AWS Developer Training - .NET

Advanced Development and Security

Abdul Rasheed Feroz Khan
Director - CodeSizzler India | UAE
www.codesizzler.in

Agenda – Day 2

Advanced Development and Security

Session 1: Serverless & Event-Driven .NET (3 hrs)

- Amazon SQS & SNS – Messaging in .NET
- EventBridge – Event-driven architectures
- Step Functions – Workflow automation with .NET

Hands-on Lab:

- Create a .NET event-driven app using SQS & Lambda

Session 2: Authentication & Security (2 hrs)

- Amazon Cognito – User auth in .NET apps
- Secrets Manager & Parameter Store – Managing .NET app secrets
- KMS & Encryption SDK – Data encryption in C#

Session 3: CI/CD for .NET on AWS (3 hrs)

- AWS CodeCommit, CodeBuild, CodeDeploy, CodePipeline
  - Building & deploying .NET apps
- Infrastructure as Code (AWS CDK in C#)

Hands-on Lab:

- Set up a CI/CD pipeline for a .NET Core app

Serverless & Event-Driven .NET

Session 1: Serverless & Event-Driven .NET

Building Scalable & Decoupled Applications on AWS

- Amazon SQS & SNS – Messaging in .NET
- Amazon EventBridge – Event-driven architectures
- AWS Step Functions – Workflow automation with .NET

Why Serverless & Event-Driven?

The Modern Application Paradigm

- **Scalability:** Automatically scale with demand
- **Cost Efficiency:** Pay-per-use pricing model
- **Resilience:** Built-in fault tolerance
- **Developer Productivity:** Focus on business logic, not infrastructure
- **Loose Coupling:** Services interact through events, not direct calls

.NET in Serverless: Perfect fit with AWS Lambda, container support, and native SDKs

Workshop Prerequisites & Setup

AWS Account
(Free Tier eligible)

AWS CLI
configured

.NET 6+ SDK
installed

IDE (Visual Studio,
VS Code, or Rider)

AWS Toolkit for
your IDE

Part 1: Amazon SQS & SNS – Messaging in .NET

Learning Objectives:

Understand messaging patterns in distributed systems

Implement SQS queues in .NET applications

Use SNS for pub/sub messaging

Handle errors and retries effectively

Messaging Fundamentals

amazon SQS

Fully managed message queuing service

Decouples application components

Supports standard and FIFO queues

Use Case: Async task processing, buffer between services

Messaging Fundamentals

Pub/Sub messaging service

Fan-out messages to multiple subscribers

Supports multiple protocols (HTTP, SMS, Email, etc.)

Use Case: Event notifications, broadcast messages

SQS with .NET - Implementation

Installing the SDK

```bash

dotnet add package AWSSDK.SQS
dotnet add package AWSSDK.Extensions.NETCore.Setup
```

Configuration in Program.cs

```csharp

builder.Services.AddAWSService<IAmazonSQS>();
builder.Services.AddDefaultAWSOptions(
    new AWSOptions { Region = RegionEndpoint.USWest2 });
```

SQS Producer Example

```
public class OrderProcessingService
{
    private readonly IAmazonSQS _sqsClient;
    private readonly string _queueUrl;

    public async Task SendOrderMessage(Order order)
    {
        var message = new SendMessageRequest
        {
            QueueUrl = _queueUrl,
            MessageBody = JsonSerializer.Serialize(order),
            MessageAttributes = new Dictionary<string, MessageAttributeValue>
            {
                {"MessageType", new MessageAttributeValue
                    { DataType = "String", StringValue = "Order" }}
            }
        };

        await _sqsClient.SendMessageAsync(message);
    }
}
```

SQS Consumer Example

```
public class OrderProcessor
{
    public async Task ProcessMessages()
    {
        var receiveRequest = new ReceiveMessageRequest
        {
            QueueUrl = _queueUrl,
            MaxNumberOfMessages = 10,
            WaitTimeSeconds = 20 // Long polling
        };

        var response = await _sqsClient.ReceiveMessageAsync(receiveRequest);

        foreach (var message in response.Messages)
        {
            var order = JsonSerializer.Deserialize<Order>(message.Body);
            await ProcessOrder(order);
            await _sqsClient.DeleteMessageAsync(_queueUrl, message.ReceiptHandle);
        }
    }
}
```

