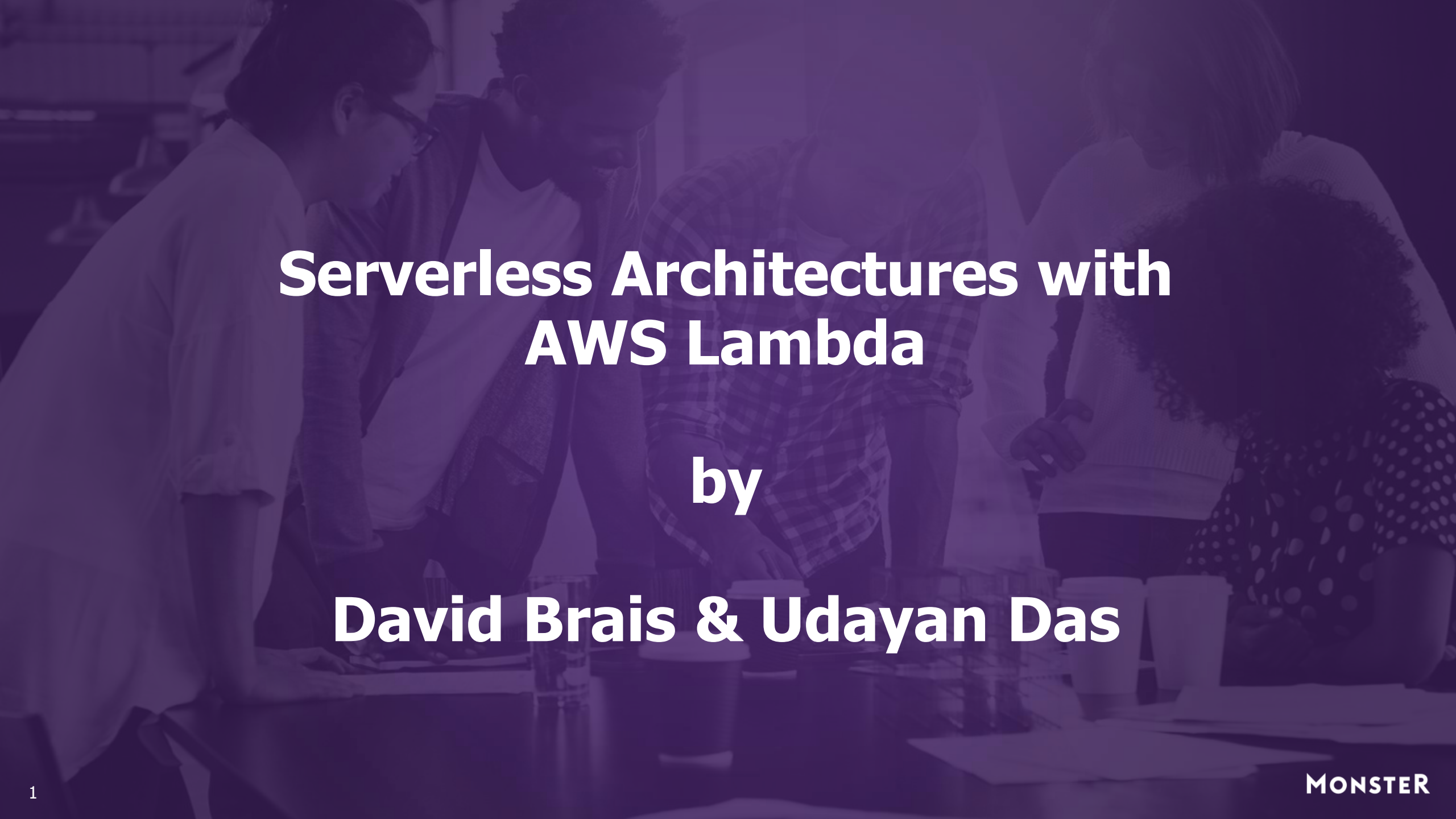


Serverless Architectures with AWS Lambda

by

David Brais & Udayan Das

MONSTER

AGENDA

- AWS Lambda Basics
- Invoking Lambda
- Setting up Lambda Handlers
- Use Cases
  - ASP .NET Web Service
  - Log Processing with AWS Lambda + SNS + Spark
- Q & A

AWS LAMBDA – THEORY

- Server-less way to run applications

+ No need to care about the server (patching etc.).
+ We do not have to take care of scaling.
+ Pay for execution only (we do not pay when not running anything).

