Serverless Application Design

Objectives

Key objectives of this chapter

- What is serverless Architecture?
- Event-Triggered Computing
- Lambda Functions

1.1 What is Serverless Architecture?

- "It's important to understand that a serverless architecture is the approach in which developers code business logic as functions, forgetting about the server's provisioning and scaling concerns where the logic will be executed." - Martin Fowler
- Serverless architecture was born through the initiative of Amazon.
- Its goal is to provide development team an environment where the team could be autonomous, small, and self-managed
- It allows development teams to work on the whole software development cycle, from writing the code to shipping and delivering to production environments.

1.2 Serverless Architecture - Examples

- Common examples
  - Authentication
  - SMS notifications
  - Email services
- Custom business logic can be coded in the form of serverless architecture that can be deployed on the cloud
  - Applications in which third-party services are adopted as a part of the system (rather than creating services from scratch).
  - These services are commonly known as backend as a service (BaaS) or mobile backend as a service (MBaaS).
  - we can code our custom business logic in the form of functions that can be deployed on the cloud. These services are known as function as a service (FaaS).

1.3 Infrastructure and File Storage in Cloud

- Traditional approach requires provisioning of all of the infrastructure, hardware and software, by using an on-premise environment
- This provisioning process has to be repeated for all of the environments, until we get into production
- Once in the production environment, other features are perform, such as scaling and monitoring.
- In many cases, the on-premises infrastructure will be under-utilized, which is a waste of money, since we have purchased powerful servers to deploy applications that don't need many resources.

- In the cloud, infrastructure and file storage are considered serverless, because we do not have to purchase, rent, or provision servers or virtual machines to use them

1.4 Serverless Architecture – Pros

- Developers can focus primarily on the code and can forget everything related to provisioning servers
- The scaling of the code is ephemeral, meaning that it can be scaled and can spin up or down, based on the number of requests retrieved.
- The tasks are focused on well-defined responsibilities, since they are loosely coupled and stateless
- Functions can be asynchronously triggered by events, since it's usually based on event-driven model.
- It's more cost-effective, since we only have to pay for the compute time consumed.
- Scaling can be achieved in a transparent way.

1.5 Serverless Architecture – Cons

- A lack of documentation and showcases that can be used as references
- Latency issues that are introduced when many services have to be consumed at the same time
- Some features are only available from certain cloud providers.
- Vendor lock-in

1.6 Backend as a Service

- The simplest scenario for using a BaaS approach occurs when we create single page applications (SPA), or mobile apps that interact with services available in the cloud.
- BaaS allows us to focus on the business activities without having to worry about the server or virtual machine of the application.
- BaaS provides us with a whole infrastructure and nodes, such as the following:
  - Load Balancer
  - Database to store our data (NoSQL or RDBMS), such as Amazon DynamoDB, Azure Cosmos DB
  - Filesystems
  - Queue servers
- BaaS also takes care of the following requirements:
  - Backup
  - Replication
  - Patches
  - Scale
  - High availability

1.7 Common Issues with serverless Architecture

- Vendor lock-in
- Security
- Framework support
- Troubleshooting

1.8 Vendor Lock-in

- Serverless is a new technology that will evolve with time
- There are ways to keep FaaS independent, such as by using Spring Cloud functionality.
- Moving to another vendor (from cloud-to-cloud) is not as difficult as it was in the past

1.9 Security

- All major cloud providers provides us with guides and practices for implementing authentication and authorization.
- A security concern to consider in serverless is the lack of a clearly shaped security perimeter.
- When the security perimeter of one of the functions ends and another starts, different cloud providers provide different ways to make those functions work as a whole; for example, AWS does this by using a service called an API Gateway.
- This API is used to orchestrate and compose the created FaaS.

