Practical Cloud Messaging

Elton Stoneman
geekswithblogs.net/eltonstoneman
@EltonStoneman

pluralsight
hardcore dev and IT training

Practical Cloud Messaging

Move all messaging to **the cloud**

Implement IMessageQueue with **Azure** and **AWS**

Use **parallelism** for message handling

Demo 1: Azure Service Bus Queues

Feature

Implement IMessageQueue using Azure

Task

Support request-response and fire-and-forget using queues

Task

Create temporary queues on demand

Demo 1: Azure Service Bus Queues

Demo 1: Azure Service Bus Queues

- **ServiceBusMessageQueue**
  - Initialise messaging factory

```
  Initialise(Direction.Inbound, name, pattern, properties);
  _factory = MessagingFactory.CreateFromConnectionString("...");
```

  - Create queue client

```
 _queueClient = _factory.CreateQueueClient(Address);
```

  - Send brokered message

```
 var brokeredMessage = new BrokeredMessage(message.ToJsonStream(), true);
 _queueClient.Send(brokeredMessage);
```

Demo 1: Azure Service Bus Queues

- **ServiceBusMessageQueue**
  - Receive brokered message

```
var brokeredMessage = _queueClient.Receive();
var messageStream = brokeredMessage.GetBody<Stream>();
var message = Message.FromJson(messageStream);
onMessageReceived(message);
```

  - Flag message as complete

```
brokeredMessage.Complete();
```

Demo 1: Azure Service Bus Queues

- **ServiceBusMessageQueue**
  - Get address

```
switch (name.ToLower())
{
    case "unsubscribe":
    return "unsubscribe";

    case "doesuserexist":
    return "doesuserexist";
}
```

Demo 1: Azure Service Bus Queues

- **ServiceBusMessageQueue**
  - Create temporary queue on demand

```
public override IMessageQueue GetResponseQueue()
{
    var responseAddress = Guid.NewGuid().ToString().Substring(0, 6);
    var manager = NamespaceManager.CreateFromConnectionString(
                                                        _connectionString);
    manager.CreateQueue(responseAddress);

    var responseQueue = MessageQueueFactory.CreateInbound(
                             responseAddress, MessagePattern.RequestResponse);
    return responseQueue;
}
```

Service Bus Topics and Subscriptions

Publish-Subscribe

Publish to Service Bus Topic

Topic relays to Subscription(s)

