# Cloud Fundamentals: AWS Services for C# Developers

## Course overview

## Getting started with AWS

## AWS SQS

### What is a queue?

The problem: A blackboard with white text and symbols

Description automatically generated



### 3.2.   Creating our first Queue in SQS

A screenshot of a computer

Description automatically generated

### There are two types of queue systems: Standard and FIFO. FIFO queues send messages once but have limitations it is slow. Standard queues do not guarantee message ordering, but they are faster. For applications like payments or similar ones, FIFO may be a good option, despite its slower speed. Alternatively, we can use a standard queue and implement application logic to handle multiple executions of the same queue object

### .3.3.   Publishing through the UI

A screenshot of a computer

Description automatically generated

You can not change type of a queue

### 3.4.  Creating a message publisher

using Amazon.SQS;

using Amazon.SQS.Model;

using System.Text.Json;

var sqsClient = new AmazonSQSClient();

var customer = new CustomerCreated {

Id = Guid.NewGuid(),

Email = "mehmetyagci53@gmail.com",

FullName = "Mehmet Yağcı",

DateOfBirth = new DateTime(1982, 1, 1),

GitHubUsername = "mehmetyagci",

};

var queueUrlResponse = await sqsClient.GetQueueUrlAsync("customers");

var sendMessageRequest = new SendMessageRequest {

QueueUrl = queueUrlResponse.QueueUrl,

MessageBody = JsonSerializer.Serialize(customer),

MessageAttributes = new Dictionary<string, MessageAttributeValue> {

{

"MessageType", new MessageAttributeValue

{

DataType = "String",

StringValue = nameof(CustomerCreated)

}

}

}

};

var response = await sqsClient.SendMessageAsync(sendMessageRequest);

Console.WriteLine($"{response.MessageId} {response.HttpStatusCode} {response.ResponseMetadata}");

### 3.5.  Creating a message consumer

using Amazon.SQS;

using Amazon.SQS.Model;

var cts = new CancellationTokenSource();

var sqsClient = new AmazonSQSClient();

var queueUrlResponse = await sqsClient.GetQueueUrlAsync("customers");

var receiveMessageRequest = new ReceiveMessageRequest {

QueueUrl = queueUrlResponse.QueueUrl,

AttributeNames= new List<string> { "All"}, // adding Attributes to the message

MessageAttributeNames = new List<string> { "All" } // adding MessageAttributes to the message

};

while (!cts.IsCancellationRequested) {

var response = await sqsClient.ReceiveMessageAsync(receiveMessageRequest, cts.Token);

foreach (var message in response.Messages) {

Console.WriteLine($"Message Id: {message.MessageId}");

Console.WriteLine($"Messaage Body: {message.Body}");

// Deleting messages

var deleteMessageResponse = await sqsClient.DeleteMessageAsync(queueUrlResponse.QueueUrl, message.ReceiptHandle);

}

await Task.Delay(1000);

}

### 3.6. The API we will be working on

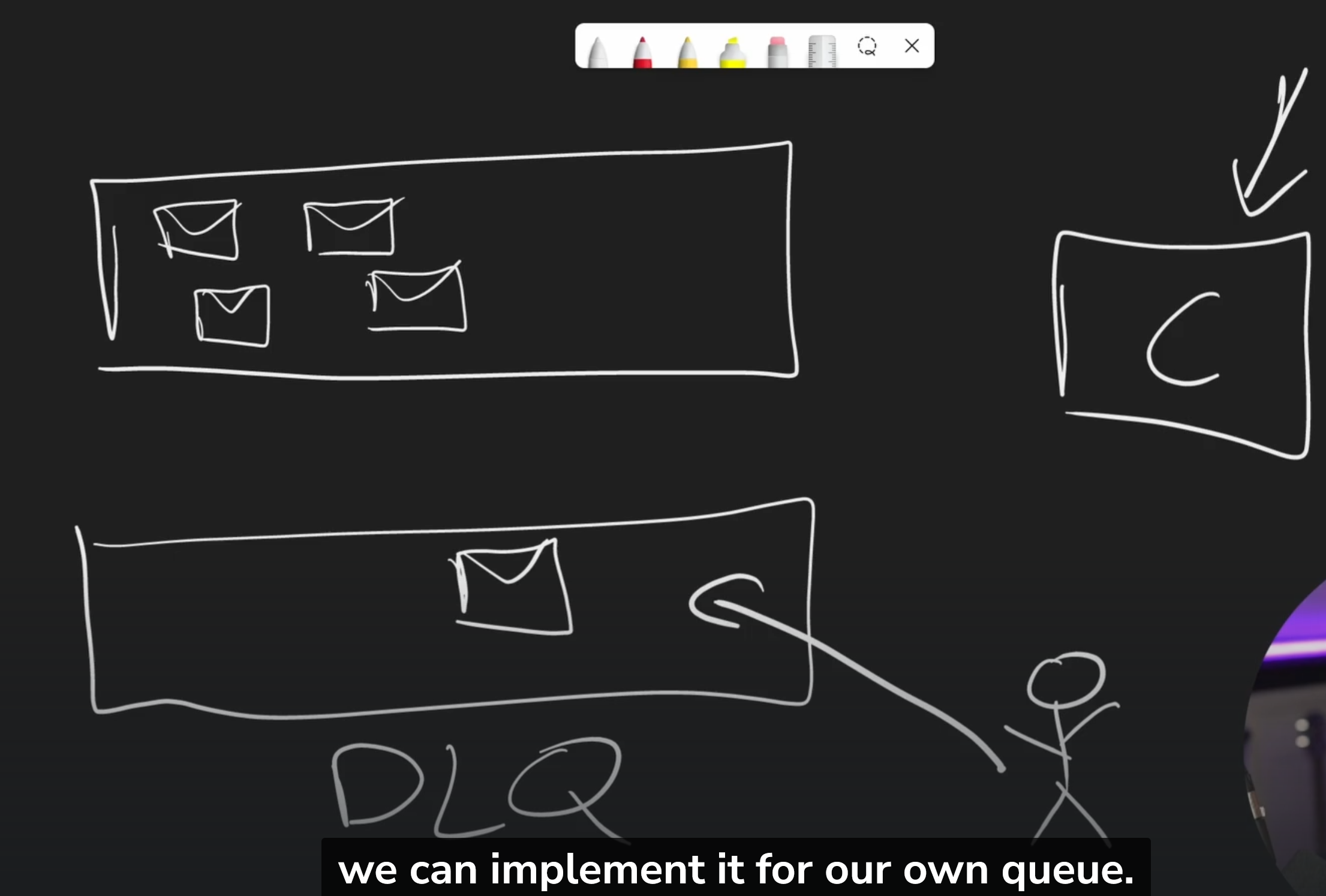
### 3.7. Adding message publishing

### 3.8. Implementing a consumer in ASP.NET Core

### 3.9. Refactoring the consumer

### 3.10. What happens when things go wrong?

### 3.11. What is a dead letter queue?



### 3.11. Creating a dead letter queue

### 3.12. Redriving dead messages

### 3.13. Section recap

## 4. AWS SNS

### 4.1. What is SNS?

Simple notification services.

### 4.2. Creating our first topic in SNS

### 4.3.

## 5. AWS DynamoDB

## 6. AWS S3

## 7. AWS Secret Manager