1.10 Framework Support

- There are several frameworks working to create environments that develop serverless architectures without being attached to the cloud provider.
- As a function is a small piece of code with a clear input or output, it is better to use the language and technology that you feel comfortable with
- Currently, the most common options to deploy FaaS are as follows:
  - AWS Lamba
  - Azure functions
  - Google functions
- One benefit for Java developers is that most cloud providers offer support for Java as a programming language to deploy functions.
- Spring Framework has a project named Spring Functions that can be used to code functions
- A benefit of using Spring Functions is that we can develop and test our code in our local machine, and then wrap the code using an adaptor, in order to deploy it on a cloud provider.

1.11 Troubleshooting

- Once an application is deployed into production, one of the key aspects to consider is how to trace, find, and fix bugs.
- With serverless, this can be tricky, because we are dealing with a more segregated scenario, and our system has small parts that are not divided into services and microservices.
- Each cloud provider has tools to monitor and trace functions, dealing with

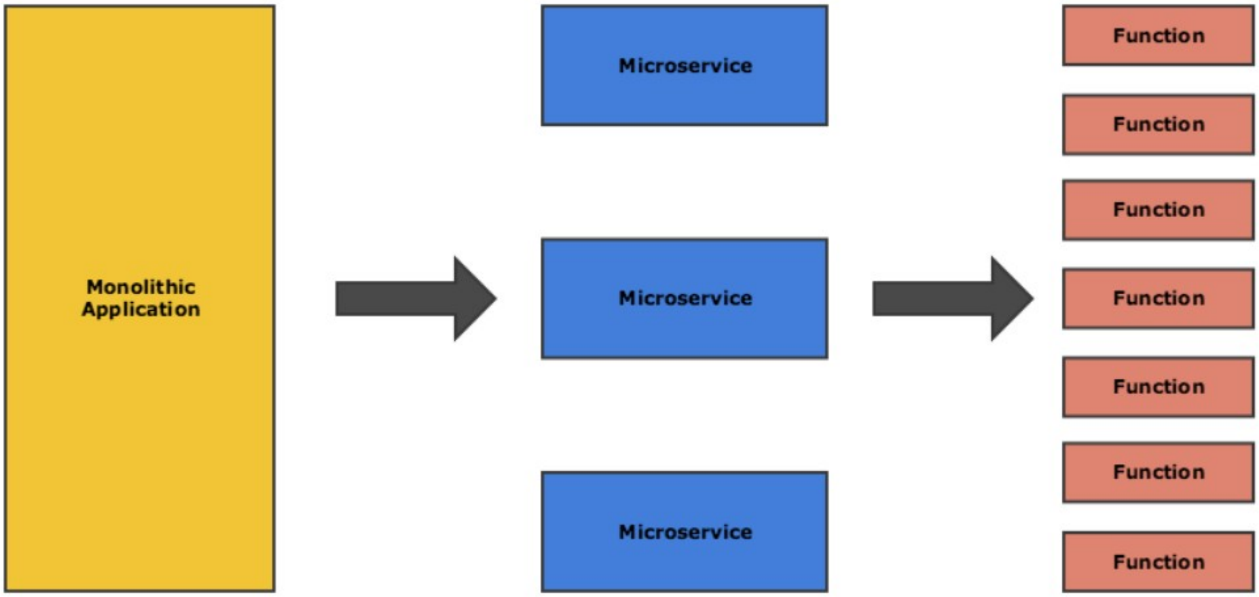
one error in an ephemeral environment.

- If we compose logic for several functions, we will have to apply techniques like aggregate logging and use tools to collect information associated with the executed code.

1.12 Serverless Architecture – Common Use Cases

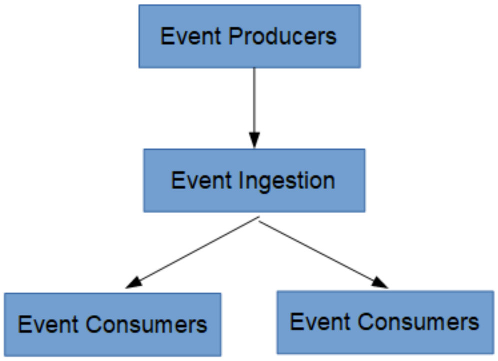
- Tasks or jobs that should be scheduled or triggered under certain circumstances
- Data transformation, for example:
  - Image manipulation, compression, or conversion
  - Voice data transcribed into text, such as Alexa or Cortana
- Single-page applications
- Chatbots

