CloudPeak
TRAINING & CONSULTING

AWS Lambda in (a bit of) theory and in action

Adam Smolnik

Amazon Web Services
User Group Poland

A bit of a function theory

- The term Lambda (λ) originated from **Lambda calculus** - a theoretical universal model for describing functions and their computation

A function concept in programming

- Function represents a bit of reusable code
- May take arguments (aka parameters)
- May yield an outcome (pure function) and side effects (impure)
- Function's constituents
  - Signature – i.e. function's name, argument(s) and a return value*
  - Executable code embedded within a part called Body

```
public double exp(double a) {
        // Body comes here
}
```

* Definition of function's signature itself differs slightly across various programming languages and platforms

Pure vs. Impure function

- A pure function does not modify non-local data used beyond the function body
- An impure one may bring about side effects

Functional approach

- A function as the first class citizen in functional programming
- Declarative paradigm (as opposed to imperative one)
- Nowadays such a model has increased its importance as it is well suited for a concurrent, event-driven and reactive style of programming
- Enables runtime’s optimization for bulk operations on data collections or for processing a great deal of arriving events
- With statelessness in place largely supports and enhances scalability and parallelism of operations

AWS Lambda service

- Enables implementations that are able to react quickly to events
- Runs code in response to events such as file uploads
- Provides means to extend other AWS services with custom logic deployed and launched directly on AWS
- Performs all operational and administrative tasks
  - Including capacity provisioning, monitoring, applying security patches etc.
- Facilitates creating discrete, event-driven applications
  - Can scale automatically from a few requests per day up to thousands per second

AWS Lambda implementation

- Currently supporting Node.js
- From nodejs.org

  "Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices"

Push and Pull models

- Push model – an event producer (like Amazon S3) directly calls a Lambda function
  - The unordered model – the order Lambda processes events is unspecified
- Pull model – AWS Lambda pulls the updates from the Stream (for AWS Kinesis or DynamoDB*) and then invokes a function
  - The ordered model – events are processed in order they are published to the Stream

* DynamoDB Streams maintains a time ordered sequence of item level changes in a log for 24 hours

Essential AWS Lambda components

- Lambda Function itself along with dependent libraries
- Event Source
- Execution Role
- Invocation Role

Lambda Function syntax

- Skeleton code illustrates the straightforward syntax in which custom Node.js code (as a function) is written:

```
exports.handler_name = function(event, context) {
    console.log("value1 = " + event.key1);
    console.log("value2 = " + event.key2);
    ...
    context.done(null, "some message");
}
```

Event format

- Event structure and its content depend on its origin (source)
- Simple generic JSON structure for user-defined events

```
{
"key1":"value1",
"key2":"value2",
"key3":"value3"
}
```

