**Lambda**

**Why Lambda?**

**EC2:**

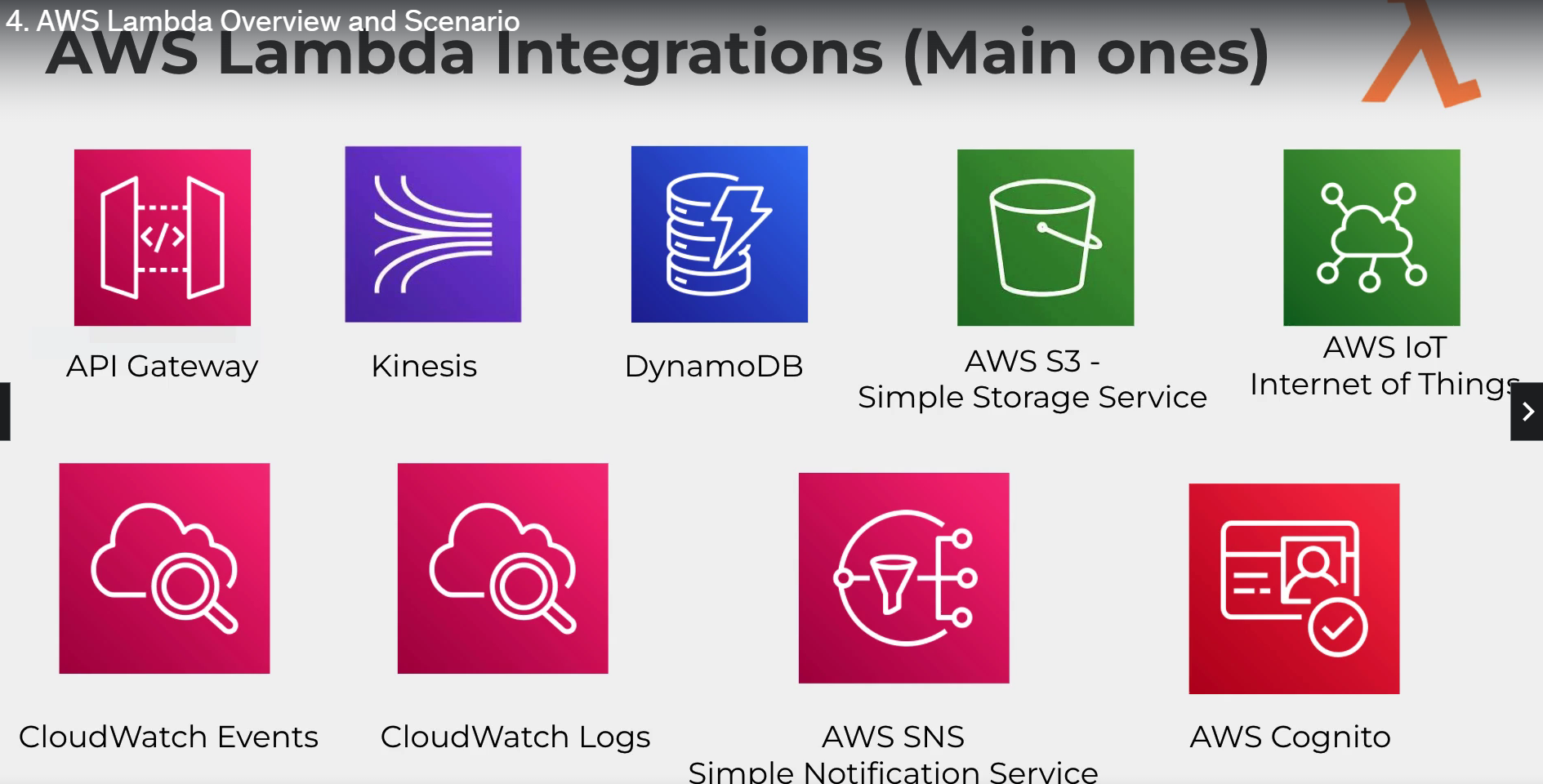
* Virtual server in the cloud
* Limited by RAM and CPU
* Continuously running
* Scaling – add/remove server

