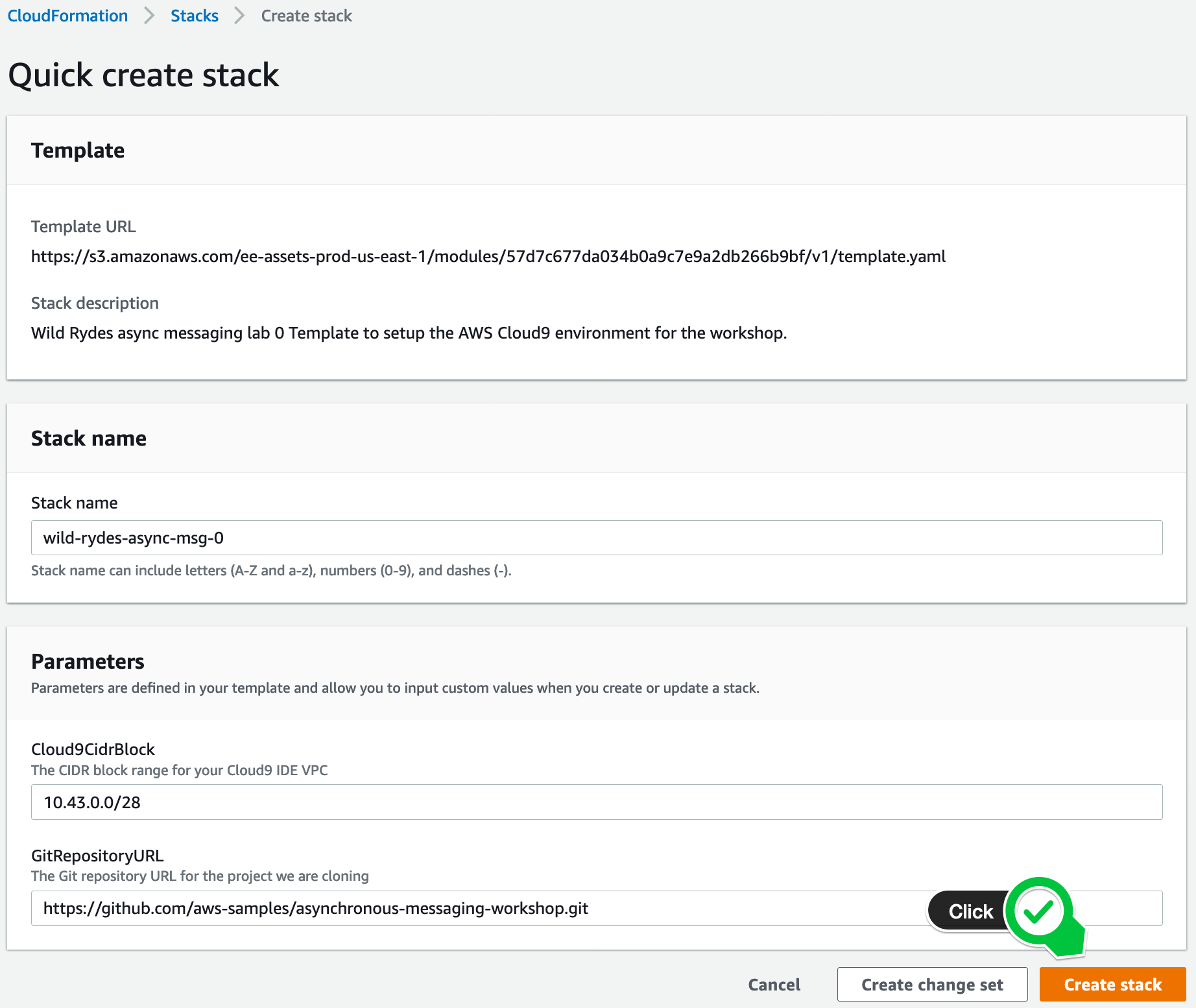
We will leverage [**AWS CloudFormation**](https://aws.amazon.com/cloudformation/) which allows us to codify our infrastructure. In addition, we use [**AWS SAM**](https://aws.amazon.com/serverless/sam/) to build serverless applications in simple and clean syntax.

To get started, follow [this deeplink](https://us-east-1.console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/quickcreate?stackName=wild-rydes-async-msg-0&templateURL=https://s3.amazonaws.com/ee-assets-prod-us-east-1%2fmodules%2f57d7c677da034b0a9c7e9a2db266b9bf%2fv1%2ftemplate.yaml) to load the CloudFormation template.