SNS with .NET - Implementation

```
// Configure SNS
builder.Services.AddAWSService<IAmazonSimpleNotificationService>();

// Publish message
var snsClient = new AmazonSimpleNotificationServiceClient();
var publishRequest = new PublishRequest
{
    TopicArn = "arn:aws:sns:us-west-2:123456789012:orders-topic",
    Message = JsonSerializer.Serialize(order),
    MessageAttributes = new Dictionary<string, MessageAttributeValue>
    {
        {"eventType", new MessageAttributeValue
            { DataType = "String", StringValue = "OrderCreated" }}
    }
};

await snsClient.PublishAsync(publishRequest);
```

Amazon EventBridge

Event-Driven Architecture Foundation

Serverless event bus service

Connect AWS services, custom apps, and SaaS applications

Schema discovery and validation

Event filtering and transformation

EventBridge Concepts

Key Components:

- **Event Buses:** Central routing for events
- **Rules:** Determine where events are sent
- **Targets:** Services that process events (Lambda, SQS, SNS, etc.)
- **Schemas:** Structure of events (Registry)

Event Pattern

```
{
  "source": ["my.order.service"],
  "detail-type": ["OrderCreated"],
  "detail": {
    "status": ["pending"]
  }
}
```

EventBridge with .NET

Sending Events from .NET

```
var eventBridgeClient = new AmazonEventBridgeClient();

var putEventRequest = new PutEventsRequest
{
    Entries = new List<PutEventsRequestEntry>
    {
        new PutEventsRequestEntry
        {
            Source = "my.order.service",
            DetailType = "OrderCreated",
            Detail = JsonSerializer.Serialize(order),
            EventBusName = "default"
        }
    }
};

await eventBridgeClient.PutEventsAsync(putEventRequest);
```

Event Processing by Lambda

Lambda Function Triggered by EventBridge

```
[LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSerial
izer))]
public async Task FunctionHandler(CloudWatchEvent<OrderEvent> orderEvent, ILambdaContext co
ntext)
{
    var order = orderEvent.Detail;

    switch (orderEvent.DetailType)
    {
        case "OrderCreated":
            await ProcessNewOrder(order);
            break;
        case "OrderCancelled":
            await CancelOrder(order);
            break;
    }
}
```

Schema Registry & Code Generation

Benefits:

- Discover event schemas from actual traffic
- Generate code bindings for type-safe events
- Validate events against schemas

Generating .NET Classes:

```
aws schemas get-code-binding-source
  --schema-arn arn:aws:schemas:us-west-2:123456789012:schema/OrderEvent
  --language CSharp
```

AWS Step Functions

AWS Step Functions

Coordinate multiple AWS services

Built-in error handling and retries

Visual workflow designer

Support for long-running processes (up to 1 year)

Step Functions Concepts

State Machine Types:

- **Standard:** High-volume, event-driven workflows
- **Express:** High-volume, short-duration workflows

State Types:

- **Task:** Single unit of work
- **Choice:** Make decisions based on input
- **Parallel:** Run multiple branches in parallel
- **Wait:** Delay execution for specific time

Step Functions with .NET Lambda

Lambda Task Implementation

```
public class OrderProcessingFunctions
{
    public async Task<Order> ProcessPayment(Order order, ILambdaContext context)
    {
        // Process payment logic
        order.Status = "payment_processed";
        return order;
    }

    public async Task<Order> ShipOrder(Order order, ILambdaContext context)
    {
        // Shipping logic
        order.Status = "shipped";
        return order;
    }
}
```

Defining State Machine (ASL)

Amazon States Language JSON

```
{
  "Comment": "Order Processing Workflow",
  "StartAt": "ProcessPayment",
  "States": {
    "ProcessPayment": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:ProcessPayment",
      "Next": "ShipOrder"
    },
    "ShipOrder": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-west-2:123456789012:function:ShipOrder",
      "End": true
    }
  }
}
```

Error Handling in Step Functions

Built-in Retry and Catch Mechanisms

```
"ProcessPayment": {
  "Type": "Task",
  "Resource": "arn:aws:lambda:us-west-2:123456789012:function:ProcessPayment",
  "Retry": [
    {
      "ErrorEquals": ["Lambda.ServiceException", "Lambda.AWSLambdaException"],
      "IntervalSeconds": 2,
      "MaxAttempts": 6,
      "BackoffRate": 2
    }
  ],
  "Catch": [
    {
      "ErrorEquals": ["States.ALL"],
      "Next": "PaymentFailed"
    }
  ],
  "Next": "ShipOrder"
}
```

