AWS Lambda

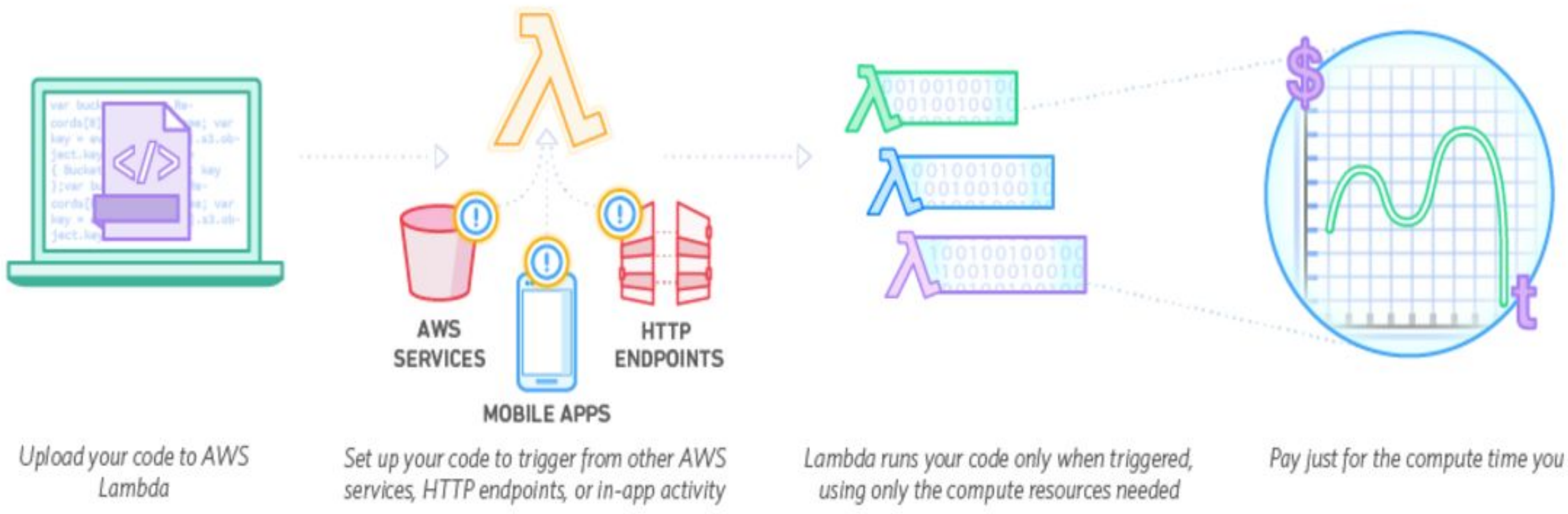
Feb 2018

What is AWS Lambda?

AWS compute service that lets you run code without provisioning or managing servers

- Introduced in 2014
- Event-driven serverless computing
- Microservices without server
- Passive
- Stateless
- Deployment & Monitoring is managed by AWS Lambda

How does it work ?



Upload your code to AWS Lambda

Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity

Lambda runs your code only when triggered, using only the compute resources needed

Pay just for the compute time you use

Development Lifecycle

1. Author the code
2. Create and Upload the deployment package to Lambda
   a. Package code as a ‘deployment package’ in specific way
      i. including libraries used in code
   b. Provide deployment configuration
      i. CPU /Memory
      ii. Runtime environment
      iii. Handler information (index.handler) `Filename.handler`
   c. Test the lambda function
      i. using Console , CLI , SAM

3. Monitor & Troubleshooting
   a. Metrics reported through cloudwatch

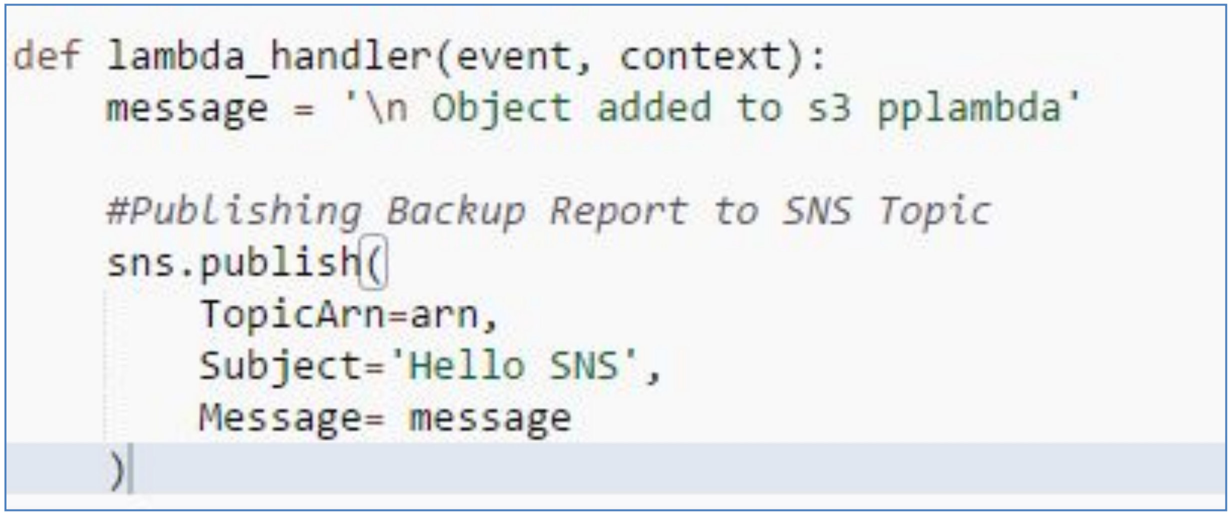
Programming Model

Programming Key Concepts

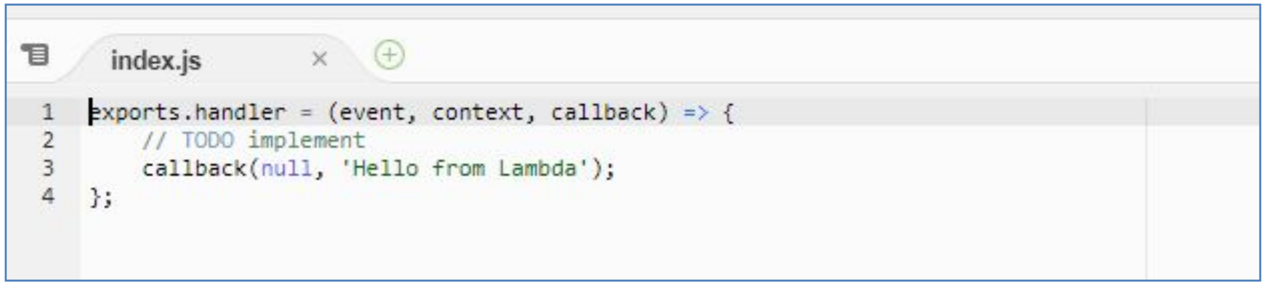
- Follows common pattern irrespective of programming language
- Must write code in stateless manner
- No affinity to underlying resources
- Persist data to S3 , Dynamodb , RDS etc.
- 4 Core Concepts
  - Handler
  - Context
  - Logging
  - Exceptions