Launch the AWS CloudFormation stack

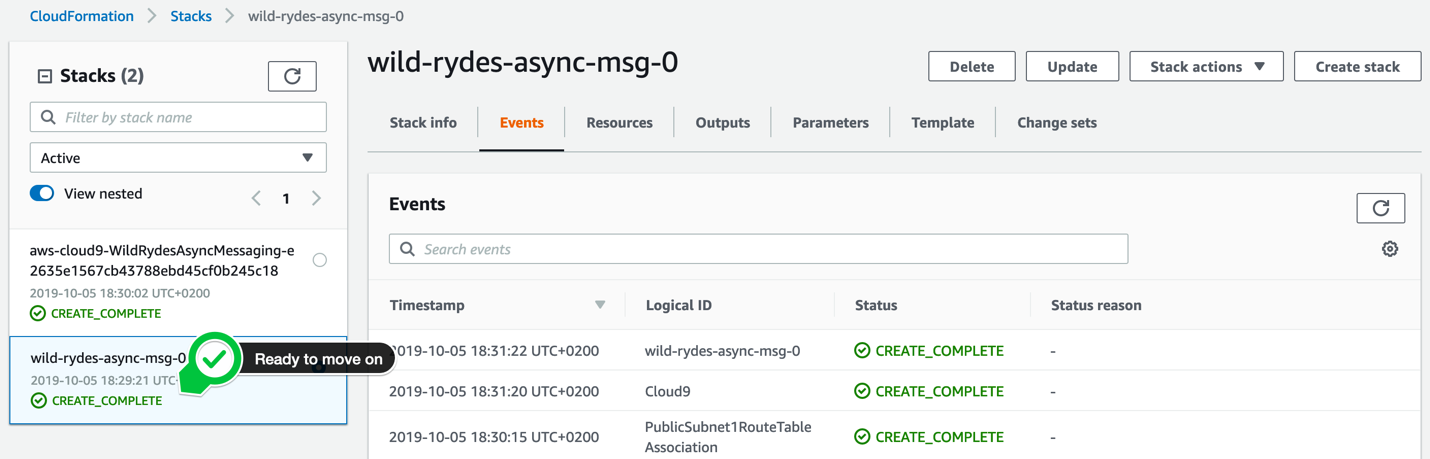
Just click the **Create Stack** button to launch the template.

 Detailed description

[](https://async-messaging.workshop.aws/prerequisites/prerequisites-1/step-1.png)

3. Wait until the AWS CloudFormation stack launched

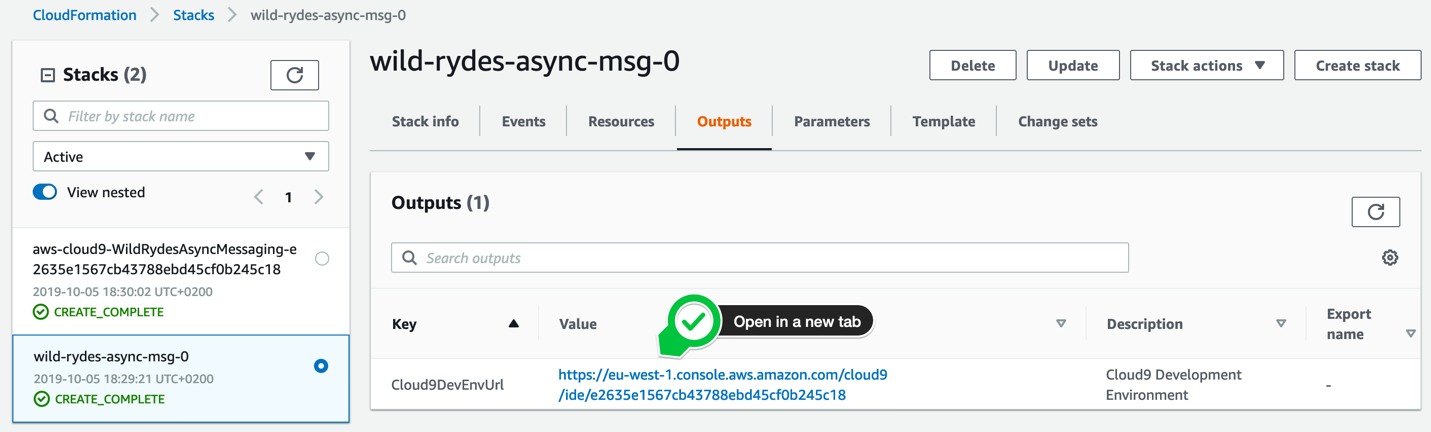
It takes usually less than 2 minutes until the stack launched. When the stack is launched, the status will change from **CREATE\_IN\_PROGRESS** to **CREATE\_COMPLETE**.