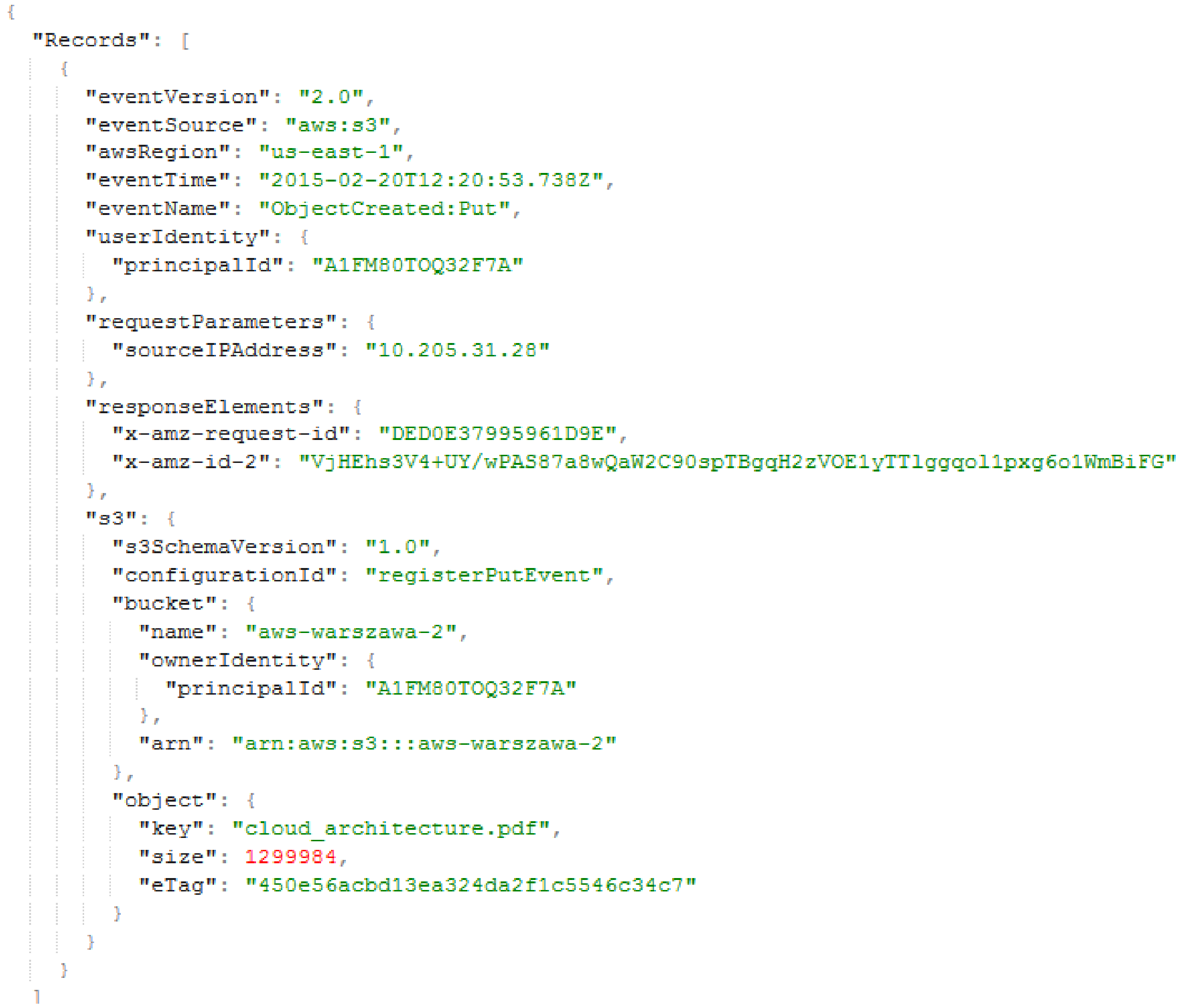
Execution Role

- Grants a function permissions to access AWS resources
- AWS Lambda assumes this role while executing code on behalf of the client

Invocation Role

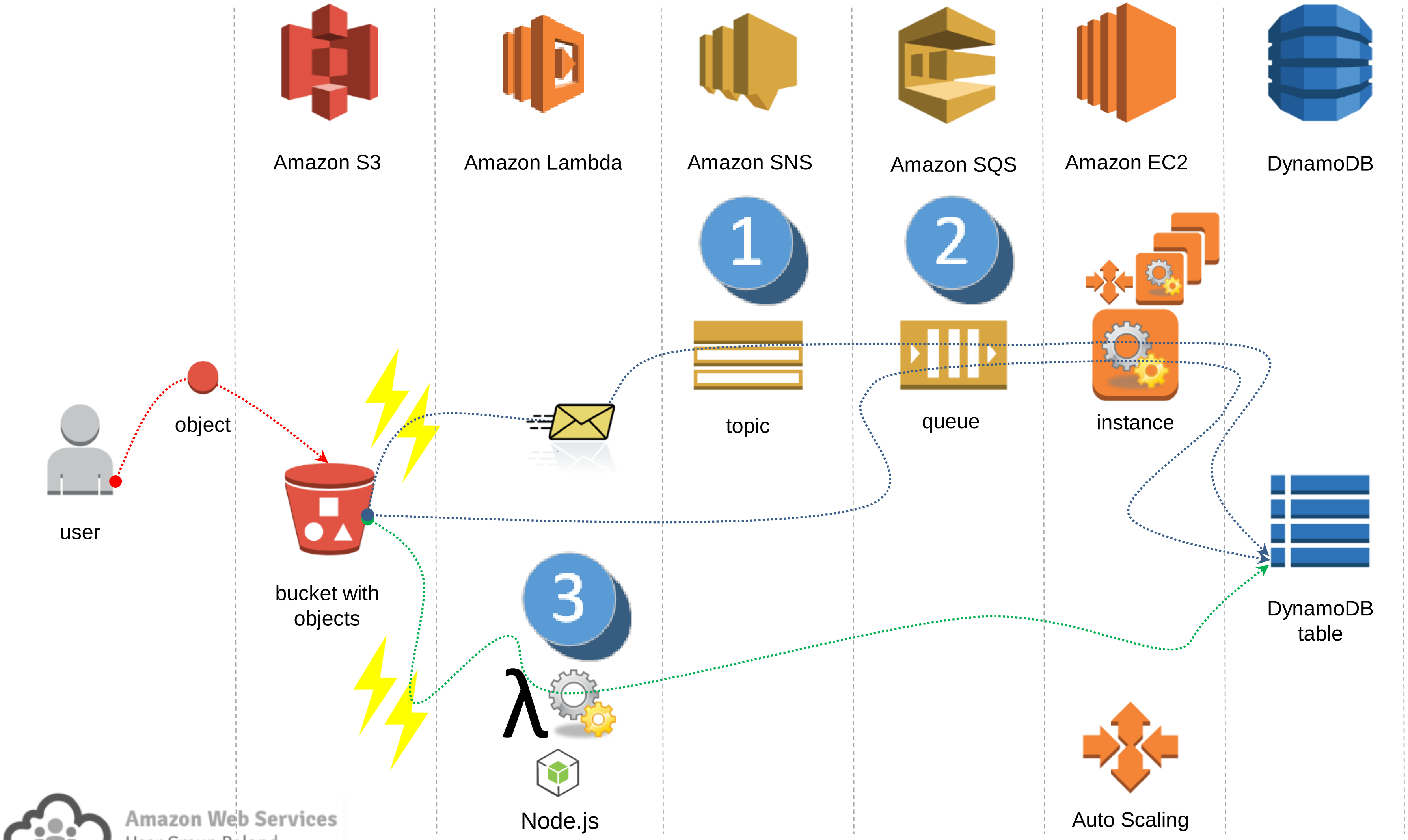
- Grants requisite permissions for the event source to leverage AWS Lambda’s components:
  - For the push model – grants permission to the event source to call a function
  - In the pull model – grants permission to AWS Lambda to allow pulling from a given Stream (AWS Kinesis or DynamoDB Stream)