1. Handler

- Function AWS Lambda calls to start execution of your Lambda function
- AWS Lambda passes any **event** data to this handler as the first parameter
- Can invoke other functions



```python
def lambda_handler(event, context):
    message = '\n Object added to s3 pplambda'

    #Publishing Backup Report to SNS Topic
    sns.publish(
        TopicArn=arn,
        Subject='Hello SNS',
        Message= message
    )
```



```javascript
exports.handler = (event, context, callback) => {
    // TODO implement
    callback(null, 'Hello from Lambda');
};
```

http://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-handler.html

2. Context Object

- Passed as a 2nd parameter
- Interacts with AWS Lambda to get information such as
    - How much time is remaining before AWS Lambda terminates your Lambda function
    - CloudWatch log group and log stream associated with the Lambda function that is executing
    - The AWS request ID returned to the client that invoked the Lambda function. You can use the request ID for any follow up inquiry with AWS support
    - If the Lambda function is invoked through AWS Mobile SDK, you can learn more about the mobile application calling the Lambda function

3. Logging

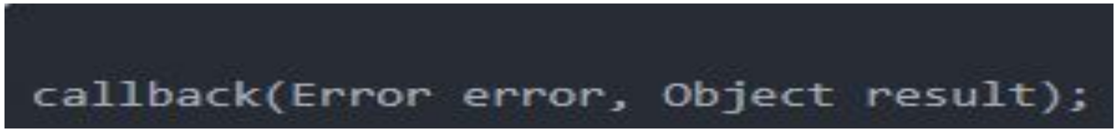
- Writes logs to cloudwatch logs
- Custom Logs
- Metrics
    - Invocations
    - Duration
    - Throttles

4. Exceptions

- Communicate the result of execution to AWS Lambda
- If Lambda function notifies AWS Lambda that it failed to execute properly, Lambda will attempt to convert the error object to a String
- Synchronous Exceptions
  - If a client specifies the RequestResponse invocation type (that is, synchronous execution), it returns the result to the client that made the invoke call.
- Asynchronous Exceptions
  - Does not return results to caller
  - logs the errors to CloudWatch
  - If a client specifies the Event invocation type (that is, asynchronous execution), AWS Lambda will not return anything. Instead, it logs the error information to CloudWatch Logs.
  - You can also see the error metrics in CloudWatch Metrics.

Programming Model with Node.js

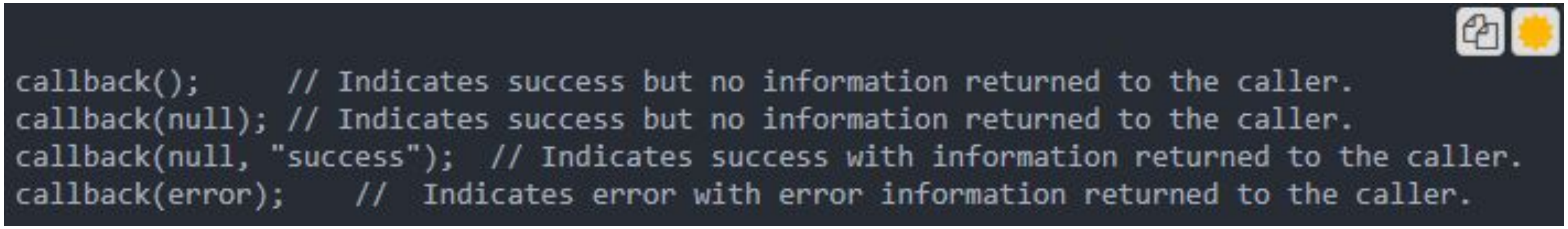
Using Callback



```
callback(Error error, Object result);
```

- error
  a. optional parameter
  b. use to provide results of the failed Lambda function execution
  c. When a Lambda function succeeds, pass null as the first parameter.
- result
  a. an optional parameter
  b. must be JSON.stringify compatible.
  c. If an error is provided, this parameter is ignored.

