AWS Lambda

Serverless Computing Platform

Run Code Without Managing Servers

Event-Driven • Scalable • Cost-Effective

Prepared By: Rashi Rana
AWS Corporate Trainer

What is AWS Lambda?

AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you.

Key Characteristics

🚀 Serverless

No servers to manage or provision

⚡ Event-Driven

Executes code in response to triggers

📈 Auto-Scaling

Scales automatically from zero to thousands

💰 Pay-per-Use

Pay only for compute time consumed

How Lambda Works

Event Source → Lambda Function → Response/Action

1. Event triggers function | 2. Lambda executes code | 3. Function returns response

Prepared By: Rashi Rana
AWS Corporate Trainer

Understanding Serverless Architecture

What is Serverless?

Serverless computing allows you to build and run applications without thinking about servers. You still have servers, but AWS manages them for you.

Traditional vs Serverless

Aspect	Traditional	Serverless
Server Management	Manual provisioning & maintenance	Fully managed by AWS
Scaling	Manual or auto-scaling groups	Automatic and instant
Pricing	Pay for running instances	Pay per execution
Availability	Configure high availability	Built-in high availability

Benefits of Serverless Architecture

- **No Server Management:** Focus on code, not infrastructure
- **Automatic Scaling:** Handles traffic spikes seamlessly
- **Cost Optimization:** Pay only when code runs
- **Built-in Availability:** Multi-AZ deployment by default
- **Faster Time to Market:** Deploy code quickly without setup

Prepared By: Rashi Rana
AWS Corporate Trainer

Event-Driven Programming

Event-driven programming is a paradigm where the flow of program execution is determined by events such as user actions, sensor outputs, or messages from other programs.

Lambda Event Model

🎯 Event Sources

- AWS Services (S3, DynamoDB, etc.)
- HTTP requests via API Gateway
- Scheduled events (CloudWatch Events)
- Custom applications

⚙️ Function Execution

- Lambda receives event
- Creates execution environment
- Runs your code
- Returns response

Event Structure Example

```
{
  "Records": [
    {
      "eventVersion": "2.1",
      "eventSource": "aws:s3",
      "eventName": "ObjectCreated:Put",
      "s3": {
        "bucket": {
          "name": "my-bucket"
        },
        "object": {
          "key": "uploaded-file.jpg"
        }
      }
    }
  ]
}
```

💡 Key Advantage

Event-driven architecture enables loose coupling between components, making systems more resilient and easier to maintain.

Prepared By: Rashi Rana
AWS Corporate Trainer

Statelessness & Execution Duration

Stateless Functions

> **Lambda functions are stateless** - each invocation is independent and doesn't retain data from previous executions.

Stateless Characteristics

🔄 Independent Execution

Each function invocation runs in isolation

💾 No Persistent Storage

Local storage is temporary (/tmp directory)

🔗 External State

Use external services for persistent data

⚡ Fast Startup

No state to restore enables quick scaling

Execution Duration Limits

Aspect	Limit	Notes
Maximum Duration	15 minutes	Configurable timeout (1 sec - 15 min)
Memory	128 MB - 10,240 MB	CPU scales with memory allocation
Temporary Storage	512 MB - 10,240 MB	/tmp directory, ephemeral
Concurrent Executions	1,000 (default)	Can request increase

⚠️ Important Considerations

- Design functions to complete within timeout limits
- Use external storage for data persistence
- Consider connection pooling for database connections
- Implement proper error handling and retries

Prepared By: Rashi Rana
AWS Corporate Trainer

Creating Lambda Functions

Supported Languages & Runtimes

🐍 Python

3.8, 3.9, 3.10, 3.11, 3.12

☕ Java

8, 11, 17, 21 (Corretto)

🟨 Node.js

16.x, 18.x, 20.x

🔷 .NET

6, 8 (Core)

🦀 Rust

Custom runtime

🐹 Go

1.x (custom runtime)

Function Creation Methods

- **AWS Console:** Web-based interface with inline editor
- **AWS CLI:** Command-line interface for automation

- **AWS SDKs:** Programmatic creation using various languages
- **Infrastructure as Code:** CloudFormation, CDK, Terraform
- **IDE Plugins:** VS Code, IntelliJ, Eclipse

> 💡 **Best Practice**
>
> Use Infrastructure as Code (IaC) for production deployments to ensure consistency and version control.