Subscribers listen on Subscriptions

Separate .NET classes

Demo 2: Azure Service Bus Topics

Feature

Complete Azure IMessageQueue implementation

Task

Create topic and subscriptions in Azure portal

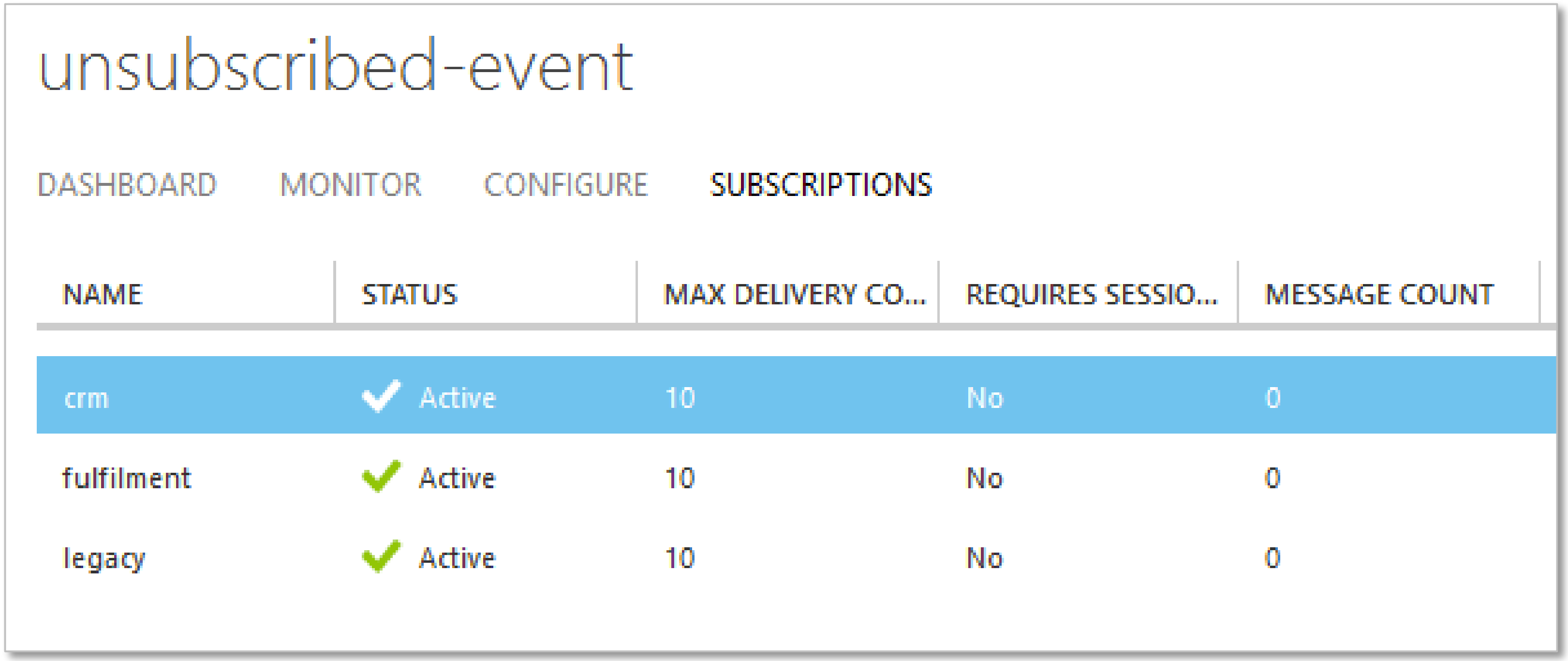
Task

Support publish-subscribe using topics

Demo 2: Azure Service Bus Topics

Demo 2: Azure Service Bus Topics

- **Azure Service Bus Topics**
  - Subscriptions accessed like queues

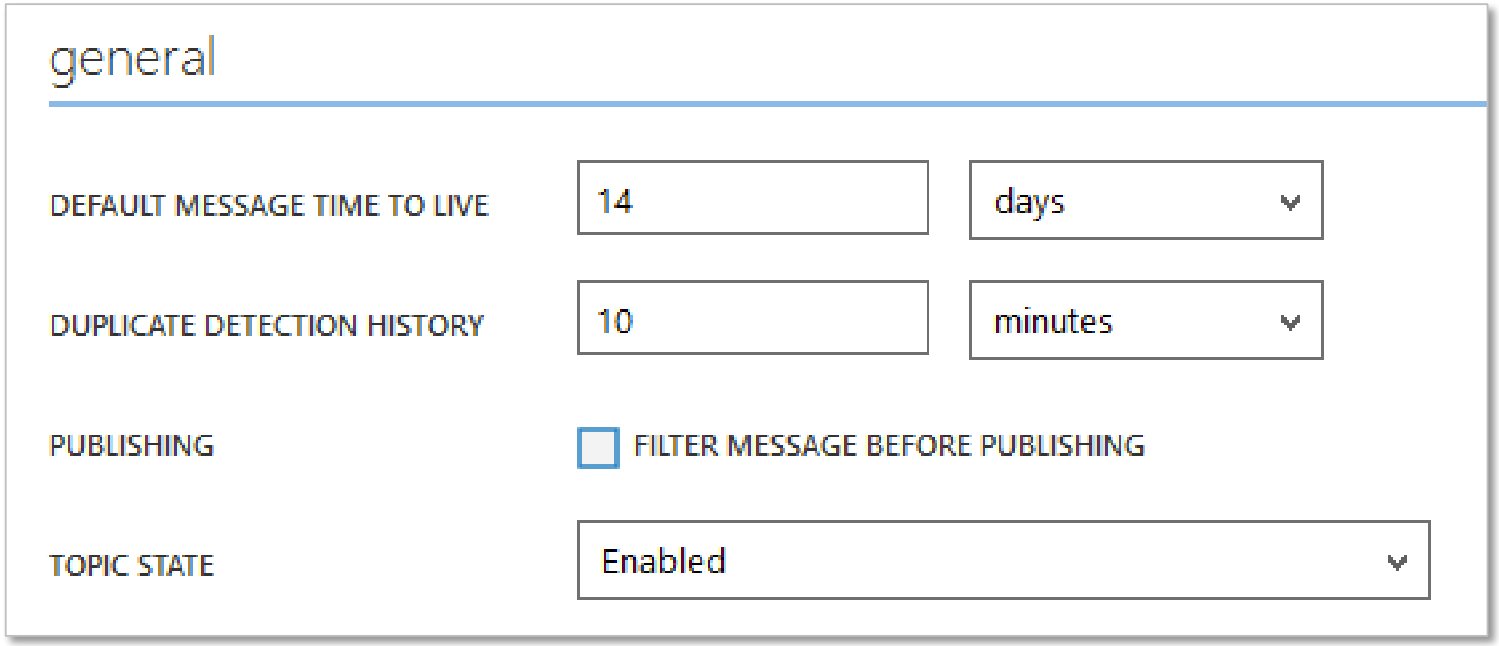


unsubscribed-event

NAME	STATUS	MAX DELIVERY CO...	REQUIRES SESSIO...	MESSAGE COUNT
crm	✓ Active	10	No	0
fulfilment	✓ Active	10	No	0
legacy	✓ Active	10	No	0

Demo 2: Azure Service Bus Topics

- **Azure Service Bus Topics**
  - Subset of queue configuration options



general

DEFAULT MESSAGE TIME TO LIVE	14	days
DUPLICATE DETECTION HISTORY	10	minutes
PUBLISHING	☐ FILTER MESSAGE BEFORE PUBLISHING	
TOPIC STATE	Enabled	

Demo 2: Azure Service Bus Topics

- **ServiceBusMessageQueue**
  - Topic client

```
if (Pattern == MessagePattern.PublishSubscribe)
{
    _topicClient = _factory.CreateTopicClient(Address);
}
```

  - Publish brokered message

```
var brokeredMessage = new BrokeredMessage(message.ToJsonStream(), true);
if (Pattern == MessagePattern.PublishSubscribe)
{
    _topicClient.Send(brokeredMessage);
}
```

