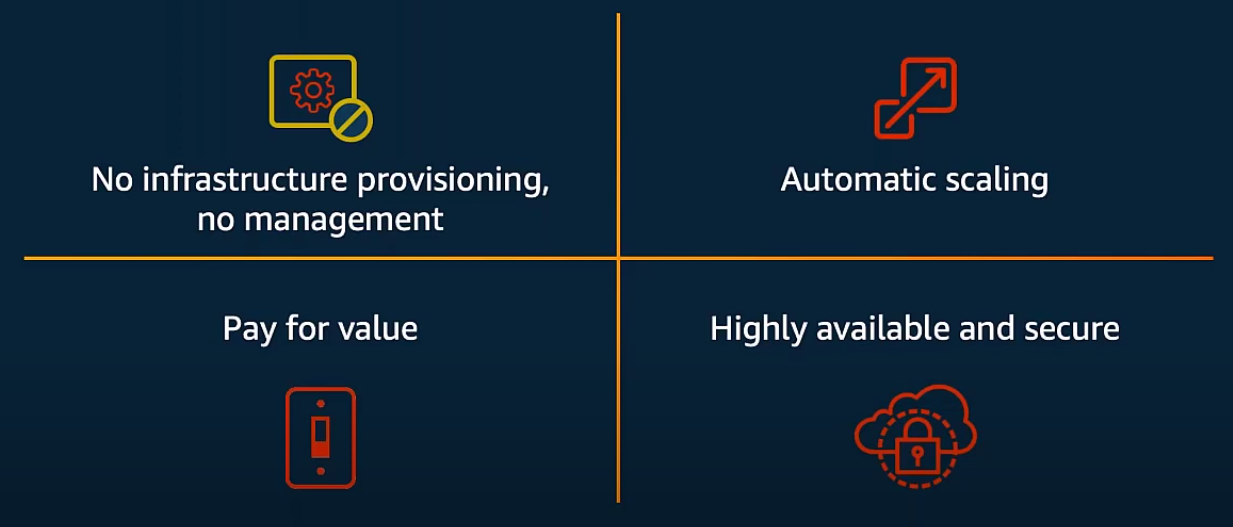
#####-----**AWS Lambda**-----####



1. Lambda handles:

- Load Balancing

- Auto Scaling

- Handling Failures

- Security Isolation

- OS Management

- Managing Utilization: Monitoing and Billing

2. Serverless COncepts:

- To build serverless application around AWS lambda

- A basic unit of compute with aws lambda is Function

- Lambda Function can be written in one of 6 provider runtimes

- Node.js

- Python

- Go

- Ruby

- java

- C#

- Runtime API - Own runtime

- A basic model is "Event Source"

- An event source happens that invoke the function, that event souce can be change in data state

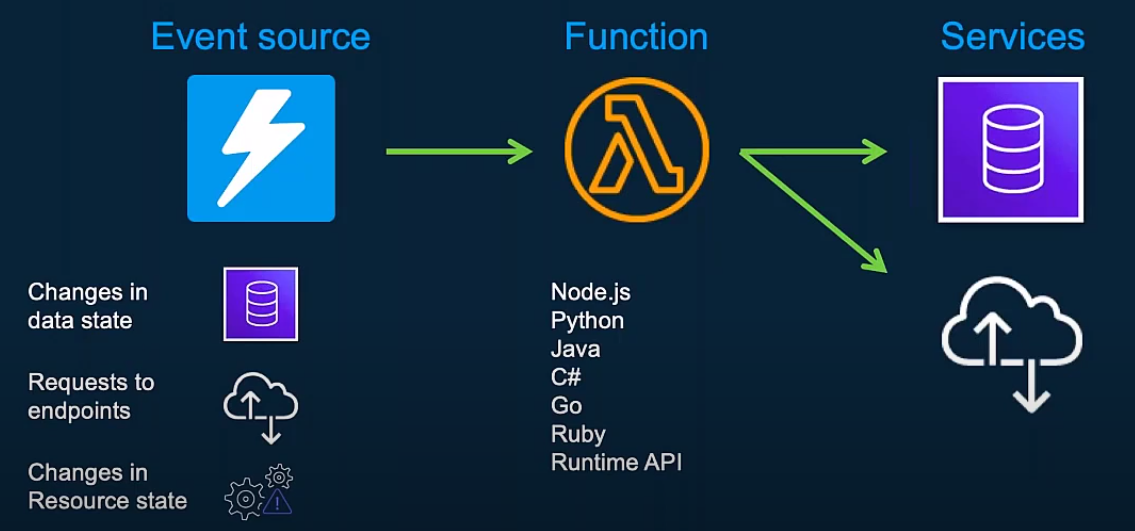
- Ex: change a table in DB or an object upload into S3 bucket.