Prepared By: Rashi Rana
AWS Corporate Trainer

Function Configuration Settings

Basic Configuration

Setting	Description	Default/Range
Function Name	Unique identifier for the function	1-64 characters
Runtime	Programming language and version	Various options
Handler	Entry point for function execution	filename.method_name
Memory	Memory allocated to function	128 MB - 10,240 MB
Timeout	Maximum execution time	1 sec - 15 min

Advanced Configuration

🔐 Execution Role

🌐 VPC Settings

IAM role with permissions for AWS services

Connect to resources in private subnets

🔧 Environment Variables

Configuration values for your function

📊 Dead Letter Queue

Handle failed function invocations

Sample Python Function

```
import json

def lambda_handler(event, context):
    # Process the event
    name = event.get('name', 'World')

    # Return response
    return {
        'statusCode': 200,
        'body': json.dumps({
            'message': f'Hello {name}!',
            'requestId': context.aws_request_id
        })
    }
```

Prepared By: Rashi Rana
AWS Corporate Trainer

Introduction to Boto3

Boto3 is the AWS SDK for Python, allowing you to create, configure, and manage AWS services from your Lambda functions.

What is Boto3?

- **AWS SDK for Python:** Official Python library for AWS services
- **Pre-installed in Lambda:** Available in all Python runtimes
- **Resource & Client APIs:** High-level and low-level interfaces
- **Automatic Authentication:** Uses Lambda execution role

Boto3 in Lambda Functions

🔧 Client Interface

- Low-level service access
- Direct API mapping
- Fine-grained control

📦 Resource Interface

- High-level abstraction
- Object-oriented approach
- Easier to use

🔐 Authentication

⚡ Performance

- Uses execution role
- No credentials needed
- Automatic permissions

- Initialize outside handler
- Reuse connections
- Connection pooling

Common Boto3 Usage Examples

```
import boto3
import json

# Initialize clients outside handler for reuse
s3_client = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')

def lambda_handler(event, context):
    # S3 operations
    response = s3_client.get_object(
        Bucket='my-bucket',
        Key='data.json'
    )

    # DynamoDB operations
    table = dynamodb.Table('my-table')
    table.put_item(
        Item={
            'id': '123',
            'data': json.loads(response['Body'].read())
        }
    )

    return {'statusCode': 200}
```

Best Practices with Boto3

- **Initialize Outside Handler:** Reuse client connections
- **Use Appropriate Interface:** Client for control, Resource for simplicity
- **Handle Exceptions:** Catch boto3 specific errors
- **Configure Retries:** Set retry policies for resilience

Prepared By: Rashi Rana
AWS Corporate Trainer

Triggers & Event Sources

Event Sources are AWS services or custom applications that trigger Lambda function execution.

Common AWS Service Triggers

🪣 Amazon S3

- Object creation/deletion
- Object restore from Glacier
- Replication events

🗄️ DynamoDB

- Stream records
- Item modifications
- Table updates

📫 Amazon SQS

- Message polling
- Batch processing
- Dead letter queues

📣 Amazon SNS

- Topic notifications
- Message filtering
- Cross-region messaging

API Gateway Integration

Client → API Gateway → Lambda Function → Response

REST APIs | HTTP APIs | WebSocket APIs

Event Source Mapping

Service	Invocation Type	Batch Size
S3	Asynchronous	Single event
DynamoDB Streams	Synchronous	1-1000 records
SQS	Synchronous	1-10 messages
API Gateway	Synchronous	Single request

Prepared By: Rashi Rana
AWS Corporate Trainer

Working with Lambda

Function Code Management

Deployment Methods

📝 Inline Editor

- Direct editing in AWS Console
- Good for simple functions
- Limited to 3MB code size

📦 ZIP Upload

- Upload code as ZIP file
- Up to 50MB direct upload
- 250MB via S3

🐋 Container Images

- Deploy as container image
- Up to 10GB image size
- Use ECR for storage

🔗 S3 Reference

- Store code in S3 bucket
- Reference S3 object
- Good for large packages

Version Management

Feature	Description	Use Case
Versions	Immutable snapshots of function	Release management
Aliases	Pointers to specific versions	Environment management
$LATEST	Always points to latest code	Development/testing

Prepared By: Rashi Rana
AWS Corporate Trainer

Lambda Layers

> **Lambda Layers** allow you to package libraries, custom runtimes, or other function dependencies separately from your function code.

Benefits of Layers

- **Code Reuse:** Share common code across multiple functions
- **Smaller Packages:** Reduce deployment package size
- **Separation of Concerns:** Keep business logic separate from dependencies
- **Version Management:** Update dependencies independently

Layer Structure

```
layer.zip
├── python/          # For Python runtime
│   └── lib/
│       └── requests/
├── nodejs/          # For Node.js runtime
│   └── node_modules/
└── bin/             # For executables
```

Layer Configuration

📚 Library Layer

Common libraries like requests, pandas, etc.

🔧 Runtime Layer

Custom runtime environments

⚙️ Configuration Layer

Configuration files and utilities

🛠️ Tool Layer

Command-line tools and binaries

💡 Best Practice

Use layers for dependencies that don't change frequently. Keep your function code lightweight and focused on business logic.