Calling Step Functions from .NET

```
var sfClient = new AmazonStepFunctionsClient();

var startRequest = new StartExecutionRequest
{
    StateMachineArn = "arn:aws:states:us-west-2:123456789012:stateMachine:OrderWorkflow",
    Input = JsonSerializer.Serialize(new { OrderId = 123, Amount = 99.99 })
};

var response = await sfClient.StartExecutionAsync(startRequest);
var executionArn = response.ExecutionArn;

// Check execution status
var describeRequest = new DescribeExecutionRequest
{
    ExecutionArn = executionArn
};
var status = await sfClient.DescribeExecutionAsync(describeRequest);
```

Express Workflows for High Volume

Perfect for:

- High-volume data processing
- Real-time stream processing
- IoT data ingestion
- Microservices coordination

.NET Integration:

- Sync execution support
- Sub-second duration workflows
- Perfect for API-driven workflows

Monitoring & Debugging

AWS Tools:

- **CloudWatch:** Logs and metrics
- **X-Ray:** Distributed tracing
- **Step Functions:** Visual workflow debugger
- **EventBridge:** Schema validation and monitoring

.NET Specific:

- Serilog/CloudWatch logging
- ASP.NET Core health checks
- Custom metrics with Amazon.CloudWatch

Lab

Questions

Authentication & Security

Session 2: Authentication & Security

Securing Your .NET Applications on AWS

- Amazon Cognito – User Authentication in .NET Apps
- Secrets Manager & Parameter Store – Managing App Secrets
- KMS & Encryption SDL – Data Encryption in C#

The Security Shared Responsibility Model

AWS Responsibility: Security OF the Cloud	Your Responsibility: Security IN the Cloud
Infrastructure security	Application security and data protection
Hardware and software maintenance	Identity and access management
Region and availability zone operations	Encryption configuration
	.NET application security practices

Part 1: Amazon Cognito – User Auth in .NET Apps

Learning Objectives:

- Understand Cognito User Pools and Identity Pools
- Implement authentication in ASP.NET Core applications
- Secure API endpoints with JWT tokens
- Manage user lifecycle in .NET apps

Amazon Cognito Overview

User Pools: User directory and authentication service

Sign-up, sign-in, profile management

Social identity providers (Google, Facebook, etc.)

JWT token generation

Identity Pools: AWS credentials federation

Grant temporary AWS credentials

Access AWS services securely

Setting Up Cognito in .NET

Install Required NuGet Packages

```bash

dotnet add package Amazon.Extensions.CognitoAuthentication
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
dotnet add package AWS.AspNetCore.Identity.Cognito
```

Setting Up Cognito in .NET

Configuration in Program.cs

```
builder.Services.AddCognitoIdentity();
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    options.Authority = "https://cognito-idp.us-west-2.amazonaws.com/your-user-pool-id";
    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidIssuer = "https://cognito-idp.us-west-2.amazonaws.com/your-user-pool-id",
        ValidateAudience = false,
        ValidateLifetime = true
    };
});
```

User Registration with Cognito

ASP.NET Core Controller Example

```
[ApiController]
[Route("api/auth")]
public class AuthController : ControllerBase
{
    private readonly CognitoUserPool _userPool;

    public AuthController(CognitoUserPool userPool)
    {
        _userPool = userPool;
    }

    [HttpPost("register")]
    public async Task<IActionResult> Register(RegisterRequest request)
    {
        var user = _userPool.GetUser(request.Email);
        var userAttributes = new Dictionary<string, string>
        {
            { "email", request.Email },
            { "given_name", request.FirstName },
            { "family_name", request.LastName }
        };

        var result = await user.SignUpAsync(request.Password, userAttributes);

        return Ok(new { UserId = result.UserID });
    }
}
```

User Authentication with Cognito

```
[HttpPost("login")]
public async Task<IActionResult> Login(LoginRequest request)
{
    var user = _userPool.GetUser(request.Email);
    var authRequest = new InitiateSrpAuthRequest
    {
        Password = request.Password
    };

    var authResponse = await user.StartWithSrpAuthAsync(authRequest);

    return Ok(new {
        AccessToken = authResponse.AuthenticationResult.AccessToken,
        RefreshToken = authResponse.AuthenticationResult.RefreshToken,
        IdToken = authResponse.AuthenticationResult.IdToken
    });
}
```

Part 2: Secrets Manager & Parameter Store

Secure Secret Management for .NET Apps

- **Secrets Manager:** For sensitive data (API keys, database passwords)
- **Parameter Store:** For configuration data (non-sensitive)
- **IAM-based access control**
- **Automatic rotation for Secrets Manager**

When to Use Each Service

AWS Secrets Manager

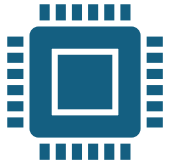
- Database credentials (RDS, DocumentDB)
- API keys and third-party secrets
- Any data that requires rotation
- Binary secrets up to 10KB

AWS Systems Manager Parameter Store

- Application configuration
- License codes
- Non-sensitive parameters
- Strings up to 8KB (Standard) or 4KB (Advanced)

Session 3: CI/CD for .NET on AWS

Automating Your Deployment Pipeline



AWS DevOps Services: CodeCommit, CodeBuild, CodeDeploy, CodePipeline

Building & Deploying .NET Applications Automatically

Infrastructure as Code with AWS CDK in C#

Why CI/CD for .NET?