Demo 2: Azure Service Bus Topics

- **ServiceBusMessageQueue**
  - Subscription client

```
if (Pattern == MessagePattern.PublishSubscribe)
{
    var addressParts = Address.Split(':');
    _subscriptionClient = _factory.CreateSubscriptionClient(
                                    addressParts[0], addressParts[1]);
```

  - Receive brokered message

```
BrokeredMessage brokeredMessage;
if (Pattern == MessagePattern.PublishSubscribe)
{
    brokeredMessage = _subscriptionClient.Receive();
}
//...
brokeredMessage.Complete();
```

Cloud Messaging

Publish-Subscribe

Easy extensibility

Add a new subscription

Fire up a new handler

Disconnected

Persistent messages in subscription

Start handler at any time

Dislocated

Handler can run on any premises

Or in any cloud

Demo 3: AWS Simple Queue Service

Feature

Implement IMessageQueue using Amazon SQS

Task

Support request-response and fire-and-forget

Task

Delete temporary queues when finished

Demo 3: AWS Simple Queue Service

Demo 3: AWS Simple Queue Service

- **AwsMessageQueue**
  - Inherit from common base
  - Initialise for outbound messaging

```
Initialise(Direction.Outbound, name, pattern, properties);
_sqsClient = new AmazonSQSClient(_accessKey, _secretKey,
                                                   RegionEndpoint.EUWest1);
```