Example of S3 Event content



```
{
  "Records": [
    {
      "eventVersion": "2.0",
      "eventSource": "aws:s3",
      "awsRegion": "us-east-1",
      "eventTime": "2015-02-20T12:20:53.738Z",
      "eventName": "ObjectCreated:Put",
      "userIdentity": {
        "principalId": "A1FM80TOQ32F7A"
      },
      "requestParameters": {
        "sourceIPAddress": "10.205.31.28"
      },
      "responseElements": {
        "x-amz-request-id": "DED0E37995961D9E",
        "x-amz-id-2": "VjHEhs3V4+UY/wPAS87a8wQaW2C90spTBgqH2zVOE1yTTlggqol1pxg6o1WmBiFG"
      },
      "s3": {
        "s3SchemaVersion": "1.0",
        "configurationId": "registerPutEvent",
        "bucket": {
          "name": "aws-warszawa-2",
          "ownerIdentity": {
            "principalId": "A1FM80TOQ32F7A"
          },
          "arn": "arn:aws:s3:::aws-warszawa-2"
        },
        "object": {
          "key": "cloud_architecture.pdf",
          "size": 1299984,
          "eTag": "450e56acbd13ea324da2f1c5546c34c7"
        }
      }
    }
  ]
```

Where Lambda can simplify design



Some Lambda limit (valid during the Lambda preview)

- Memory available – 128 ÷ 1024 MB
- Ephemeral disk capacity – 512 MB
- Total number of processes and threads – (256?) 1024
- Concurrent requests – 25 per second
- Execution duration per request – 60 seconds (max)
- Compressed function .zip file – (20?) 30 MB
- Uncompressed function .zip file – 250 MB

Costs incurred

- Pay-for-use pricing model
  - Per request to call a function
    - First 1 million requests per month are free
    - $0.20 per 1 million requests henceforth
  - Duration – function's execution time
    - $0.00001667 for every GB-second used
- Example

A function with 512MB of memory allocated, run 3 million times in 1 month, and it took 2 second of processing each time.

Request charges per month (1 000 000 = 1M)
3M requests – 1M free tier requests = 2M
Request charges = 2M * $0.2/M = $0.40

Compute charges per month
Total compute (seconds) = 3M * 2s = 6M seconds
Total compute (GB-s) = 6M * 512MB/1024 = 3M GB-s
Total compute – Free tier compute = 3M GB-s – 0.4M free tier GB-s = 2.6M GB-s
Compute charges = 2.6 * $0.00001667 = $43.34