Callback Examples

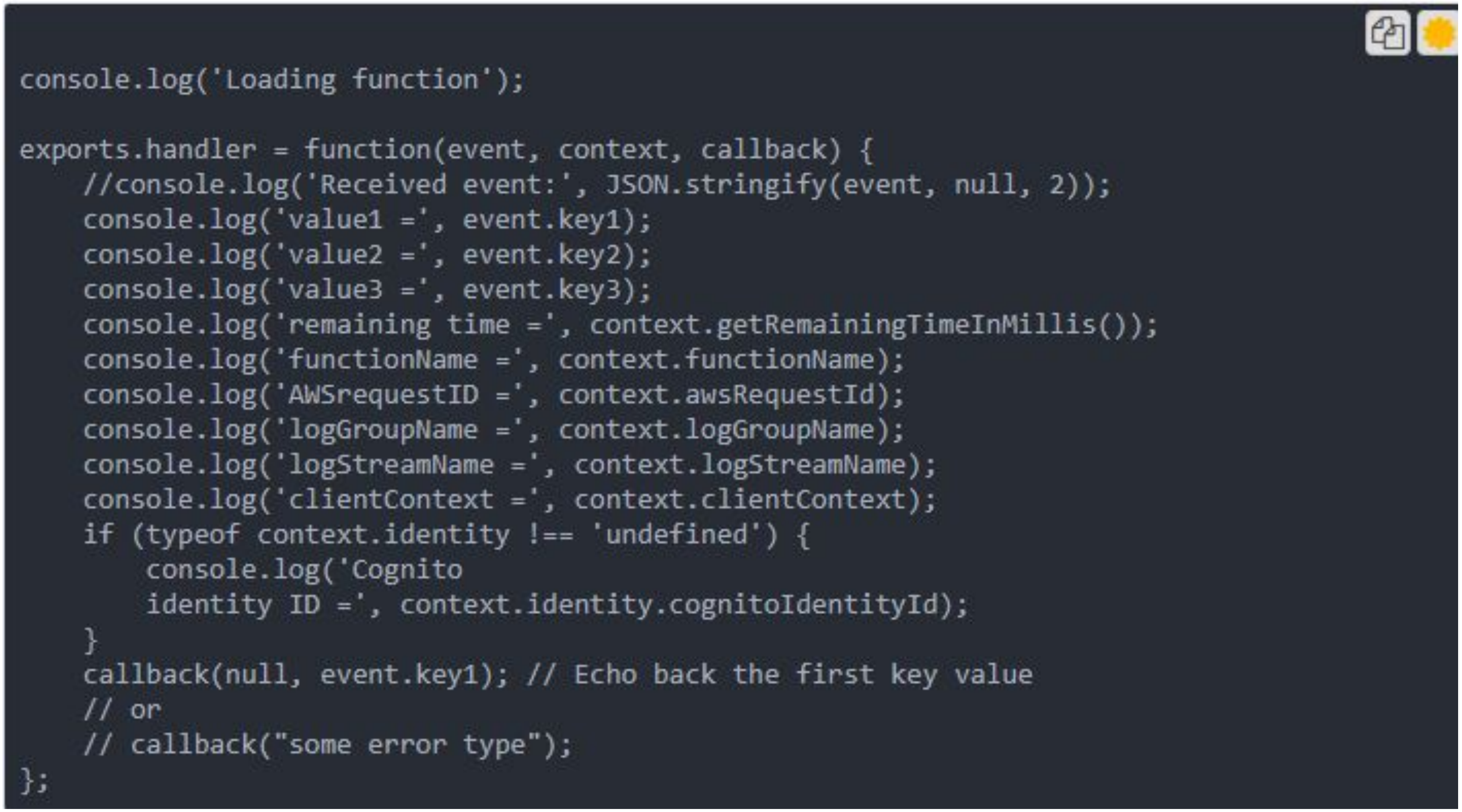


```
callback();     // Indicates success but no information returned to the caller.
callback(null); // Indicates success but no information returned to the caller.
callback(null, "success");  // Indicates success with information returned to the caller.
callback(error);    //  Indicates error with error information returned to the caller.
```

If code does not have callback statement then Lambda adds it as

```
callback(null)
```

Using Context



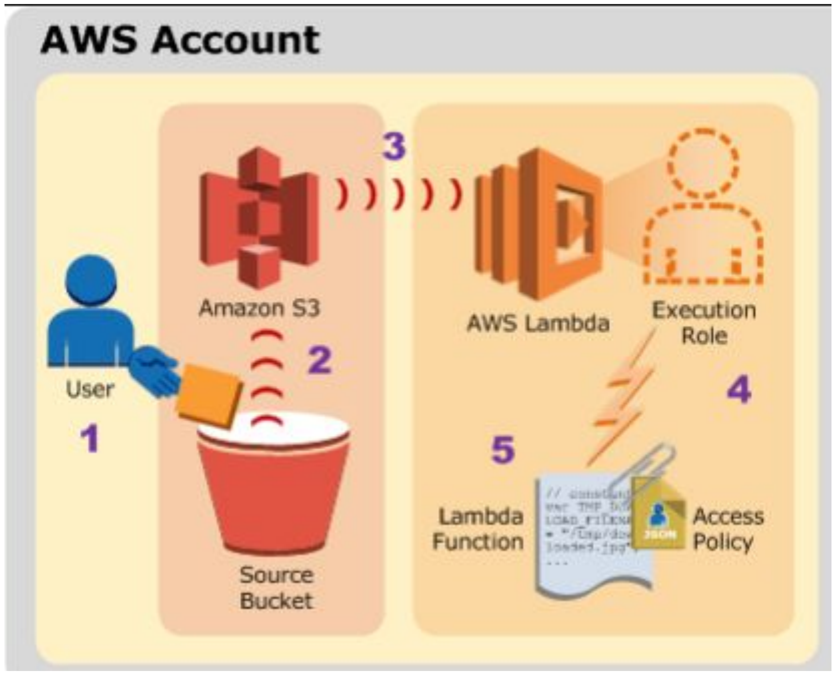
```
console.log('Loading function');

exports.handler = function(event, context, callback) {
    //console.log('Received event:', JSON.stringify(event, null, 2));
    console.log('value1 =', event.key1);
    console.log('value2 =', event.key2);
    console.log('value3 =', event.key3);
    console.log('remaining time =', context.getRemainingTimeInMillis());
    console.log('functionName =', context.functionName);
    console.log('AWSrequestID =', context.awsRequestId);
    console.log('logGroupName =', context.logGroupName);
    console.log('logStreamName =', context.logStreamName);
    console.log('clientContext =', context.clientContext);
    if (typeof context.identity !== 'undefined') {
        console.log('Cognito
        identity ID =', context.identity.cognitoIdentityId);
    }
    callback(null, event.key1); // Echo back the first key value
    // or
    // callback("some error type");
};
```

Lambda Event Model

AWS Lambda – Push Model

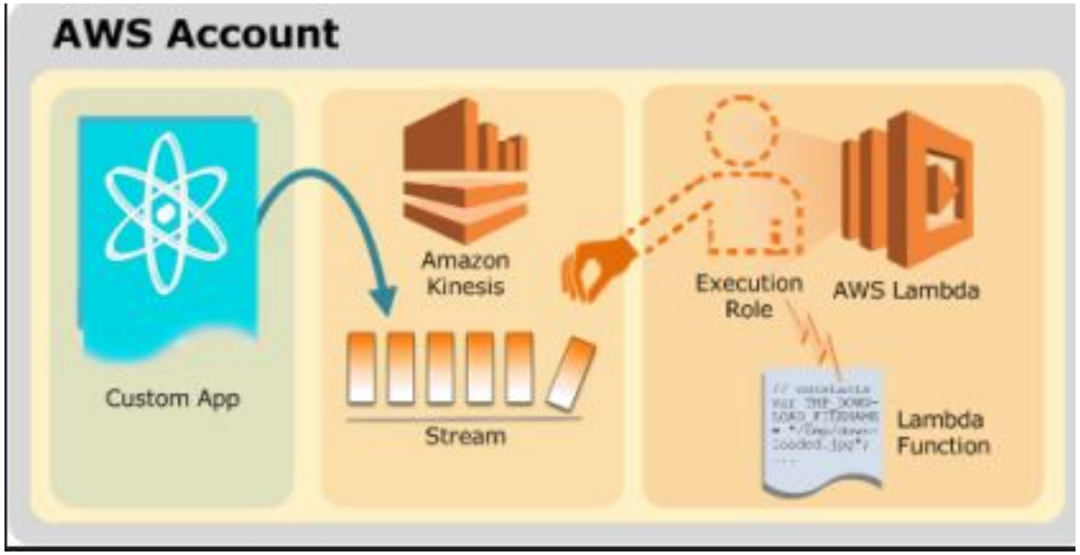
- Event source **pushes** the event to Lambda
- Event source mapping is held in event source