  - Send message

```
var request = new SendMessageRequest();
request.MessageBody = message.ToJsonString();
request.QueueUrl = Address;
_sqsClient.SendMessage(request);
```

Demo 3: AWS Simple Queue Service

- **AwsMessageQueue**
  - Initialise for inbound messaging

```
Initialise(Direction.Inbound, name, pattern, properties);
_sqsClient = new AmazonSQSClient(_accessKey, _secretKey,
                                              RegionEndpoint.EUWest1);
```

  - Receive message

```
var request = new ReceiveMessageRequest();
request.QueueUrl = Address;
var response = _sqsClient.ReceiveMessage(request);
var firstMessage = response.Messages.FirstOrDefault();
if (firstMessage != null)
{
    var message = Message.FromJson(firstMessage.Body);
    onMessageReceived(message);
```

Demo 3: AWS Simple Queue Service

- **AwsMessageQueue**
  - Manually delete handled message
  - By receipt handle

```
var deleteRequest = new DeleteMessageRequest();
deleteRequest.QueueUrl = request.QueueUrl;
deleteRequest.ReceiptHandle = firstMessage.ReceiptHandle;
_sqsClient.DeleteMessage(deleteRequest);
```

Demo 3: AWS Simple Queue Service

- **AwsMessageQueue**
  - Explicit delete queue functionality

```
public static void Delete(IMessageQueue queue)
{
    queue.DeleteQueue();
    var queueEntries = _Queues.Where(x => x.Value.Address ==
                                            queue.Address).ToList();
    queueEntries.ForEach(x =>
    {
        x.Value.Dispose();
        _Queues.Remove(x.Key);
    });
}
```

Demo 3: AWS Simple Queue Service

- **AwsMessageQueue**
  - Explicit delete queue functionality
  - Called by consumer
  - After received response

```
responseQueue.Receive(x => exists = x.BodyAs<DoesUserExistResponse>()
                                                        .Exists, 5000);
MessageQueueFactory.Delete(responseQueue);
```

AWS Simple Notification Service

Publish-Subscribe

Publish to SNS Topic

Topic relays to SQS Queue(s)