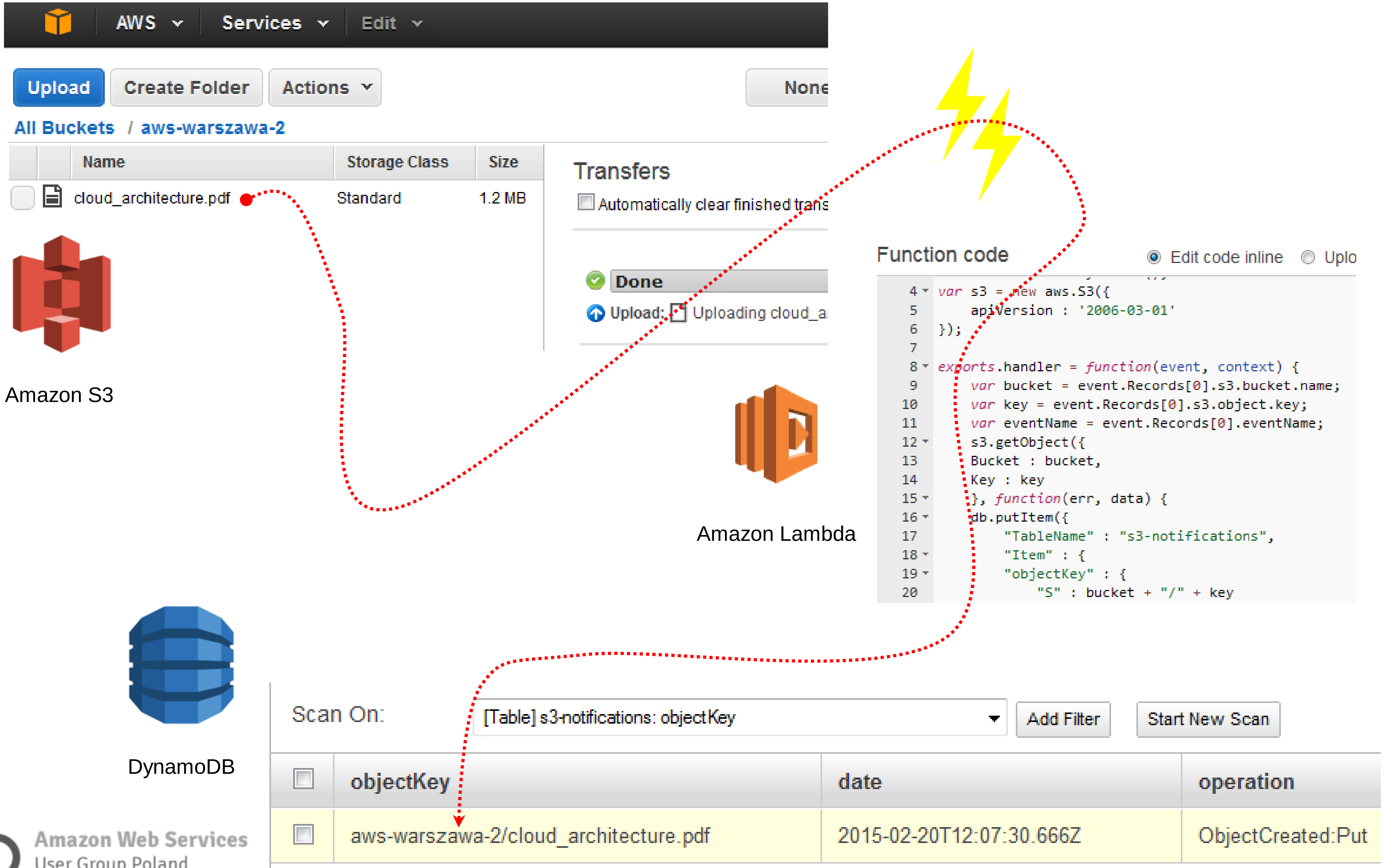
Total charges
Total charges = **Request charges** + **Compute charges** = $43.34 + $0.40 = **$43.74** per month

*The Lambda free tier does not automatically expire at the end of 12 month AWS Free Tier term

Potential downsides

- Less control over the code execution
- Troubleshooting issues due to business logic dispersed over various components
- Only Node.js implementation available for the time being

(Fast) Live Cooking – Lambda at work



Name	Storage Class	Size
cloud_architecture.pdf	Standard	1.2 MB

□ Automatically clear finished trans

Upload: Uploading cloud_a

Function code ● Edit code inline ○ Uplo

```
var s3 = new aws.S3({
    apiVersion : '2006-03-01'
});

exports.handler = function(event, context) {
    var bucket = event.Records[0].s3.bucket.name;
    var key = event.Records[0].s3.object.key;
    var eventName = event.Records[0].eventName;
    s3.getObject({
    Bucket : bucket,
    Key : key
    }, function(err, data) {
    db.putItem({
        "TableName" : "s3-notifications",
        "Item" : {
        "objectKey" : {
            "S" : bucket + "/" + key
```

Scan On: [Table] s3-notifications: objectKey

□	objectKey	date	operation
□	aws-warszawa-2/cloud_architecture.pdf	2015-02-20T12:07:30.666Z	ObjectCreated:Put