Services :
- Amazon S3
- Amazon SNS
- Amazon Cognito
- Amazon Echo



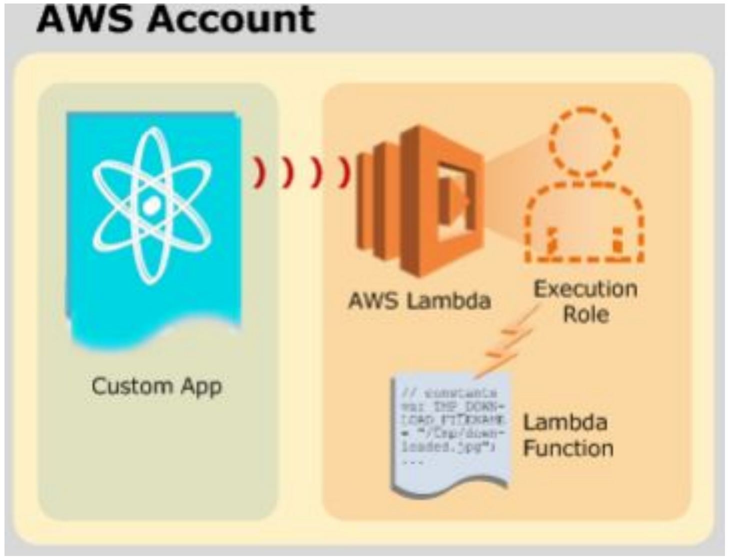
AWS Lambda – Pull Model

- **Polls** the event source
- Event source mapping held in Lambda
- **Services**
  - Amazon Kinesis
  - Amazon DynamoDB Streams



AWS Lambda – Direct Invocation Model

- Respond to invocation
- Services
    - Custom code
    - Amazon API Gateway



Authentication & Authorization

AWS Lambda Permissions

- Private by default
- Must grant permissions to services and resources
- Service invoking Lambda function must have permission
- Execution Role
    - IAM role
    - Lambda uses this role while executing lambda function
- Permissions for Services
    - Push model
        - Service must have permission to invoke lambda function
    - Pull model
        - Lambda must have permission to poll

Invocation Types

Invocation Types

- Synchronous

- Asynchronous

- Can be controlled only for on-demand invocation

- AWS Service event source has pre-defined invocation type
  - S3 - > asynchronous
  - Cognito - > synchronous
  - Kinesis & Dynamodb -> synchronous -Pull Model

Advantages of using Lambda

- Pay only for the compute time(in milliseconds) used.
- Don’t worry about server
- Spend more time in writing code.
- Bring your own code… even native libraries
- Never pay for idle time
- You don’t have to think about
    - Servers Being over/under capacity
    - Deployments
    - Scaling
    - Fault tolerance
    - OS or language updates
    - Metrics and
    - logging

Consequences

- Cannot choose underlying server’s OS
- Cannot customize underlying runtime
- Cannot log in to underlying instance
- Terrible at error handling
- Temporary Building block, not a tool
- 5 minutes timeout limit
- Memory : 128MB to 1.15GB
- Timeout : 100ms to 5min

Some more key Lambda Features

Controlling Concurrency

- **Controlling Concurrency**
  - limit the number of requests to control costs
  - limit the concurrent requests for downstream services
    - DB may not sale to the level of lambda
  - throttling the number of events processed
  - Can be set at Account level - default 1000
  - Can be set at function level
    - upto 900
    - 100 are reserved so that other functions can execute avoiding ‘starving ‘ situation

DLQ ,Retry and Failed Executions

- **Retry and Handling Errors**
  - AWS Lambda will **automatically retry f**ailed executions for **asynchronous** invocations.
  - You can additionally optionally configure Lambda to forward payloads that were not processed to a dead-letter queue (DLQ), such as an SQS queue or an SNS topic
- **What can cause lambda function to fail** ?
  - function times out while trying to reach an endpoint
  - input data issue
  - resource constraints, such as out-of-memory errors or other timeouts