Losing some level of visibility. -
Server-less makes some things more complex (example logging, managing state). -

- Supports .NET (.NET Core Only)

+ We waited for that.
+ Seems to be the direction Microsoft is going.

Not 100% stable tooling. -
Will change relatively soon and pretty significantly (.NET Core 2.0 / .NET Standard). -

AWS LAMBDA - INVOCATION

You can **call it directly**

Let's send it an **SNS** message

How about exposing it as a service via **API Gateway**

Run it regularly with **CloudWatch Scheduled Events**

CloudWatch Logs can trigger your Lambda on number of events

S3 can do the same

Integrate it with your **Kinesis Stream**

Or with your **DynamoDB** stream or events

AWS LAMBDA – FUNCTION HANDLER (.NET)

Sync

```
public static class QualityProcessorFunction
{
    [LambdaSerializer(typeof(JsonSerializer))]
    0 references | maintainability 65
    public static LambdaProcessorResultWrapper<QualityProcessorResponse> FunctionHandler(QualityProcessorRequest request, ILambdaContext context)...
}
```

Async

```
public static class CustomMetricsPopulator
{
    [LambdaSerializer(typeof(JsonSerializer))]
    0 references | maintainability 59
    public static void FunctionHandler(List<CustomMetric> customMetrics, ILambdaContext context)...
}
```

Template

```
"Resources": {
    "BgCacheUpdateManagerLambdaFunction": {
        "Type": "AWS::Lambda::Function",
        "Properties": {
            "Handler": "Lambda::Monster.Apps.BGW.JobAggregation.CacheUpdate.Lambda.CacheUpdatePopulator::FunctionHandler",
            "FunctionName": {
                "Ref": "FunctionName"
            },
```

- *Will see later how to setup Lambda function handlers in Java world*

AWS LAMBDA – FUNCTION HANDLER

Input / Output of Function Handler:

- Primitives (string, int, …)
- Stream (supported by default)
- Void (return type for async. invocations)
- Predefined AWS event types
- Custom types
- Collections, maps
- Task, Task<T> (if one is using .NET asynchronous programming)

If the type is not stream or primitive, one needs to define serializer to use:

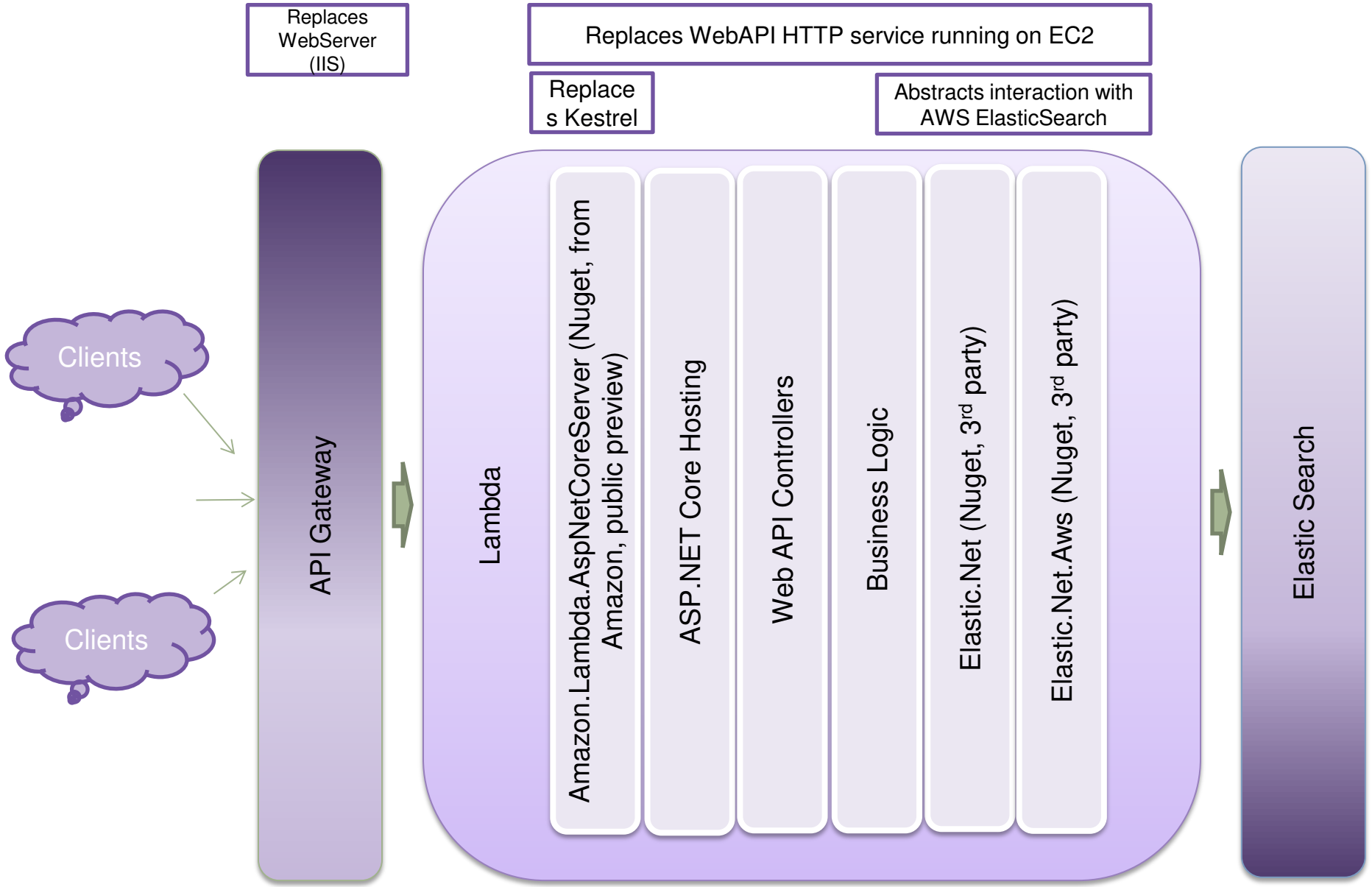
- Predefined Amazon.Lambda.Serialization.Json.JsonSerializer
- Custom implementation of ILambdaSerializer

