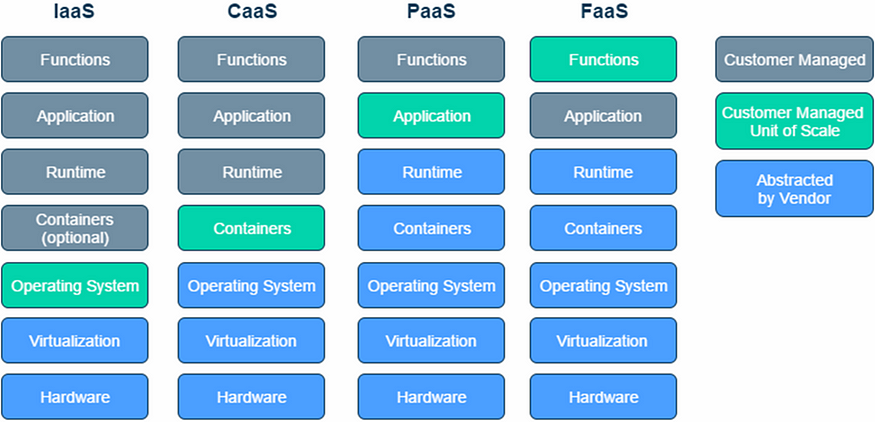
**Figure 1 :** Spring Boot Microservice application (Spring Boot as an Application Development Framework) and it provides to handle all these ****logging****, ****monitoring****, ****configurations****, ****connections****and so on.

# **AWS as an Application Development Framework: AWS already provides these features by means of AWS Serverless Services without thinking , scalability, availability issues and no configuration code requirement, these applications have built-in availability and fault tolerance. There is no need extra architecture for high availability and high scalability designs, these features comes from the beginning. Serverless applications are  important because exceptions can happen anytime, so reliable and resilience architecture are crucial when you running business so Serverless provide this from the beginning. Think to design multi-cluster applications from scratch, it will really costly and unpredictable results. So Serverless handle all automatically.**



**Serverless applications have built-in service integration, more than 200 services are easy to integrate each with each other to provide value of business and automatic scaling from zero to peak demands, you can adapt to customer needs faster than ever. So you can focus on building your application instead of configuring it.**

# **Evolution of Cloud Infrastructures:**



## **Infrastructure as a Service (IaaS): Renting a computer through a cloud provider. Here,I.e. that all control, including the operating system level, is yours. Example services: Amazon EC2, Azure VMs.**

# **Container as a Service (CaaS):** C**loud provider puts another level of abstraction here so that you can easily manage your containers. If you use Amazon EC2 Container Service (ECS) as an example, Amazon will be managing your docker containers for you. Example services: Docker Cloud, Amazon ECS.**

## **Platform as a Service (PaaS):** We can call PaaS services to managed services. What you will be interested in PaaS services is to upload your program after choosing the language/framework in which your program will run. Here, the service provider takes care of many operational tasks for you, but you still run, for example, the Spring Boot application. You have to make and manage many configurations required for your application. Even if the service will automatically open a new server when there is a sudden load (again, some configuration will be required), it will take time for your application to stand up. In fact, we can say that the way of our application works is the biggest change between PaaS and FaaS. While PaaS applications are working as we are used to that is, the application runs and continues to do its job until a new version comes, FaaS applications run as a result of an incoming event and our application closes after the function finishes its work, for example, it can be an HTTP request, it can be called periodically or if we think for AWS, it can work with events from many other services of AWS (S3, Kinesis, DynamoDB Streams…). Example PaaS services: Heroku, Amazon Elastic Beanstalk.

## **Functions as a Service (Faas) — Serverless:** If you run your application on a Serverless service, you have to switch to an event driven architecture, switching to a Serverless architecture makes many issues such as scaling of the application, deployment, operating system or programming language updates not entirely our problem after you write and upload your code, or make it much easier.