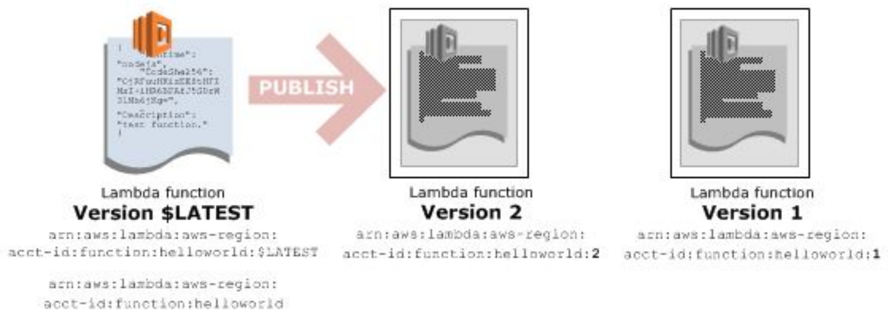
Versioning and Aliases

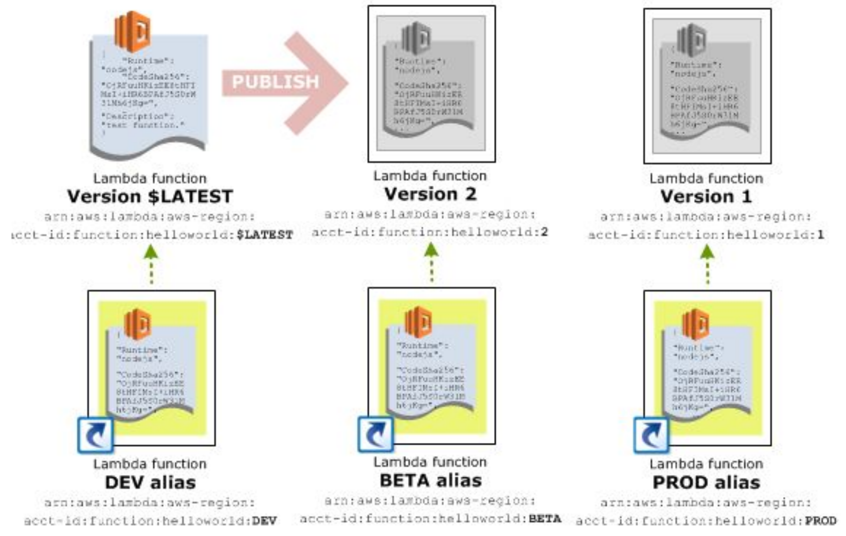
Versioning

- Create multiple versions of FUNCTION
- $LATEST is the default

Alias

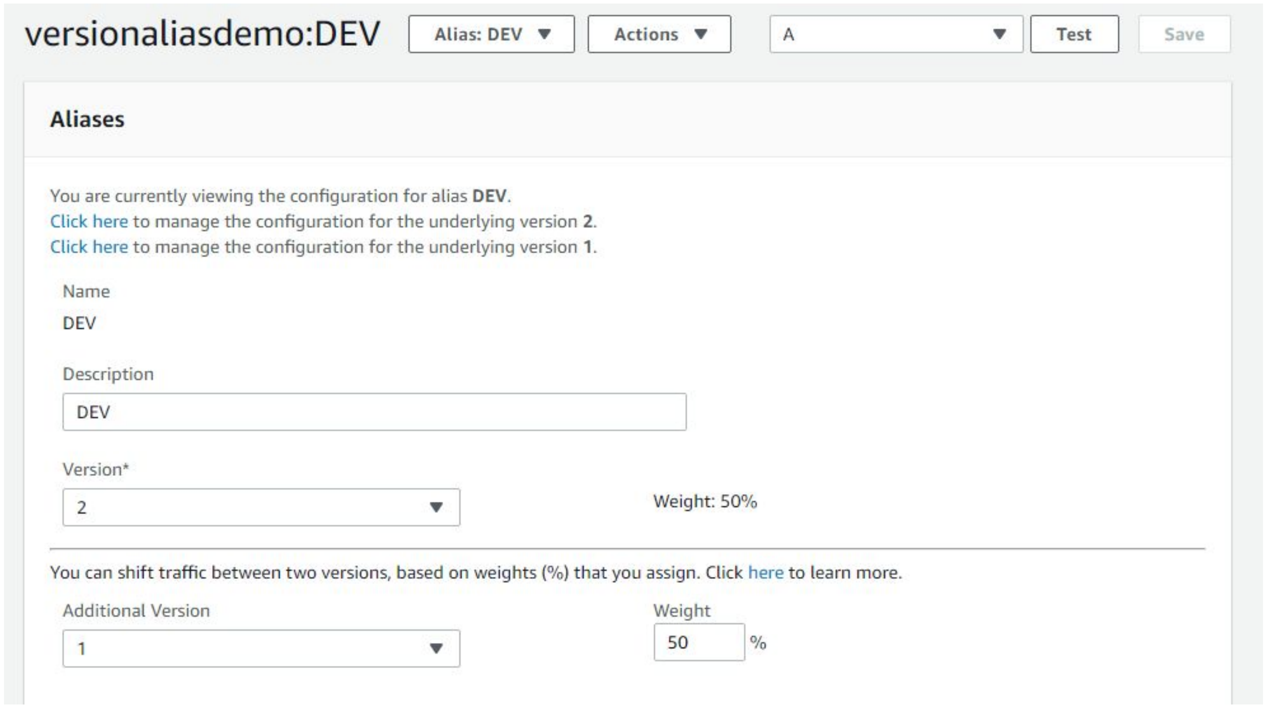
- Alias is a POINTER to specific version
- Very handy during updating new release
- rolling back to new release


PUBLISH
Lambda function
Version $LATEST
arn:aws:lambda:aws-region:
acct-id:function:helloworld:$LATEST
arn:aws:lambda:aws-region:
acct-id:function:helloworld
Lambda function
Version 2
arn:aws:lambda:aws-region:
acct-id:function:helloworld:2
Lambda function
Version 1
arn:aws:lambda:aws-region:
acct-id:function:helloworld:1


PUBLISH
Lambda function
Version $LATEST
arn:aws:lambda:aws-region:
acct-id:function:helloworld:$LATEST
Lambda function
Version 2
arn:aws:lambda:aws-region:
acct-id:function:helloworld:2
Lambda function
Version 1
arn:aws:lambda:aws-region:
acct-id:function:helloworld:1
Lambda function
DEV alias
arn:aws:lambda:aws-region:
acct-id:function:helloworld:DEV
Lambda function
BETA alias
arn:aws:lambda:aws-region:
acct-id:function:helloworld:BETA
Lambda function
PROD alias
arn:aws:lambda:aws-region:
acct-id:function:helloworld:PROD

Traffic Shifting using Aliases

- `routing-config` parameter of the Lambda alias that allows you to point to two different versions of the Lambda function and dictate what percentage of incoming traffic is sent to each version



You are currently viewing the configuration for alias **DEV**.
Click here to manage the configuration for the underlying version **2**.
Click here to manage the configuration for the underlying version **1**.

Name: DEV

Description: DEV

Version*: 2

Weight: 50%

You can shift traffic between two versions, based on weights (%) that you assign. Click here to learn more.