Prepared By: Rashi Rana
AWS Corporate Trainer

Testing & Error Handling

Testing Lambda Functions

Testing Methods

🧪 Console Testing

- Built-in test events
- Custom test events
- Real-time execution logs

🔧 Local Testing

- SAM CLI local invoke
- Docker containers
- Unit testing frameworks

🚀 Integration Testing

- End-to-end testing
- Event source simulation
- Performance testing

📊 Load Testing

- Concurrent execution testing
- Scaling behavior
- Performance metrics

Error Handling Strategies

```
import json
import logging
```

```
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        # Process event
        result = process_data(event)

        return {
            'statusCode': 200,
            'body': json.dumps(result)
        }
    except ValueError as e:
        logger.error(f"Validation error: {str(e)}")
        return {
            'statusCode': 400,
            'body': json.dumps({'error': 'Invalid input'})
        }
    except Exception as e:
        logger.error(f"Unexpected error: {str(e)}")
        raise  # Re-raise for retry mechanism
```

Retry Mechanisms

- **Synchronous:** Client handles retries
- **Asynchronous:** Lambda retries automatically (0-2 times)
- **Stream-based:** Retries until success or record expires
- **Dead Letter Queues:** Handle failed invocations

Prepared By: Rashi Rana
AWS Corporate Trainer

Logging & Monitoring

CloudWatch Integration

CloudWatch provides comprehensive logging and monitoring capabilities for Lambda functions out of the box.

CloudWatch Logs

📝 Automatic Logging

- Function execution logs
- Error messages and stack traces
- Custom log messages

🔍 Log Groups

- /aws/lambda/function-name
- Retention period configuration
- Log streaming and filtering

🔎 Log Insights

- Query and analyze logs
- Performance analysis

📊 Structured Logging

- JSON formatted logs
- Searchable fields

- Error pattern detection

- Better analytics

CloudWatch Metrics

Metric	Description	Use Case
Invocations	Number of function executions	Usage tracking
Duration	Execution time in milliseconds	Performance monitoring
Errors	Number of failed invocations	Error rate monitoring
Throttles	Number of throttled invocations	Concurrency monitoring
Dead Letter Errors	Failed async invocations	Failure handling

Custom Metrics Example

```
import boto3
import json

cloudwatch = boto3.client('cloudwatch')

def lambda_handler(event, context):
    try:
        # Your business logic here
        result = process_data(event)
```

```
        # Custom metric
        cloudwatch.put_metric_data(
            Namespace='MyApp/Lambda',
            MetricData=[
                {
                    'MetricName': 'ProcessedRecords',
                    'Value': len(result),
                    'Unit': 'Count'
                }
            ]
        )

        return {'statusCode': 200, 'body':
json.dumps(result)}
    except Exception as e:
        # Error metric
        cloudwatch.put_metric_data(
            Namespace='MyApp/Lambda',
            MetricData=[
                {
                    'MetricName': 'ProcessingErrors',
                    'Value': 1,
                    'Unit': 'Count'
                }
            ]
        )
        raise
```

Prepared By: Rashi Rana
AWS Corporate Trainer

Lambda Best Practices

Performance Optimization

⚡ Cold Start Optimization

- Minimize package size
- Use provisioned concurrency
- Optimize initialization code

💾 Memory Configuration

- Right-size memory allocation
- More memory = more CPU
- Monitor and adjust

🔄 Connection Reuse

- Initialize outside handler
- Reuse database connections
- Cache configuration data

📦 Deployment Packages

- Use layers for dependencies
- Remove unnecessary files
- Compress efficiently

Security Best Practices

- **Least Privilege:** Grant minimal required permissions

- **Environment Variables:** Use for configuration, encrypt sensitive data
- **VPC Configuration:** Use when accessing private resources
- **Input Validation:** Always validate and sanitize inputs
- **Secrets Management:** Use AWS Secrets Manager or Parameter Store

Cost Optimization

💰 Cost Factors

- **Requests:** $0.20 per 1M requests
- **Duration:** $0.0000166667 per GB-second
- **Data Transfer:** Standard AWS rates apply

Tip: Monitor usage with AWS Cost Explorer and set up billing alerts.

Prepared By: Rashi Rana
AWS Corporate Trainer

Summary

Key Takeaways

- **Serverless Computing:** Focus on code, not infrastructure
- **Event-Driven:** Responds to triggers from various AWS services
- **Stateless Functions:** Each execution is independent
- **Multiple Runtimes:** Support for popular programming languages
- **Automatic Scaling:** Handles traffic spikes seamlessly
- **Integrated Monitoring:** Built-in CloudWatch logging and metrics

Next Steps

🧪 Hands-on Practice

- Create your first Lambda function
- Set up event triggers
- Configure monitoring and alerts

📚 Advanced Topics

- Lambda@Edge for CDN computing
- Step Functions for workflows
- Lambda Extensions

- Test error handling scenarios

- Performance optimization techniques

💡 Remember

Lambda is designed for short-running, stateless functions. It's perfect for microservices, data processing, and event-driven architectures. Start small, monitor performance, and scale as needed.

Thank You!

Questions & Discussion

Prepared By: Rashi Rana
AWS Corporate Trainer