[](https://async-messaging.workshop.aws/prerequisites/prerequisites-1/step-2.png)

# **CONFIGURE AWS CLOUD9**

1. Browse to your AWS Cloud9 development environment

Open your [CloudFormation Console](https://console.aws.amazon.com/cloudformation/home?#/stacks) and select the stack at the bottom (the name could vary). The Outputs tab exposes the Cloud9DevEnvUrl parameter. Click at the corresponding URL in the value column and open your AWS Cloud9 development environment in a new tab.

[](https://async-messaging.workshop.aws/prerequisites/prerequisites-2/step-1.png)

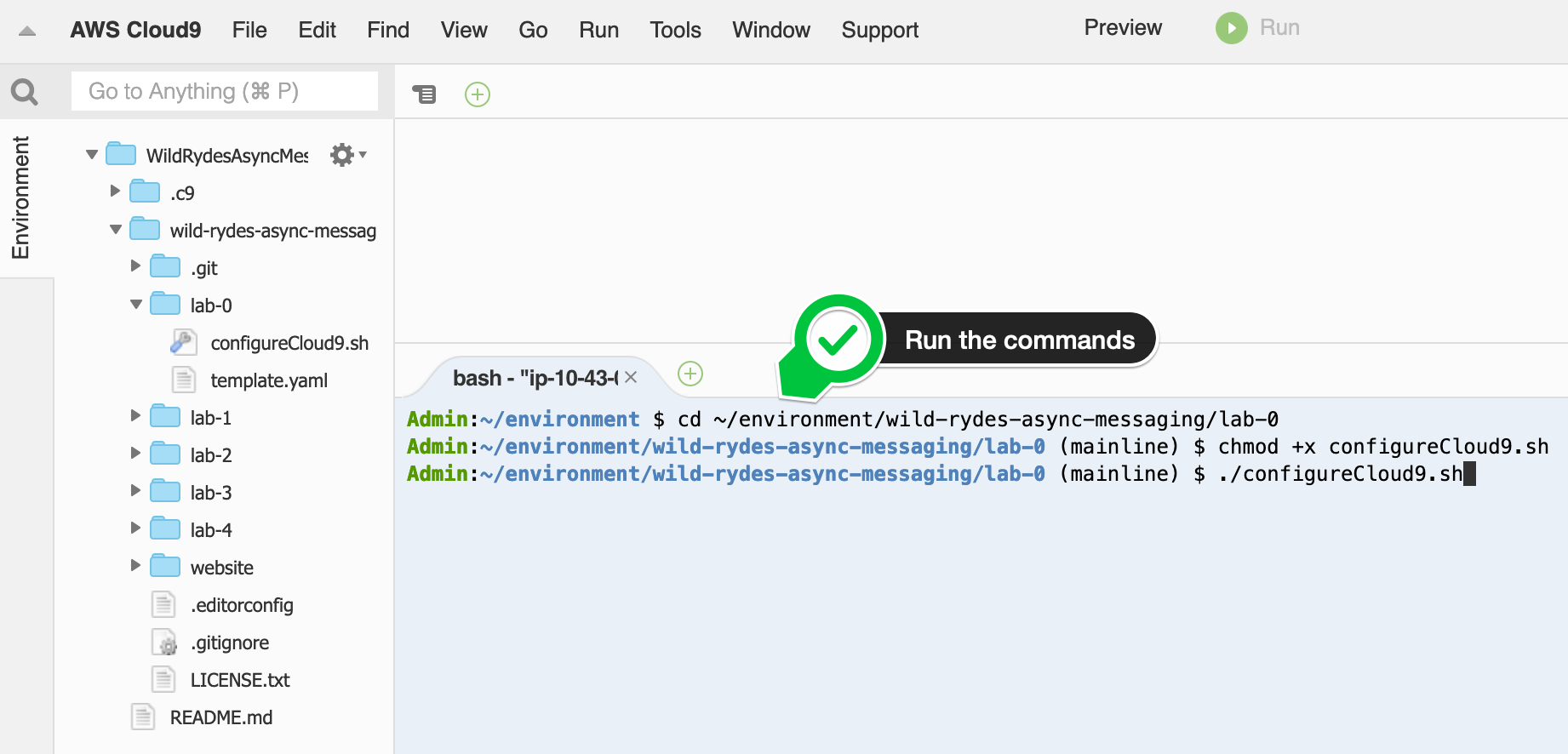
2. Configure your AWS Cloud9 development environment