Additional Version: 1

Weight: 50 %

Best Practices for using Lambda

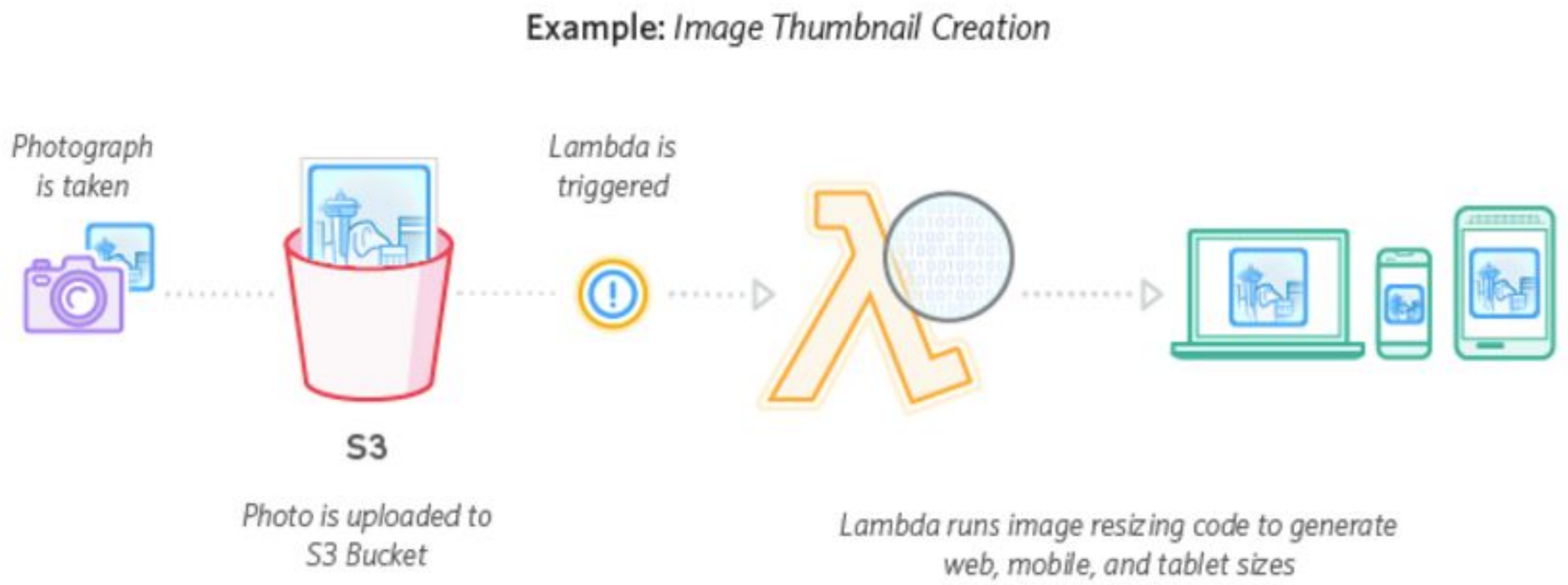
Best Practices

- Code
  - Write Stateless Code
- Configuration
  - setup **concurrency** level
- Control dependencies in your package Put all dependencies with your deployment package.
- Minimize the runtime dependencies Similar to import.* vs import java.util.Date
- Simplify the framework Use simpler IoC framework (Dagger) instead of Spring
- Avoid using recursive code

Use Cases

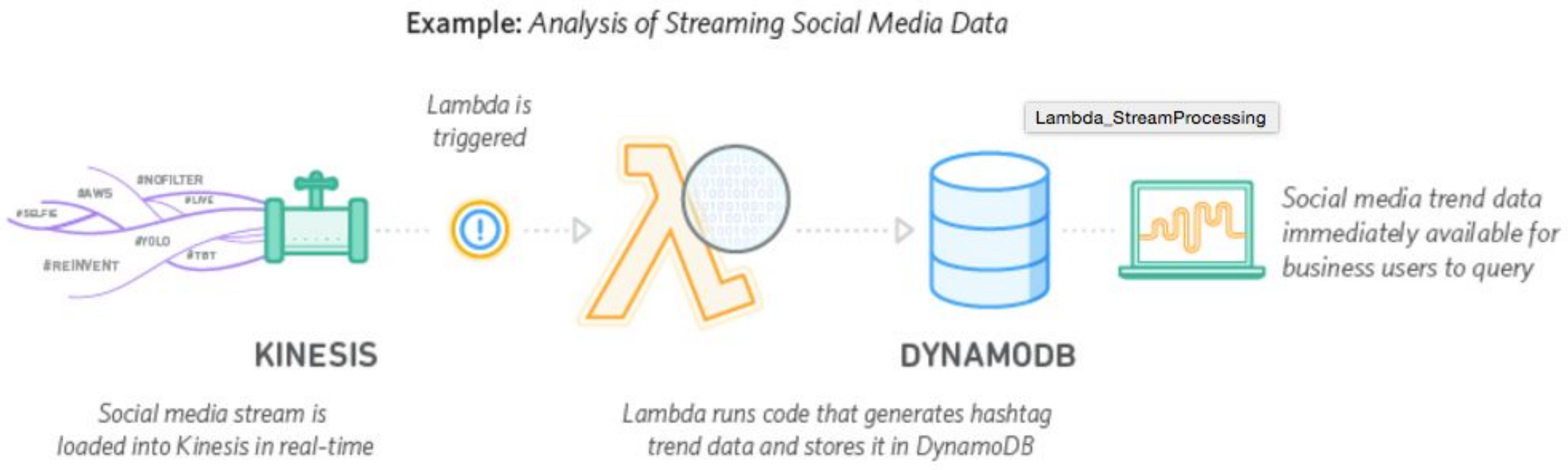
Real Time File Processing

- To execute code in response to triggers such as
  - changes in data
  - shifts in system state or
  - actions by users
- Can be directly triggered by AWS services such as S3, DynamoDB, Kinesis, SNS, and CloudWatch
- Real Time File Processing


Example: *Image Thumbnail Creation*
Photograph is taken
Lambda is triggered
S3
Photo is uploaded to S3 Bucket
Lambda runs image resizing code to generate web, mobile, and tablet sizes