- Request to endpoint on API gateway to trigger your lambda function.

- Change in a resource state

- AWS config allow to provide a rows and change across resource managed by the rows and capture the changes by the lambda function

- Once your lambda function is been invoked, your lambda function can be interact with the other services. It can be down stream services (S3, DynamoDB, etc)



**Anatomy of Lambda Function:**

**1. Handler Function**: Function to be executed when the lambda function invoked.

* A Lambda function has a few requirements. The first requirement you need to satisfy is to provide a **handler**. The handler is the entry point for the Lambda. A Lambda function accepts JSON-formatted input and will usually return the same.

**2. Event Object**: Data sent during the lambda function invocation

* Data sent from event source to your lambda function
* The final requirement is a **trigger**. You can configure a Lambda invocation in response to an event, such as a new file uploaded to S3, a change in a DynamoDB table, or a similar AWS event. You can also configure the Lambda to respond to requests to AWS API Gateway, or based on a timer triggered by AWS Cloudwatch.
* Example: API Gateway request that should contain information sent the http

**3. Context Object**: Methods available to interact with runtime information.

* The second requirement is that you'll need to specify the **runtime environment** for the Lambda. The runtime will usually correlate directly with the language you selected to write your function.
* This is metadata about to the lambda function.

from \_\_future\_\_ import division

def lambda\_handler(event, context):

number1 = event['Number1']

number2 = event['Number2']

sum = number1 + number2

product = number1 \* number2

difference = abs(number1 - number2)

quotient = number1 / number2

return {

"Number1": number1,

"Number2": number2,

"Sum": sum,

"Product": product,

"Difference": difference,

"Quotient": quotient

}

You can see that we've defined so called *handler*, the function that will be executed by Lambda Service, every time the event occurs. Handler takes two arguments: an *Event object,* and a *Context*.

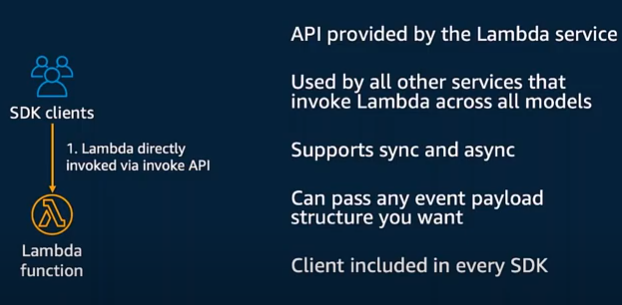
Every Lambda function receives two arguments: *Event*and*Context*. While **Event** provides function with the detailed information about the event that has triggered the execution (for example, Event from API Gateway can be used to retrieve request details such as query parameters, header or even request body).

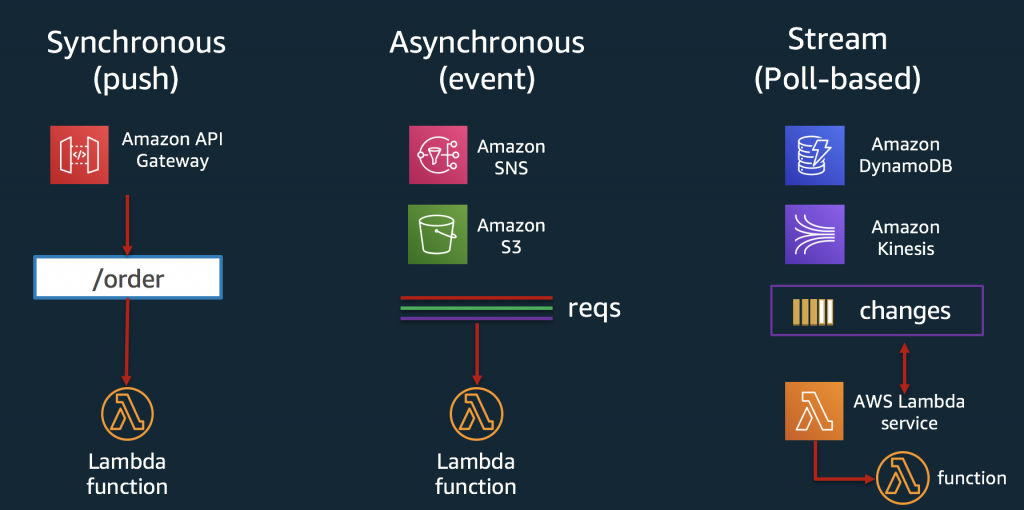
**Context** on the other hand, contains methods and properties that provide information about the invocation, function, and execution environment (such as assigned memory limit or upcoming execution timeout).