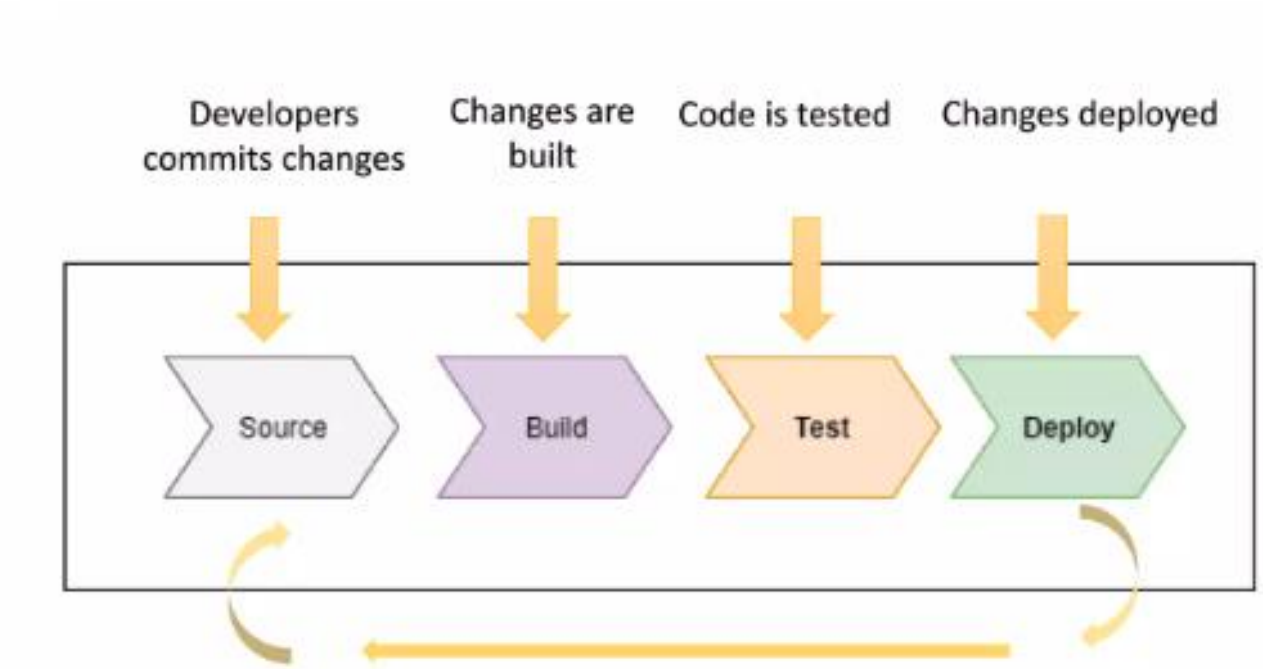
The Modern Development Cycle

- **Faster Time to Market:** Automated deployments
- **Higher Quality:** Automated testing and validation
- **Reduced Risk:** Consistent, repeatable processes
- **Developer Productivity:** Focus on code, not deployment

.NET on AWS:

- Native support for .NET in AWS services
- C# CDK for infrastructure as code
- Optimized build environments for .NET

CI/CD


Developers commits changes
Changes are built
Code is tested
Changes deployed
Source
Build
Test
Deploy

Areas that can be automated

- Version control
- Build process
- Testing
- Configuration management
- Infrastructure provisioning
- Deployment & release