1.13 Monolithic vs Microservice vs FaaS



1.14 Serverless and Event-driven Computing

- Event-driven computation is an architecture pattern that emphasizes action in response to or based on the receptions of events.
- This pattern promotes loosely coupled services
- It ensures that a function executes only when it is triggered.
- It encourages developers to think about the type of events and responses a function needs in order to handle these events before programming the functions



- A system built with event-driven architecture consists of
  - Event Producers that produce a stream of events
  - Events are ingested using Event Ingestion
  - Event Consumers that listen for the events

1.15 Serverless and Event-driven Computing

- Events are delivered to consumers in near real-time so that they can respond immediately to the events as they happen.
- Event producers are decoupled from the event consumers.
- Serverless applications are usually built by combining multiple functions (FaaS) using offerings, such as AWS Lambda or Microsoft Azure Functions, together with external backend resources, such as Amazon S3, Amazon DynamoDB
- The architecture that ties these multiple functions (FaaS) is event-driven architecture.
- By combining tools, such as AWS Lambda and Amazon S3, you can develop applications without having to think about the provision and management of the infrastructure.

1.16 What is AWS Lambda?

- AWS Lambda lets you run your code written in a number of supported languages in a PaaS-like environment with zero administration on your part. There is no EC2 instance needed to run your code.
  - This arrangement is referred to as Serverless Computing or NoOps
- AWS provisions, scales, and manages servers needed to run your code, referred to as the *Lambda function*
  - You pay only for the actual time your Lambda function runs
    - You are charged in 0.1 second increments multiplied by the number of Lambda function invocations

  - ◊ You are only required to provide the amount of computing memory (the default is 128MB)
    - ✔ CPU is allocated in proportion to the requested memory
- ■ You write code as Lambda functions. They can:
  - ◊ Respond to changes (events) in other AWS services (sources of events), such as an S3 update or HTTP requests.
  - ◊ Interact with external resources like Amazon DynamoDB or other web services.

Notes:

Requesting 256MB of memory for your Lambda function allocates approximately twice as much computing power as in case of the default 128MB of memory.

1.17 Supported Languages

- ■ As of 2017, Lambda gives you a choice of the following languages:
  - ◊ Node.js (JavaScript)
  - ◊ Python 2.7
  - ◊ Java 8
    - ✔ Lambda provides the Amazon Linux build of openjdk 1.8
  - ◊ C#
    - ✔ Distributed as a NuGet package with the “dotnetcore1.0” runtime parameter
- ■ Lambda functions can used dependent libraries, including native ones
  - ◊ There is no white-list APIs

- Your code can spawn additional threads, if needed
- **Note**: Disabled activities are mentioned in the Slide Notes

Notes:

According to Lambda documentation:

"*There are a few activities that are disabled: Inbound network connections are blocked by AWS Lambda, and for outbound connections only TCP/IP sockets are supported, and ptrace (debugging) system calls are restricted. TCP port 25 traffic is also restricted as an anti-spam measure*"

1.18 Getting Your Code Up And Running in Lambda

- You have three options:
  1. Create your Lambda function code inside the AWS Management Console (the in-line option)
     - ✔ Only available for Node.js and Python runtimes
  2. Develop code locally, build a ZIP or JAR bundle with all the dependencies and upload it to AWS
  3. Upload your ZIP or JAR to S3

Notes:

According to Lambda documentation:

"*Uploads must be no larger than 50MB (compressed). You can use the AWS Eclipse plugin to author and deploy Lambda functions in Java. You can use the Visual Studio plugin to author and deploy Lambda functions in C#, and Node.js.*

...

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more.

For storing sensitive information, we recommend encrypting values using KMS and the console's encryption helpers.