AWS LAMBDA – ASP.NET CORE WEB API'S

Let’s distinguish between:

- The fact that you can expose pretty much any AWS Lambda through API Gateway as a Web Service
- There is a specific pattern how to do it that has a lots of advantages
  - Based on Amazon pattern as described in “Running Serverless ASP.NET Core Web APIs with Amazon Lambda” - https://aws.amazon.com/blogs/developer/running-serverless-asp-net-core-web-apis-with-amazon-lambda/
  - Built around Amazon.Lambda.AspNetCoreServer

AWS LAMBDA – ASP.NET CORE WEB API'S – GEOS SERVICE



MONSTER

AWS LAMBDA – ASP.NET CORE WEB API'S – GEOS SERVICE

MONSTER

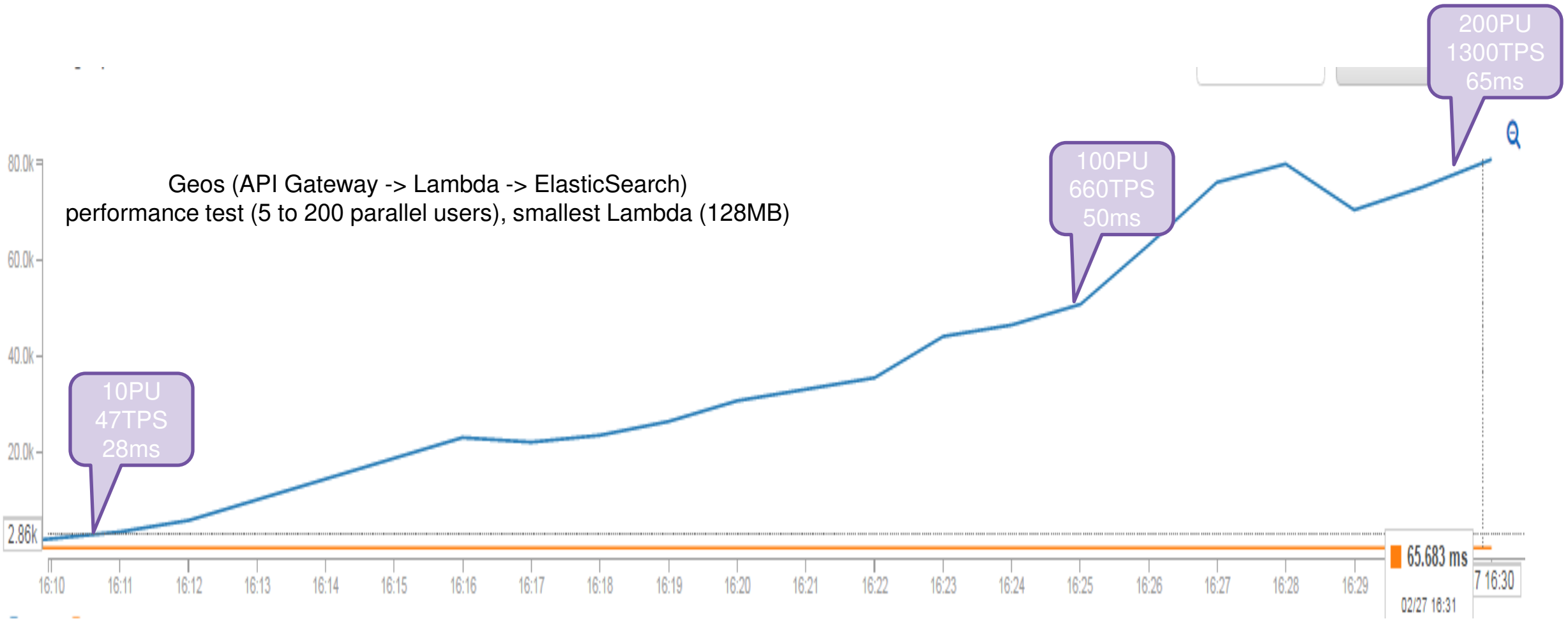
AWS LAMBDA – ASP.NET CORE WEB API'S

Advantages of following this pattern:

- Working with well known pattern of HTTP Web APIs
- Ability to turn into and deploy as Web Service with almost no code changes
- Ability to run and debug locally as a Web Service
- No 24/7 running EC2 instances
- Check in -> deploy in ~5 minutes (no AMI baking)
- “Built-in” scaling

AWS LAMBDA – ASP.NET CORE WEB API'S - PERFORMANCE

Geos (API Gateway -> Lambda -> ElasticSearch)
performance test (5 to 200 parallel users), smallest Lambda (128MB)



MONSTER

AWS LAMBDA – COST SAVINGS

Web Api Request Volume to Support : **1300 TPS**

EC2

8 c3.large @ $0.165/hr (reserved)