AWS DevOps Toolchain Overview

CODECOMMIT: GIT-BASED SOURCE CONTROL

CODEBUILD: MANAGED BUILD SERVICE

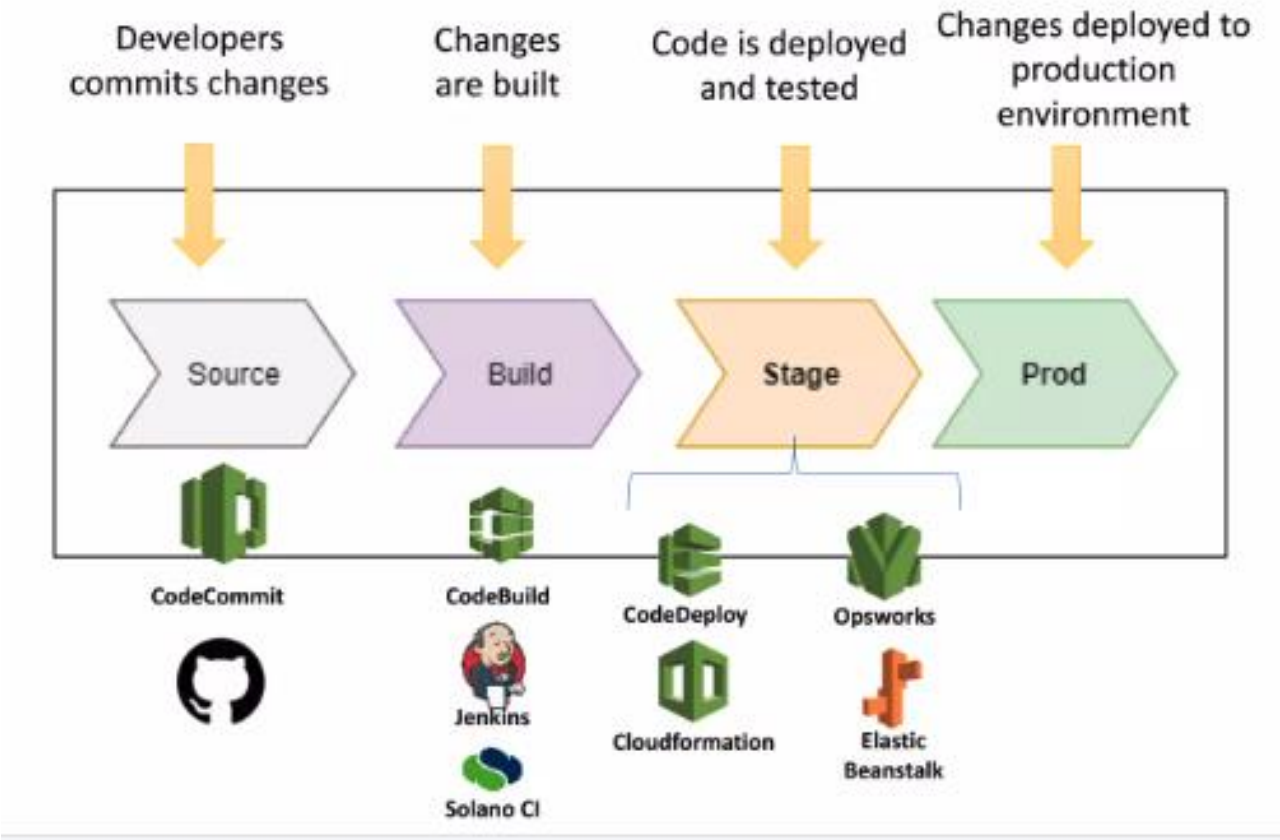
CODEDEPLOY: AUTOMATED DEPLOYMENTS

CODEPIPELINE: END-TO-END DELIVERY PIPELINE

AWS CDK: DEFINE CLOUD RESOURCES IN C#

CLOUDFORMATION: AWS DEPLOYMENT TEMPLATES

CI/CD on AWS


Developers commits changes
Changes are built
Code is deployed and tested
Changes deployed to production environment
Source
Build
Stage
Prod
CodeCommit
CodeBuild
CodeDeploy
Opsworks
Jenkins
Cloudformation
Elastic Beanstalk
Solano CI

Demo

Summary – Day 2

Advanced Development and Security

Session 1: Serverless & Event-Driven .NET (3 hrs)

- Amazon SQS & SNS – Messaging in .NET
- EventBridge – Event-driven architectures
- Step Functions – Workflow automation with .NET

Hands-on Lab:

- Create a .NET event-driven app using SQS & Lambda

Session 2: Authentication & Security (2 hrs)

- Amazon Cognito – User auth in .NET apps
- Secrets Manager & Parameter Store – Managing .NET app secrets
- KMS & Encryption SDK – Data encryption in C#

Hands-on Lab:

- Secure a .NET Web API with Cognito

Session 3: CI/CD for .NET on AWS (3 hrs)

- AWS CodeCommit, CodeBuild, CodeDeploy, CodePipeline
  - Building & deploying .NET apps
- Infrastructure as Code (AWS CDK in C#)

Hands-on Lab:

- Set up a CI/CD pipeline for a .NET Core app