1.19 Examples of the Base Lambda Function

Target Runtime	Lambda Function Code

Node.js (ECMAScript 2015)	`exports.handler = (event, context, callback) => {`
	`    callback(null, 'Hello from Lambda');`
	`};`

Python	`def lambda_handler(event, context):`
	`    return 'Hello from Lambda'`

1.20 Use Cases

- High throughput real-time workflows handling billions of events per day
- Back-ends for you apps
- Media objects transcoding
- Real-time tracking of calls made to any Amazon Web Service from your app
- Front-end HTTP service
  - ◊ You can invoke your Lambda function via an HTTP endpoint using Amazon API Gateway

1.21 How It Works

- AWS Lambda functions run inside a default AWS-managed VPC on the computing infrastructure provided by AWS

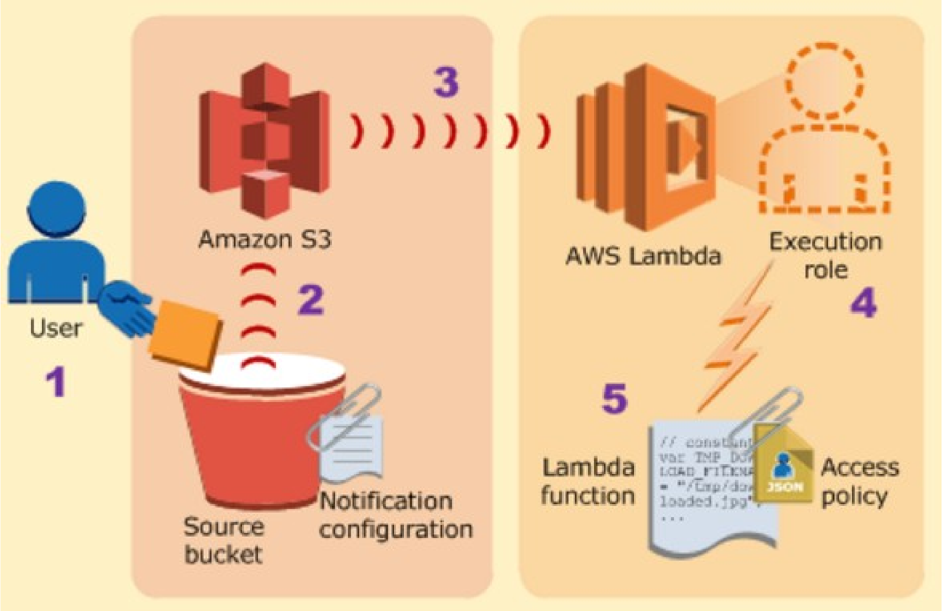
  - Optionally, you can configure Lambda to run within your custom VPC
- It is highly recommended you write your code in a “stateless” style
  - Any state you may want to retain beyond the lifetime of the request should be externalized to an AWS persistent store, e.g. Amazon S3, Amazon DynamoDB, etc.
- You can configure triggers for the Lambda, such as uploading a file on S3, that will cause the Lambda to be executed.
- You can also invoke lambda functions directly using the AWS SDKs or AWS Mobile SDKs, such as the AWS Mobile SDK for Android
- Lambda provides a great foundation for building microservices

Notes:

According to Lambda documentation:

"To improve performance, AWS Lambda may choose to retain an instance of your function and reuse it to serve a subsequent request, rather than creating a new copy. Your code should not assume that this will always happen."

1.22 Example: Processing S3 Source Events with Lambda



- Source: AWS Documentation

1.23 The Programming Model

- The following core concepts apply to Lambda functions created in any of the supported languages:
  - **Handler** - The user call-back function invoked by AWS Lambda in response to a registered event; this function is passed the event object and the context object
  - **The context object** - The AWS Lambda object that provides information of the call context
  - **Logging** - Any user Lambda function can contain language-specific logging statements output of which AWS redirects to CloudWatch Logs

Notes:

According to Lambda documentation:

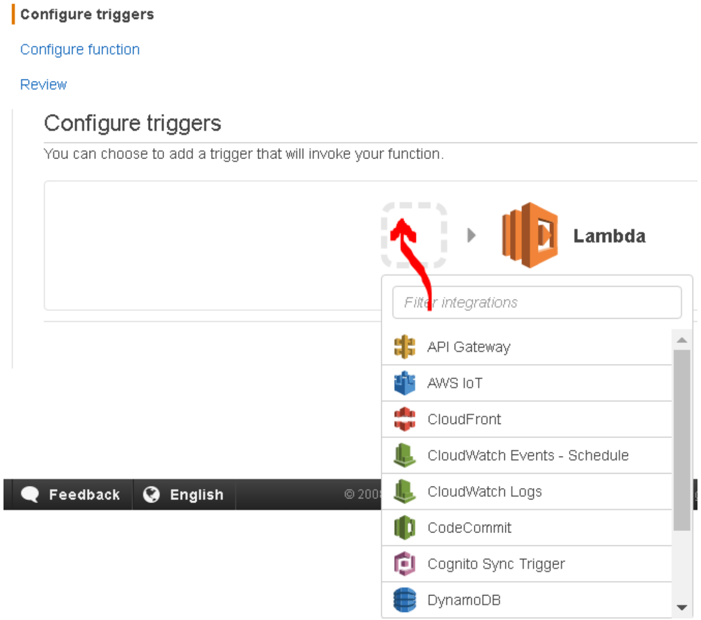
"*AWS Lambda provides this information via the context object:*

- *How much time is remaining before AWS Lambda terminates your Lambda function (timeout is one of the Lambda function configuration properties).*
- *The 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.*"

1.24 Configuring Lambda Functions

- The Lambda Dashboard of the AWS Management Console offers you a wizard-like Lambda function creation flow, where you need:
  - Specify the source of events to which your lambda function will be triggered to respond
  - Provide the function body written in the language of your choice
    - Write code in-line or upload the ZIP file containing your code
  - Specify the role under which your code will be running
  - Optionally, provide memory size (default is 128 MB), the call timeout (a value between the default 3 seconds and the maximum of 5 minutes), and configuration key-value pairs

1.25 Configure Triggers Page



Configure triggers

You can choose to add a trigger that will invoke your function.

1.26 Lambda Function Blueprints

- You have an option to either create a Lambda function from scratch, or use Lambda blueprints
- A Lambda blueprint is a sample configuration of an event source and a skeletal Lambda function for handling such events
- As of 2017, Lambda offers 78 blueprints, such as
  - ***dynamodb-process-stream***
    - *"An Amazon DynamoDB trigger that logs the updates made to a table"*
  - ***lex-book-trip-python***

    - ✔ "*Book details of a visit, using Amazon Lex to perform natural language understanding*"
  - ◊ ***microservice-http-endpoint***
    - ✔ "*A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway*"

1.27 How Do I Troubleshoot and Monitor My Lambda Functions?

- AWS Lambda automatically monitors your Lambda functions reporting the following real-time metrics through Amazon CloudWatch:
  - ◊ Invocation count
  - ◊ Invocation duration
  - ◊ Invocation errors
- You can use these metrics to set CloudWatch custom alarms
- The CloudWatch Logs group associated with your Lambda function can be accessed by this logical path: */aws/lambda/<your function name>*

1.28 Developing Lambda in Java

- You can create a Lambda in a plain Maven project or use the AWS Toolkit plugin for Eclipse. If you are using Maven, add dependency for com.amazonaws:aws-lambda-java-core.
- Develop POJO classes that will carry request and response data.
- Develop the Lambda class which must implement

```
public class MyLambda
```

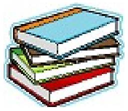
```
  implements RequestHandler<MyRequest, MyResponse> {

  public MyResponse handleRequest(MyRequest request, 
Context context) {
    //...
  }

}
```

- Build a JAR file that includes all the dependency classes and upload it in for your Lambda.

1.29 Summary



- In this chapter, we reviewed the AWS Lambda service that provides an excellent platform for building microservices in the Amazon Cloud.