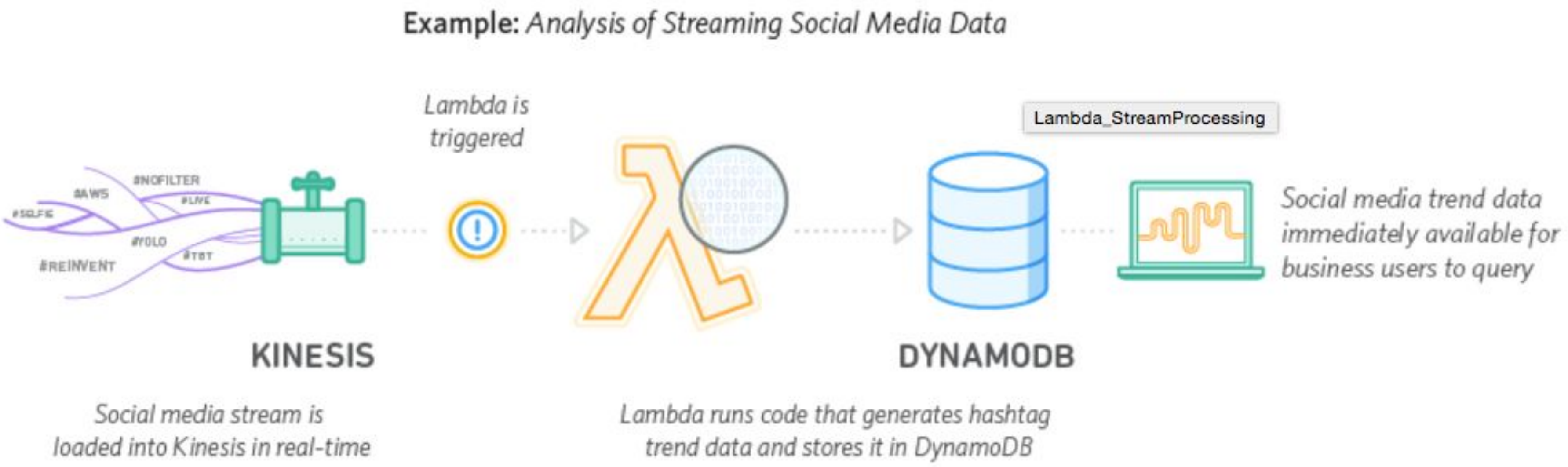
Real-time streaming analysis

- application activity tracking,
- transaction order processing
- click stream analysis
- data cleansing
- metrics generation
- log filtering, indexing, social media analysis, and IoT device data telemetry and metering.


Example: *Analysis of Streaming Social Media Data*
Lambda is triggered
Lambda_StreamProcessing
#NOFILTER
#AWS
#SELFIE
#LIVE
#YOLO
#TBT
#REINVENT
KINESIS
DYNAMODB
Social media trend data immediately available for business users to query
Social media stream is loaded into Kinesis in real-time
Lambda runs code that generates hashtag trend data and stores it in DynamoDB

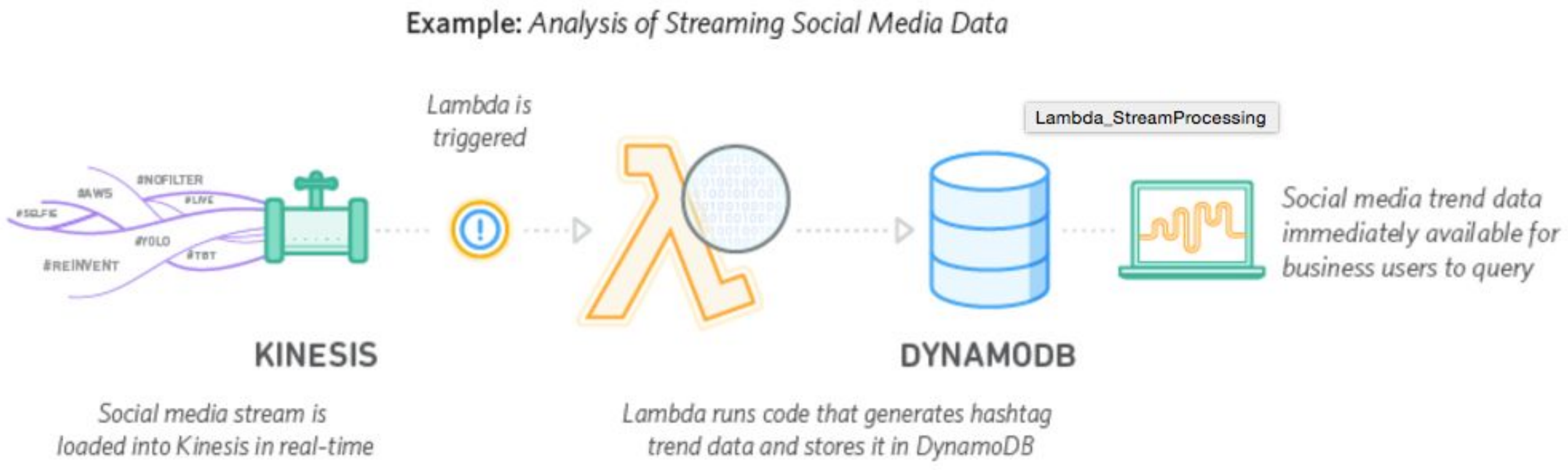
Extract , Transform and Load

- application activity tracking,
- transaction order processing
- click stream analysis
- data cleansing
- metrics generation
- log filtering, indexing, social media analysis, and IoT device data telemetry and metering.