Subscribers listen on Queues

Separate .NET topic client class

Demo 4: AWS Simple Notification Service

Feature

Complete IMessageQueue using Amazon SNS

Task

Create SNS Topic and link SQS queues in AWS Console

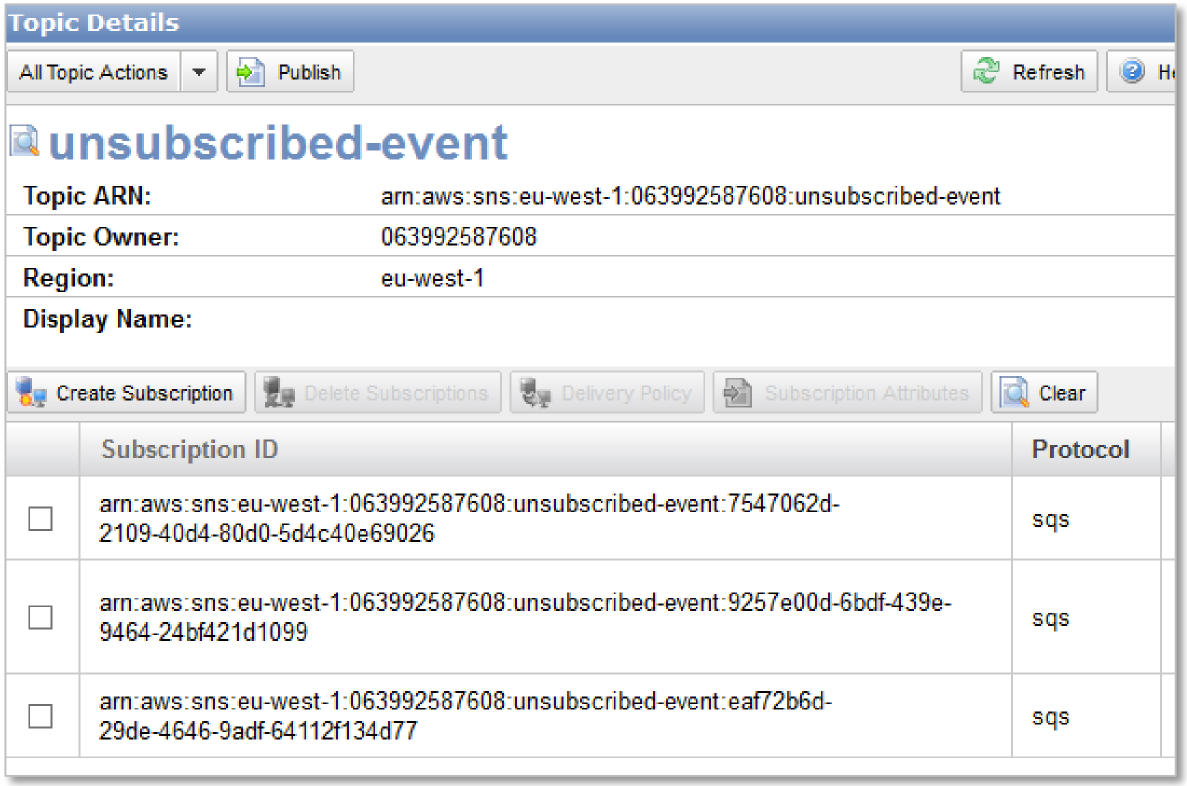
Task

Support publish-subscribe using topics

Demo 4: AWS Simple Notification Service

Demo 4: AWS Simple Notification Service

- **SNS Topic**
  - Create in AWS Management Console
  - Link SQS queues as (raw message) subscribers



unsubscribed-event

Topic ARN:	arn:aws:sns:eu-west-1:063992587608:unsubscribed-event
Topic Owner:	063992587608
Region:	eu-west-1
Display Name:	

	Subscription ID	Protocol
☐	arn:aws:sns:eu-west-1:063992587608:unsubscribed-event:7547062d-2109-40d4-80d0-5d4c40e69026	sqs
☐	arn:aws:sns:eu-west-1:063992587608:unsubscribed-event:9257e00d-6bdf-439e-9464-24bf421d1099	sqs
☐	arn:aws:sns:eu-west-1:063992587608:unsubscribed-event:eaf72b6d-29de-4646-9adf-64112f134d77	sqs

Demo 4: AWS Simple Notification Service

- **AwsMessageQueue**
  - Initialise for outbound publishing

```
Initialise(Direction.Outbound, name, pattern, properties);
if (Pattern == MessagePattern.PublishSubscribe)
{
    _snsClient = new AmazonSimpleNotificationServiceClient(_accessKey,
                                  _secretKey, RegionEndpoint.EUWest1);
}
```

  - Send message

```
if (Pattern == MessagePattern.PublishSubscribe)
{
    var publishRequest = new PublishRequest();
    publishRequest.TopicArn = Address;
    publishRequest.Message = message.ToJsonString();
    _snsClient.Publish(publishRequest);
}
```

Demo 4: AWS Simple Notification Service

- **AwsMessageQueue**
  - Topic address is ARN (Amazon Resource Name)
  - Queue is URL

```
switch (name.ToLower())
{
    //...
    case "unsubscribed-event":
        return "arn:aws:sns:eu-west-1:063992587608:unsubscribed-event";

    case "unsubscribe-legacy":
        return "https://sqs.eu-west-1.amazonaws.com/063992587608/...";
    //...
```