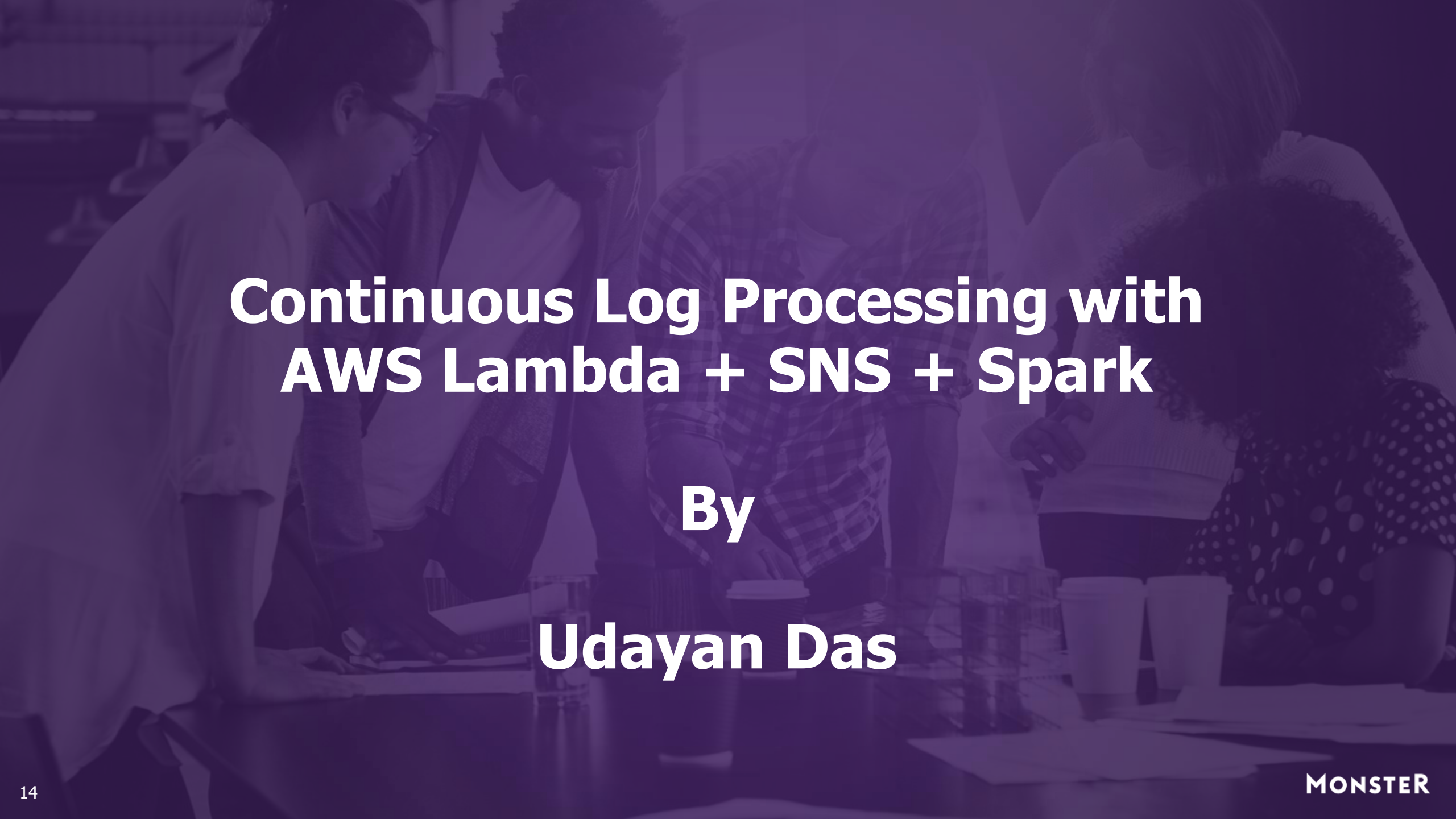
Monthly: $31.68

Lambda

56M requests
128 mb + 65 ms execution

Monthly: $11.97

⅓ the cost of hosting on EC2 !!!
~62% cost savings



Continuous Log Processing with AWS Lambda + SNS + Spark

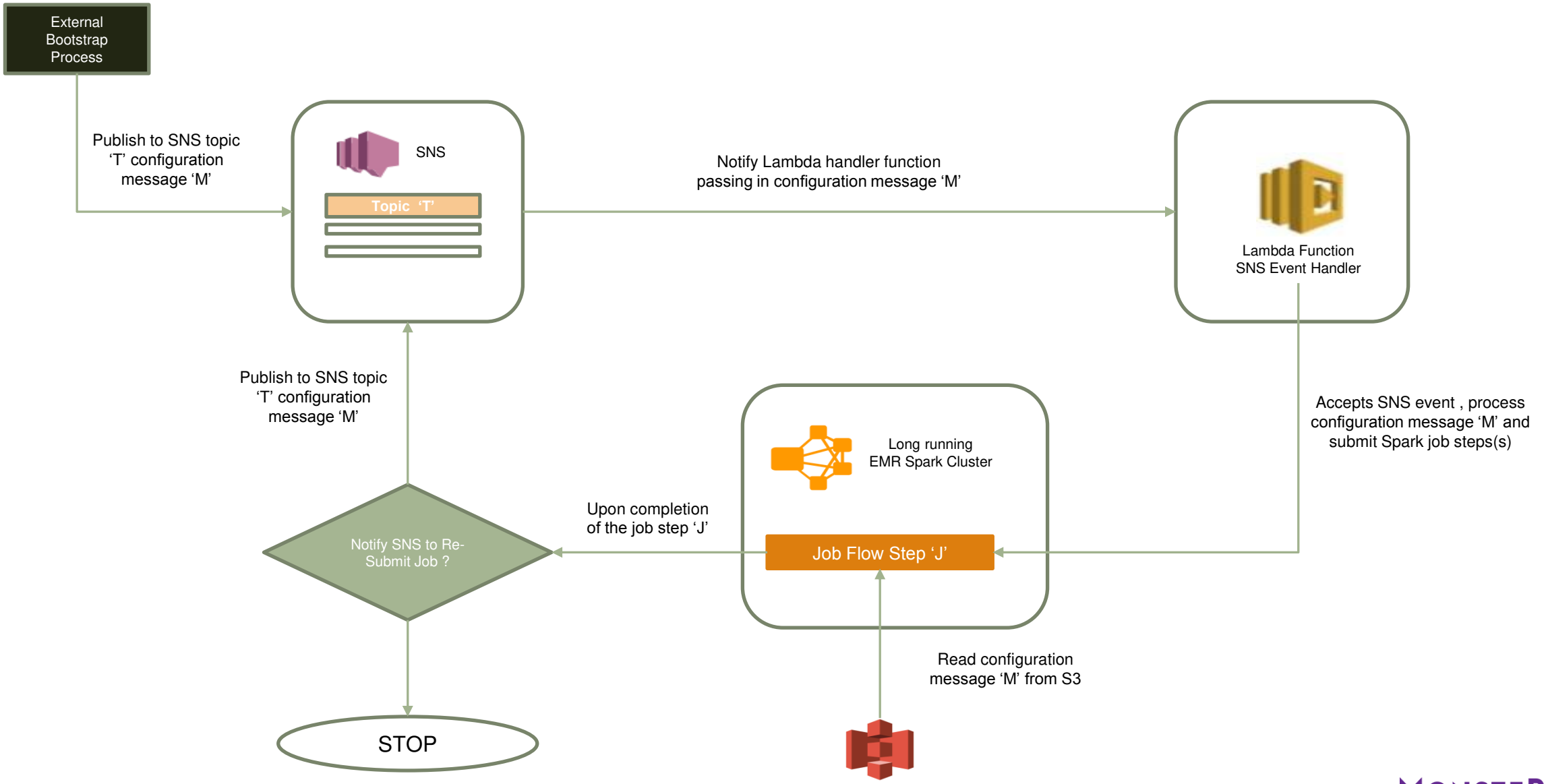
By

Udayan Das

DESIGN GOALS

- Continuously process S3 logs to generate aggregated views and store them into ES
- Handle up to ~25K events / sec
- Support easy plug-n-play of different types of aggregation jobs
- Provide support for one-off and scheduled processing over historical time period
- Be able to process data with varying volume and velocity
- Be able to pause and restart continuous aggregations
- Don’t want to have pre-configured scheduled cron jobs
- Use the same cluster for both batch and stream log processing

ARCHITECTURE



MONSTER

AWS LAMBDA + SNS SETUP

Create SNS Topic and the IAM Execution Role (1)

1. Using AWS IAM service, create a policy named "**SNSRoleEMRAction**" with the policy document as shown in **Fig 1.**
2. Create an IAM role (execution role). As you follow the steps to create a role, note the following:
   a. In Role Name, use a name that is unique within your AWS account (for example, **lambda-sns-execution-role**).
   b. In Select Role Type, choose **AWS Service Roles**, and then choose **AWS Lambda**. This grants the AWS Lambda service permissions to assume the role.