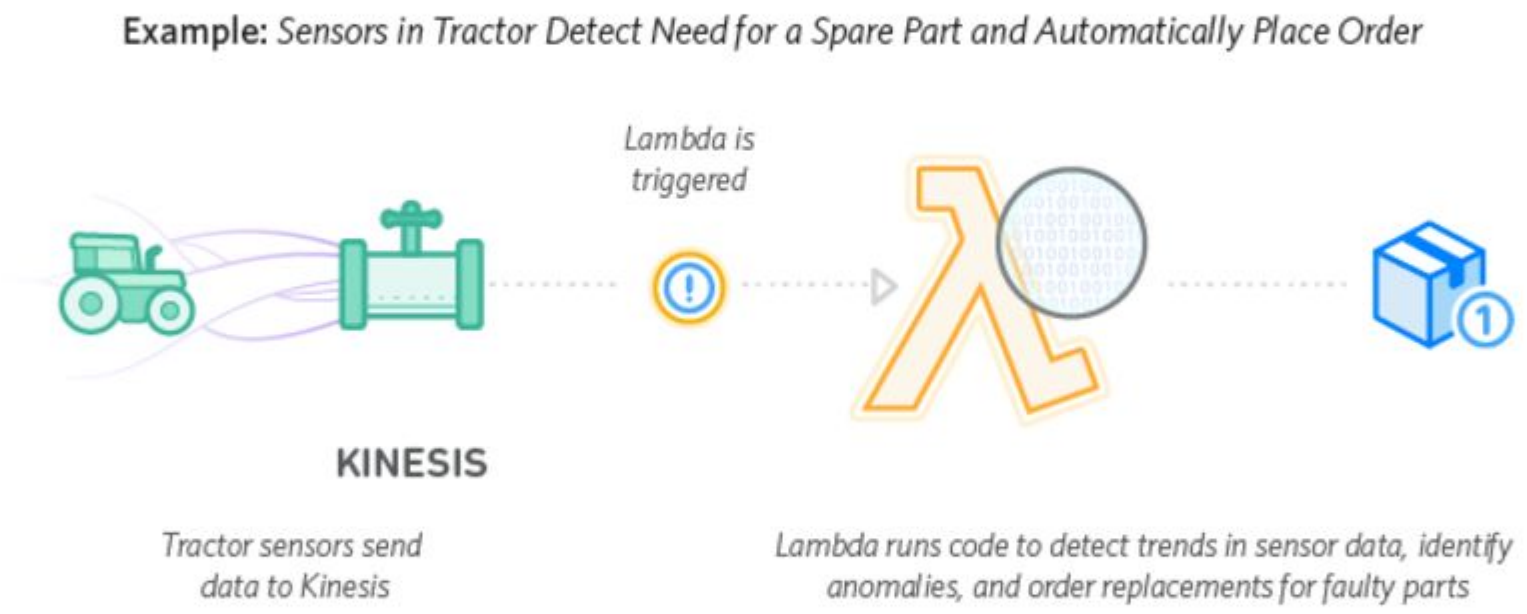

Example: *Analysis of Streaming Social Media Data*
#NOFILTER
#AWS
#LIVE
#SELFIE
#YOLO
#TBT
#REINVENT
KINESIS
Social media stream is loaded into Kinesis in real-time
Lambda is triggered
Lambda runs code that generates hashtag trend data and stores it in DynamoDB
Lambda_StreamProcessing
DYNAMODB
Social media trend data immediately available for business users to query

Extract , Transform and Load

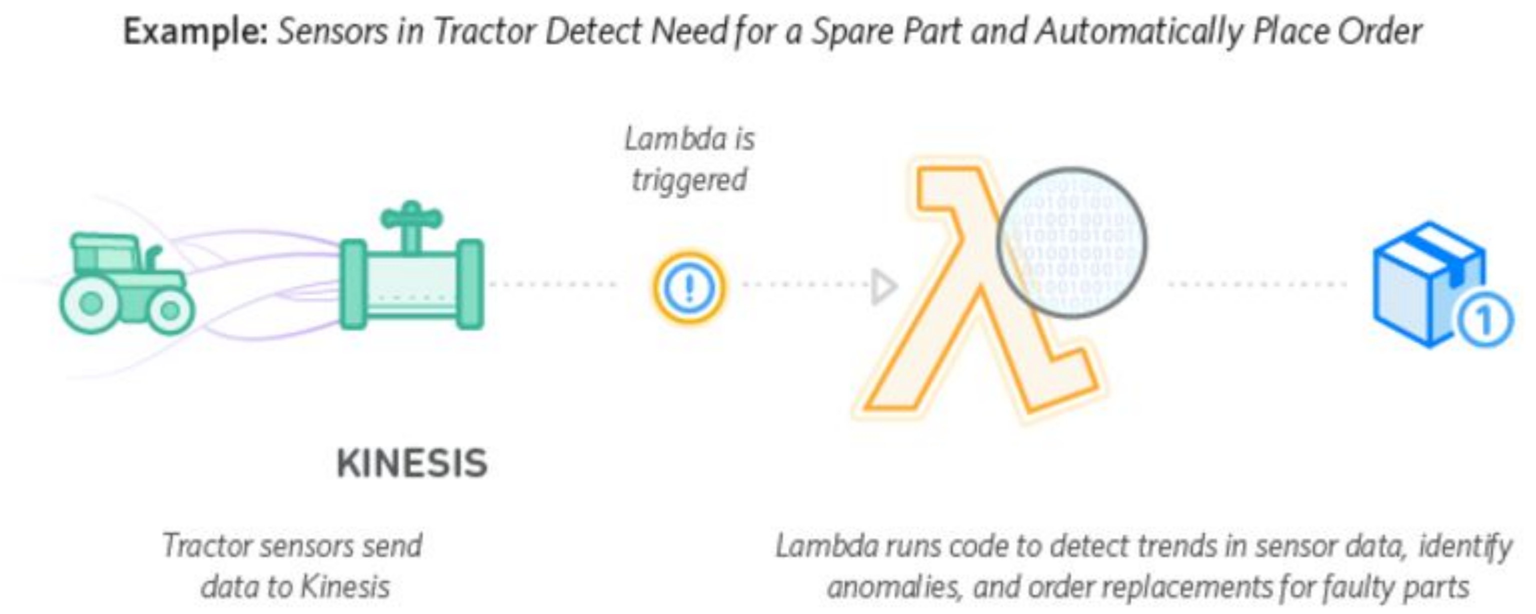
- application activity tracking,
- transaction order processing
- click stream analysis
- data cleansing
- metrics generation
- log filtering, indexing, social media analysis, and IoT device data telemetry and metering.


Example: *Analysis of Streaming Social Media Data*
#NOFILTER
#AWS
#LIVE
#SELFIE
#YOLO
#TBT
#REINVENT
Lambda is triggered
Lambda_StreamProcessing
Social media trend data immediately available for business users to query
KINESIS
DYNAMODB
Social media stream is loaded into Kinesis in real-time
Lambda runs code that generates hashtag trend data and stores it in DynamoDB

IoT Backends


Example: Sensors in Tractor Detect Need for a Spare Part and Automatically Place Order
Lambda is triggered
KINESIS
Tractor sensors send data to Kinesis
Lambda runs code to detect trends in sensor data, identify anomalies, and order replacements for faulty parts

IoT Backends


Example: *Sensors in Tractor Detect Need for a Spare Part and Automatically Place Order*
Lambda is triggered
KINESIS
Tractor sensors send data to Kinesis
Lambda runs code to detect trends in sensor data, identify anomalies, and order replacements for faulty parts

Pricing

Total= Compute charge + Request charge

Compute charge = Execution time * memory
Request charge = Monthly requests charge

Details pricing:

➢The Lambda free tier includes 1M free requests per month and 400,000 GB-seconds of compute time per month.

First 1 million requests per month are free
$0.20 per 1 million requests thereafter ($0.0000002 per request)

➢ Duration is calculated from the time your code begins executing until it returns or otherwise terminates, rounded up to the nearest 100ms.

➢ The price depends on the amount of memory you allocate to your function. You are charged $0.00001667 for every GB-second used.

Thank You