**Lambda:**

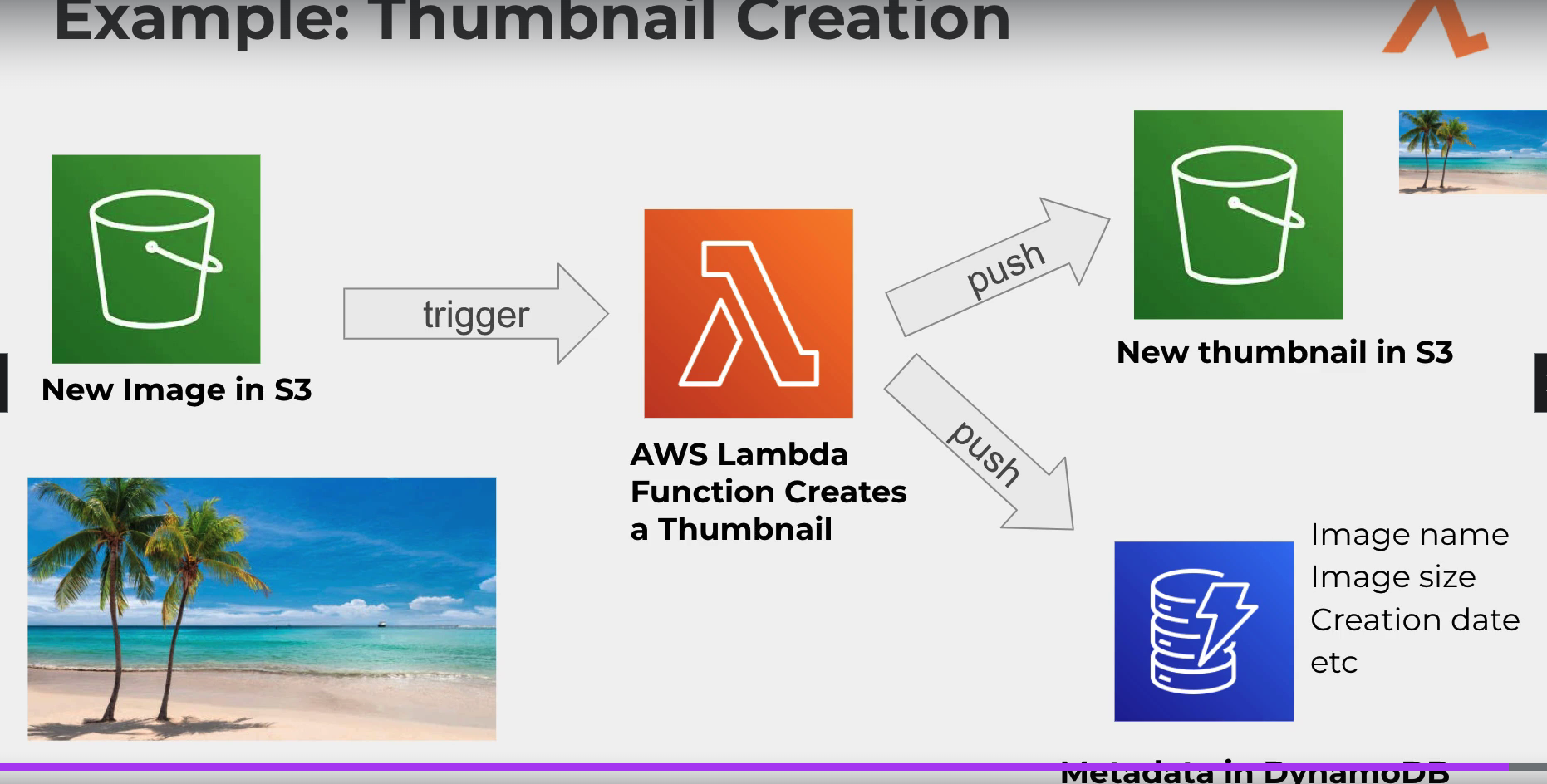
* Virtual functions, no server to manage
* Limited by time – short execution
* Run on-demand
* Scaling is automated

**Benefits of Lambda:**

1. Pricing is easy – Pay per request/Compute time
2. Integrated with all AWS Stacks
3. Integrated with so many programming languages
4. Easy to monitor through Cloud Watch
5. Easy to get more resources per function
6. Increasing RAM will also improve CPU and network



**Example1:**



**Serverless Framework:**

* Aims to ease the pain of creating, deploying, managing and de-bugging
* It integrated well with CI/CD tools
* It has CloudFormation support so your entire stack can be deployed using this framework.