   c. In Attach Policy, choose **AWSLambdaBasicExecutionRole** and **SNSRoleEMRAction**
3. Write down the role ARN. You will need it in the next step when you create your Lambda function (C)
4. Create SNS topic as follows (and make not of the generated TopicArn – you will need it in Step 3) (A)
   a. aws sns create-topic --region us-east-1 --name okeanos-tasks

Fig 1

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": "*",
      "Action": [
        "elasticmapreduce:Describe*",
        "elasticmapreduce:ListSteps",
        "elasticmapreduce:AddJobFlowSteps",
        "s3:*",
        "sns:*"
      ]
    }
  ]
}
```

Create Lambda Function Handler (2)

```
aws lambda create-function ^
  --region us-east-1 ^
  --function-name okeanos-tasksubmit-handler ^        (D)
  --runtime java8 ^
  --role arn:aws:iam::[redacted]role/okeanos-lambda-sns-exec-role ^        (C)
  --handler com.monster.avengers.can.lambda.EMRStepSubmitHandler ^
  --description "Trigger step submission" ^
  --timeout 60 ^
  --memory-size 512 ^
  --zip-file fileb://<fullpath>/<your-lambda-function>.jar
```

In the output make note of the "FunctionArn" (you will need it in Step 4) (B)

Lambda Function Handler *(details in next slide)*

Add permission to Lambda Function (3)