Parallel Handling

Single-threaded

Each message processed in turn

Performance limitation

Cloud – polling uses messages

Additional latency

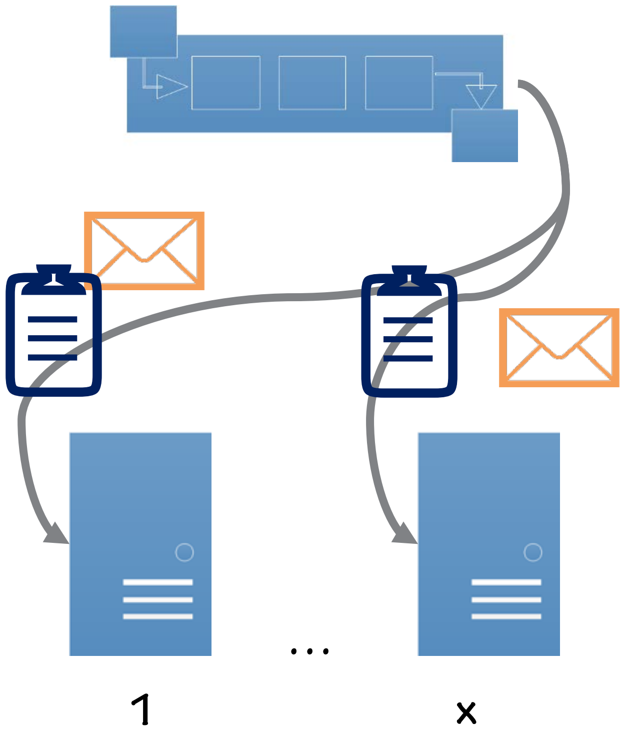
And cost

Parallel processing

Handle multiple messages concurrently

Without extra polling

Multi-threaded Listeners



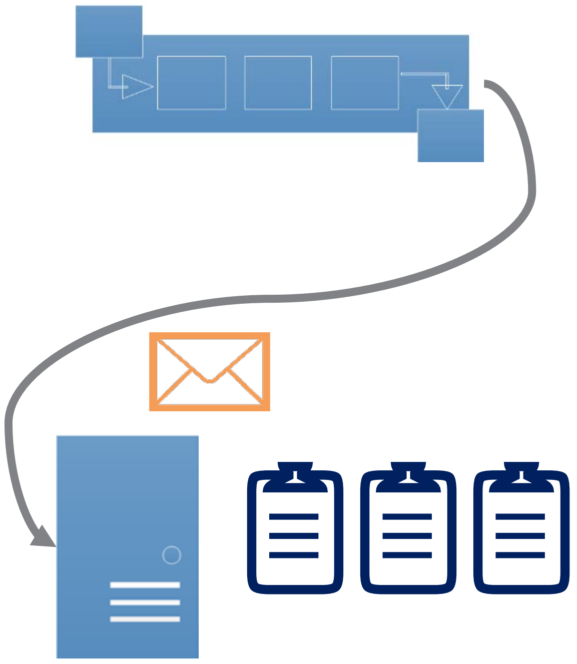
Multiple handler instances

Poll queue & process message

App & messaging code threadsafe

Increased receive requests

Multi-threaded Message Handling



Single polling thread

New thread for message handler

Parallel message processing

App code threadsafe

Demo 5: Multi-threaded Message Handling

Feature

Walkthrough multi-threaded handler

Task

Run handler actions using TPL for listeners

Task

Use most efficient polling process for queue

Demo 5: Multi-threaded Message Handling

Demo 5: Multi-threaded Message Handling

- **IMessageQueue**
  - Async Listen() method – with CancellationToken
  - Blocking Receive() method – with timeout

```
void Listen(Action<Message> onMessageReceived,
                                    CancellationToken cancellationToken);

void Receive(Action<Message> onMessageReceived,
                                        int maximumWaitMilliseconds = 0);
```

- **Cloud queues**
  - Multiple messages from one receive call

- **All queues**
  - Single polling thread
  - Process each message with TPL

Summary

- **Cloud IMessageQueues** ✓
  - Azure & AWS
- **Azure Service Bus** ✓
  - Queues: request-response, fire-and-forget
  - Topics: publish-subscribe
- **Amazon Web Services** ✓
  - SQS: request-response, fire-and-forget
  - SNS + SQS: publish-subscribe
- **Practical refactoring** ✓
  - Tighter control over temporary queues
  - Parallel message handling

WebSphere MQ