AWS Lambda JAVA8 Training Course Online

Curriculum 5 Days

Day - 1

- **Understanding Serverless Functions**
  - Contextualizing Serverless
  - Key Elements of Serverless Functions
  - Looking at Serverless Function Providers
  - Demo Overviews - What Are You Building?

- **Working with AWS**
  - Introduction to the AWS Free Tier
  - AWS Free Tier Service Walkthrough
  - Overview of AWS Identity and Access Management (IAM)
  - Creating and Managing AWS Identity and Access Management Policies

Day - 2

- **Starting with Lambda Functions**
  - Considerations and Limitations for Lambda Functions
  - Lambda Prerequisites
  - Creating and Configuring Your First Lambda Function
  - Monitoring and Alerting for Your First Lambda Function

- **Using Lambda and Third Party APIs**
  - Planning Function Scope and Dependencies
  - Credential Storage with the AWS Key Management Service
  - Gathering API Keys and Preparing Your Environment
  - Working with External Libraries, Sensitive Credentials, and Your Lambda Function Package
  - Deploying Your Function Package and Configuring Your Twitter Bot

Day - 3

- **Lambda Expressions and Functional Interfaces**
  - Project and Resource Overview
  - Installing Jinja and Configuring IAM and SES
  - Uploading Templates to S3 and Creating Cloudwatch Events with the AWS Command Line
  - Creating a Dynamic Lambda Handler
  - Testing Your Lambda Function with the AWS Command Line
  - Understanding Function Package Setup
  - Function Deployment and Configuration with the AWS Command Line

- **Using ELB to Scale Applications**
  - Lambda Expression: Introduction, Instances of Anonymous Classes
  - Lambda Expression: Passing Code as a Parameter
  - Let Us Write Our First, Simple Lambda Expressions
  - Lambda Expression: Remarks and Precisions
  - Method References: A First Example with an Instance Method
  - Method References: A Second Example with a Static Method
  -

Day - 4

- **Writing Data Processing Functions with Lambdas in Java**
  - Introduction to the Module
  - What Is a Functional Interface? The Predicate Example
  - How to Implement a Functional Interface with a Lambda Expression
  - How Does the Compiler Recognize the Type of a Lambda Expression?
  - A Lambda Is Still an Interface with Usable Methods
  - Functional Interface: The Complete and Exact Definition
  - How to Use the @FunctionalInterface Annotation
  - The Four Categories of the java.util.function Package
  - First Category: The Consumers
  - Second Category: The Supplier
  - Third Category: The Functions
  - Fourth Category: The Predicates
  - Functional Interfaces for Java Primitive Types
  - Introduction to the Live Coding Section: The Predicate Example
  - Writing and Using a First, Simple Predicate Lambda Expression
  - Chaining Predicates with the AND Boolean Operation
  - Adding a and() Method on the Predicate Functional Interface
  - Implementing the and() Default method on the Predicate Interface
  - Adding a or() Default Method on the Predicate Interface
  - Creating Predicates with a Static Call on a Functional Interface
  - Making the isEqualsTo() Method Generic of the Predicate Interface

- **Data Processing Using Lambdas and the Collection Framework**
  - Introduction to the Module
  - First Methods on Iterable, Collection and List
  - First Method on Map: forEach()
  - More Methods on Map: getOrDefault()
  - More Methods on Map: putIfAbsent()
  - More Methods on Map: replace() and replaceAll()
  - New Pattern on Map: remove()
  - New Patterns on Map: The compute() method
  - New Patterns on Map: computeIfAbsent(), computeIfPresent()
  - Building Maps of Maps and Maps of Lists with computeIfAbsent()
  - New Pattern on Map: The merge() method
  - Using merge() to Merge Two Maps Together
  - Live Coding Session Introduction, forEach() in Action
  - Methods removeIf(), replaceAll(), sort() in Action
  - Setting Default Value for map.get(): getOrDefault()
  - Adding Default key / value pairs: putIfAbsent, computeIfAbsent
  - Merging Maps with the map.merge() Method
  - Merging Maps: Analysis of the Result
  - Live Coding and Module Wrap-up

Day - 5

- **Implementing Map Filter Reduce Using Lambdas and Collections**
  - Introduction to the Module
  - Computing the Average of People Older than 20, Taken From a List
  - Map / filter / reduce: A Precise Explanation
  - A First Implementation, in the JDK7 Way
  - A Closer Look at the Reduction Step: How Does it Work?
  - Parallel Implementation of the Reduction Step
  - First Caveat: Non-associative Reduction Operations
  - How to Detect Non-associative Reduction Operations
  - Second Caveat: Reduction of a Singleton
  - Second Caveat: Reduction of a Set with Several Elements
  - Second Caveat: Reduction That Do Not Have Identity Element
  - Live Coding: Setting up the Environment
  - Simulating Parallel Computation of a Non-associative Reduction
  - Non-associative Reduction: The Average Reduction Operation
  - Computing a Max: Reduction with No Identity Element
  - Live Coding Wrap-up
  - Using Optional to Handle Reductions with No Identity Element
  - Wrap-up on the Reduction Step
  - Implementation in the JDK7 Way: a Closer Look
  - CPU Load and Memory Footprint Evaluations
  - Example of an all Match Reduction Operation: Lost Optimizations
  - Why is this First, Naive Implementation Should be Avoided
  - A First Glimpse at the Stream API

- **The Stream API, How to Build Streams, First Patterns**
  - Introduction to the Module
  - A First Technical Definition of the Stream Interface
  - First Definitions of the Concept of Stream
  - The Notion of Unbounded Stream
  - How to Build Streams: Empty Streams, Singletons, varargs
  - How to Build Streams: The Generator and Iterator Pattern
  - How to Build Streams on Strings, Regular Expressions, and Text Files
  - The Stream.Builder Pattern
  - The map / filter / reduce Pattern Written with a Stream
  - A Second Example of the ap / filter / reduce Pattern on Streams
  - Intermediate and Terminal Calls on Streams: peek() and forEach()
  - How to Tell an Intermediate Call from a Terminal Call
  - Selecting Ranges of Data in Streams: skip() and limit()
  - Simple Reductions: Matchers, Short-circuiting Reductions
  - Finder Reductions, Use of Optional
  - Example of Finder Reductions: find First(), find Any()
  - General Reductions: Use of the reduce () Method
  - Live Coding Session Introduction
  - Example of a First Simple Stream Built on a vararg
  - Building a Stream: The Generate Pattern, Use of Limit()
  - Building a Stream: The Iterate Pattern
  - Building Streams of Random Numbers Using Random.ints()
  - Live Coding Session Wrap-up