```
aws lambda add-permission ^
  --region us-east-1 ^
  --function-name okeanos-tasksubmit-handler ^        (D)
  --statement-id okeanos-tasksubmit-handler ^        (D)
  --action "lambda:InvokeFunction" ^
  --principal sns.amazonaws.com ^
  --source-arn arn:aws:sns:us-east-1:[redacted]okeanos-tasks        (A)
```

This allows the Lambda function handler to be triggered by a source, in this case an SNS event

Setup Lambda Function as Subscriber to SNS Topic (4)

```
aws sns subscribe ^
  --region us-east-1 ^
  --topic-arn arn:aws:sns:us-east-1[redacted]okeanos-tasks ^        (A)
  --protocol lambda ^
  --notification-endpoint arn:aws:lambda:us-east-1[redacted]function:okeanos-tasksubmit-handler        (B)
```

This sets up the Lambda function as a subscriber to listen to SNS endpoint for new messages

Setup External Notification to SNS (5)

```
aws sns publish ^
  --region us-east-1 ^
  --topic-arn arn:aws:sns:us-east-1:[redacted]okeanos-tasks ^        (A)
  --message file://D:\msgfile-for-sns-dev.json ^
  --subject aggregate_clicks_by_day_DEV
```

Kick-off the processing cycle. 'Message file' is the configuration file that will be passed to the Lambda function

Update Existing Lambda Function

```
aws lambda update-function-code ^
  --region us-east-1 ^
  --function-name okeanos-tasksubmit-handler ^        (D)
  --zip-file fileb://<fullpath>/<your-lambda-function >.jar
```

MONSTER

AWS LAMBDA – FUNCTION HANDLER (JAVA)

Request Handler

```
public class EMRStepSubmitHandler implements RequestHandler<SNSEvent, Object> {

    @Override
     public Object handleRequest(SNSEvent request, Context context){
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd_HH:mm:ss");
        String timeStamp = sdf.format(Calendar.getInstance().getTime());
        context.getLogger().log("Invocation started: " + timeStamp);
        String msg_from_sns = request.getRecords().get(0).getSNS().getMessage();
        SparkStepManager ssmgr = new SparkStepManager(msg_from_sns,context);
        ssmgr.submitStep();
        timeStamp = sdf.format(Calendar.getInstance().getTime());
        context.getLogger().log("Invocation completed: " + timeStamp);
        return msg_from_sns + ":"+ msg_from_sns;
    }

}
```

Lambda function to handle incoming SNS event messages

Get the SNS Message 'M'

- Pass the SNS Message 'M' (see below) and the context reference to the Spark Step Manager and then call 'submitStep()'
- SparkStepManager processes the message 'M' and makes use of **EMR SDK** to submit jobs to a running cluster

SNS Message 'M'

```
{
    "clusterid": "[redacted]",
    "classname": "com.monster.avengers.raptor.spark.DailyAggregation",
    "stepname": "Aggregation JOBPERF Today",
    "deploymode": "client",
    "jarloc": "/home/hadoop/raptor/raptor2-1.8.0.jar",
    "jarargs": "-runasof=today,-lastnhours=2",
    "resubmit": true,
    "snstopicarn": "arn:aws:sns:us-east-1:[redacted]:okeanos-tasks",
    "s3configloc": "[redacted]/configurations/raptor/cluster/msgfile-for-jobperf-linux.json"
}
```

Id of the long running Spark Cluster

Driver class of Spark program

Job Flow Step Name

Deployment mode for Spark job

Spark program jar location

Notify SNS

Arguments to be passed to Spark's driver

SNS topic ARN

S3 location of the configuration file

MONSTER

SPARK CLUSTER

Steps

Filter: All steps | Filter loaded steps ... | 450 steps loaded | load more

	ID	Name	Status	Start time (UTC-4)	Elapsed time	Log files
▶	s-227L8VYXE5A55	Aggregation JOBPERF today	Pending			View logs
▶	s-YJLC1HYNADCJ	Aggregation JOBPERF today	Running	2017-07-18 21:12 (UTC-4)	1 minute	View logs
▶	s-3SX7X2PXPAYAU	Aggregation JOBPERF today	Completed	2017-07-18 21:10 (UTC-4)	1 minute	View logs
▶	s-5E7ZLCZ8OGFZ	Aggregation JOBPERF today	Completed	2017-07-18 21:08 (UTC-4)	1 minute	View logs

Down from ~40 mins i.e. ~90% improvement over scheduled batch job based processing

MONSTER

Q & A

Thank You