**Execution Models:**

There are 4-key way lambda function has invoke.

**1. Lambda directly invoked via invoke API**





**2. Synchronous Model:**

* It is Push based model.
* An Amazon API Gatway, the user request from data from an end point.
* Synchronous invocations are the most straight forward way to invoke your Lambda functions. In this model, your functions execute immediately when you perform the Lambda Invoke API call. This can be accomplished through a variety of options, including using the CLI or any of the supported SDKs.
* Here is an example of a synchronous invoke using the CLI:

aws lambda invoke —function-name MyLambdaFunction —invocation-type RequestResponse —payload “[JSON string here]”

* The Invocation-type flag specifies a value of “RequestResponse”. This instructs AWS to execute your Lambda function and wait for the function to complete. When you perform a synchronous invoke, you are responsible for checking the response and determining if there was an error and if you should retry the invoke.
* Many AWS services can emit events that trigger Lambda functions. Here is a list of services that invoke Lambda functions synchronously:
  + Elastic Load Balancing (Application Load Balancer)
  + Amazon Cognito
  + Amazon Lex
  + Amazon Alexa
  + Amazon API Gateway
  + Amazon CloudFront (Lambda@Edge)
  + Amazon Kinesis Data Firehose

**Asynchronous Invokes**

Here is an example of an asynchronous invoke using the CLI:

aws lambda invoke —function-name MyLambdaFunction —invocation-type Event —payload “[JSON string here]”

Notice, the Invocation-type flag specifies “Event.” If your function returns an error, AWS will automatically retry the invoke twice, for a total of three invocations.

Here is a list of services that invoke Lambda functions asynchronously:

* [Amazon Simple Storage Service](https://docs.aws.amazon.com/lambda/latest/dg/with-s3.html)
* [Amazon Simple Notification Service](https://docs.aws.amazon.com/lambda/latest/dg/with-sns.html)
* [Amazon Simple Email Service](https://docs.aws.amazon.com/lambda/latest/dg/services-ses.html)
* [AWS CloudFormation](https://docs.aws.amazon.com/lambda/latest/dg/services-cloudformation.html)
* [Amazon CloudWatch Logs](https://docs.aws.amazon.com/lambda/latest/dg/services-cloudwatchlogs.html)
* [Amazon CloudWatch Events](https://docs.aws.amazon.com/lambda/latest/dg/with-scheduled-events.html)
* [AWS CodeCommit](https://docs.aws.amazon.com/lambda/latest/dg/services-codecommit.html)
* [AWS Config](https://docs.aws.amazon.com/lambda/latest/dg/services-config.html)

Asynchronous invokes place your invoke request in Lambda service queue and we process the requests as they arrive. You should use [AWS X-Ray](https://docs.aws.amazon.com/lambda/latest/dg/lambda-x-ray.html) to review how long your request spent in the service queue by checking the “dwell time” segment.

**Poll-Based Invokes**

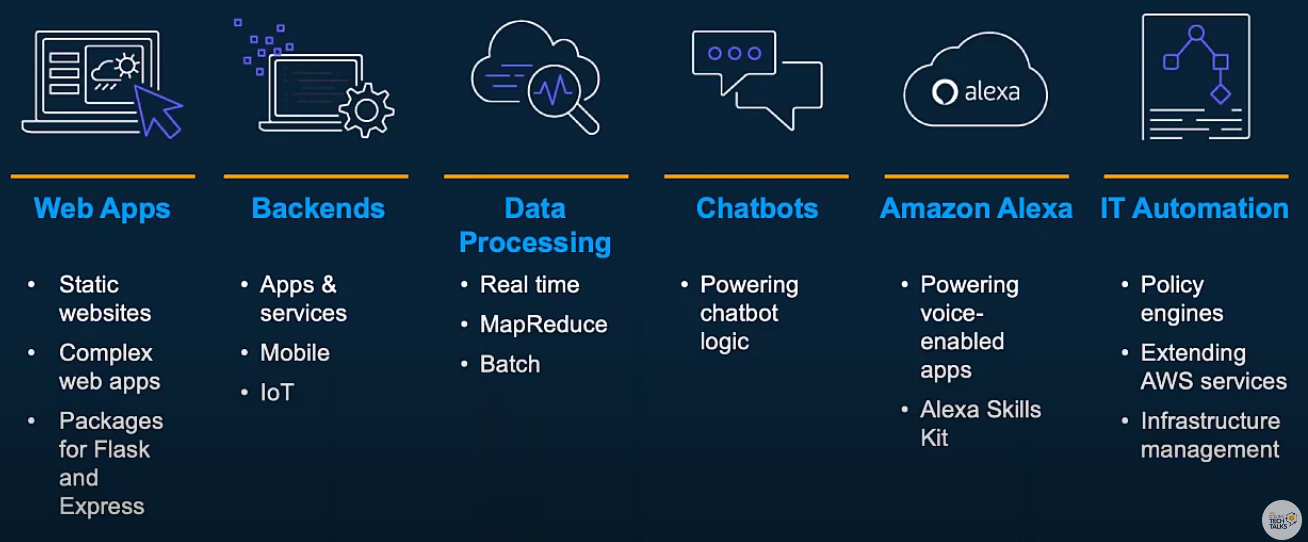
This invocation model is designed to allow you to integrate with AWS Stream and Queue based services with no code or server management. Lambda will poll the following services on your behalf, retrieve records, and invoke your functions. The following are supported services:

* [Amazon Kinesis](https://docs.aws.amazon.com/lambda/latest/dg/with-kinesis.html)
* [Amazon SQS](https://docs.aws.amazon.com/lambda/latest/dg/with-sqs.html)
* [Amazon DynamoDB Streams](https://docs.aws.amazon.com/lambda/latest/dg/with-ddb.html)

AWS will manage the poller on your behalf and perform Synchronous invokes of your function with this type of integration. The retry behavior for this model is based on data expiration in the data source. For example, Kinesis Data streams store records for 24 hours by default (up to 168 hours). The specific details of each integration are linked above.

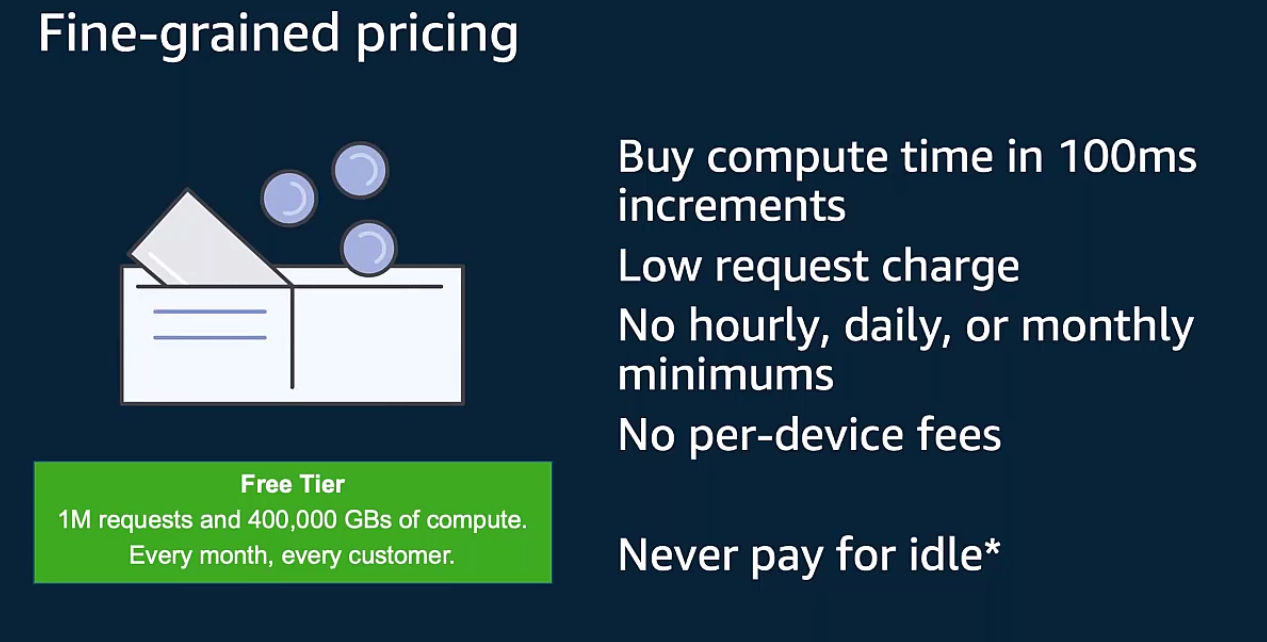
**Common AWS Lambda use cases:**

* Example Mobile backend, All mobile backend are traditional client application, these can be used again API Gateway connected to AWS lambda function.
* Policy Engine: can invoke AWS lambda function when every see violation or simply need to test resource



**Prices and Resource Allocation**

* **Fine-Grained Pricing**



**Permissions:**

* **Function policies**
* **Execution Role**