**To install serverless:**

1. Install node.js
2. Once installed, run below command in command prompt/power shell.

**npm install -g serverless**

1. In AWS IAM console – create user ‘serverless-admin’ and attach admin policy, also give programmatic access and download credentials
2. In command prompt give access ID and secrete key using below command

**serverless config credentials –provider aws –key <Access key> --secret <secret key> --profile serverless-admin**

1. Give ‘serverless’ and choose a template **aws python-starter.** It will download the necessary packages for serverless deployment. Give “**code . “** to open the package in visual editor.
2. The function would have created in the AWS Lambda console
3. .py file with function will be there. Do small change and run below command to deploy the function.

**serverless deploy function -f <function\_name>**

**serverless deploy** *--to deploy entire stack*

1. To invoke function, run below command

**serverless invoke –function <function\_name>**

**serverless invoke –function <function\_name> --log** *–it will give log of the function*

**YAML**

YAML is a markup Language used to edit ‘serverles.yml’

* It has key-value pair like dictionary
* Nested objects - brown
* Support arrays - blue

---

doe: "a deer, a female deer"

ray: "a drop of golden sun"

pi: 3.14159

xmas: true

french-hens: 3

calling-birds:

  - huey

  - dewey

  - louie

  - fred

xmas-fifth-day:

  calling-birds: four

  french-hens: 3

  golden-rings: 5

  partridges:

    count: 1

    location: "a pear tree"

  turtle-doves: two

**example.yml:**

**json :**

**{**

**“Numbers” : [ “One”,”Two”,”Three”]**

**}**

**Yaml:**

**Numbers :**

**-One**

**-Two**

**-Three**

Also in yaml we can create anchor, where values defined at one place and be used at different places. While defining will use (&) and while invoking use (\*).

Authors: **&Jpaul**

Name: James

Last-name : Paul

Books:

-Rich dad Poor dad:

Author : **\*Jpaul**

-My next book:

Author : **\*Jpaul**

**AWS Functions and serverless Framework core concept:**

* **Functions**: An independent unit of deployment – code that can be deployed in the cloud

It should do a single job (saving user to Dynamo /convert png to thumbnail)

* **Events:** Anything that triggers an lambda function to execute (eg: Object upload in S3)
* **Resources:** AWS infrastructure component which your function use (eg: Dynamo db, S3)
* **Services:** Framework’s unit organization (project file)

1. Aws python-project
2. Where you define your function, the event which trigger them and the resources your function use.
3. The service can be described in YAML or JSON

**Commands executed from AWS CLI:**

**Open CloudShell in aws console:**

*--to check version*

**aws –version**

## Create the execution role

*--create execution role for lambda*

**aws iam create-role --role-name lambda-ex --assume-role-policy-document '{"Version": "2012-10-17","Statement": [{ "Effect": "Allow", "Principal": {"Service": "lambda.amazonaws.com"}, "Action": "sts:AssumeRole"}]}'**

*or create separate trust\_policy.json document*

**aws iam create-role --role-name lambda-ex --assume-role-policy-document** [**file://trust-policy.json**](file://trust-policy.json)

*--attach execution policy for lambda*

**aws iam attach-role-policy --role-name lambda-ex --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole**

## Create the function

*--Upload handler.py file to AWS CloudShell*

*--To deploy package in lambda, zip the file/ use container to deploy*

**zip function.zip handler.py**

*–Create Lambda function with below command*

**aws lambda create-function --function-name my-function --zip-file fileb://function.zip --handler handler.hello --runtime python3.10 --role arn:aws:iam::907360357323:role/lambda-ex**

*-- To get logs for an invocation from the command line. The response includes a LogResult field that contains up to 4 KB of base64-encoded logs from the invocation.*

**aws lambda invoke --function-name my-function out --log-type Tail**

## Update the function

*-- After you create a function, you can configure additional capabilities for the function, such as triggers, network access, and file system access. You can also adjust resources associated with the function, such as memory and concurrency. These configurations apply to functions defined as .zip file archives and to functions defined as container images.*

**aws lambda update-function-configuration --function-name my-function --memory-size 256 --timeout 6**

## List the Lambda functions in your account

*-- List the Lambda functions in your account*

**aws lambda list-functions --max-items 10**

## Retrieve a Lambda function

*The Lambda CLI get-function command returns Lambda function metadata and a presigned URL that you can use to download the function's deployment package.*

**aws lambda get-function --function-name my-function**

## Clean up

Run the following delete-function command to delete the my-function function.

**aws lambda delete-function --function-name my-function**