In your AWS Cloud9 IDE, you can close the welcome tab. In the left environment navigation window, you can see the project **wild-rydes-async-messaging** we have already checked out for you from Github. This project also contains a shell script to setup your environment. Run the following command in the **bash** tab (at the bottom of the IDE):

cd ~/environment/wild-rydes-async-messaging/lab-0

chmod +x configureCloud9.sh

./configureCloud9.sh

[](https://async-messaging.workshop.aws/prerequisites/prerequisites-2/step-2.png)

It takes usually less then 3 minutes, until the AWS Cloud9 IDE is updated. In the meantime while your waiting, you may want to have a look at some handy AWS Cloud9 shortcuts, like the [**multiple cursors**](https://docs.c9.io/docs/multiple-cursors) or the AWS Cloud9 **[keybindings](https://docs.c9.io/docs/keybindings" \t "_blank)**.

**You are now ready to get started!!!**

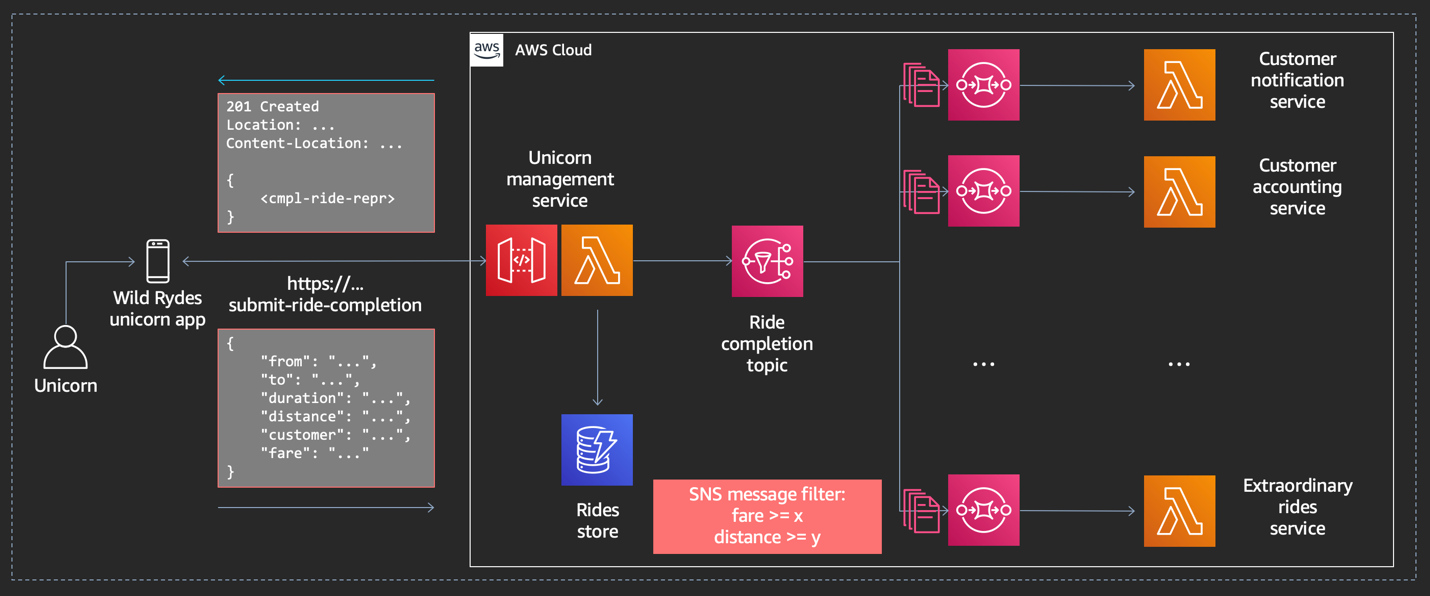
# **TOPIC-QUEUE CHAINING & LOAD BALANCING**

Let’s look at the publish/subscribe channel between the unicorn management service and all 3 backend services on the right hand side that are interested in getting notified about ride completions.

One of these services could happen to be taken offline for maintenance. Or the code that processes messages coming in from the ride completion topic could run into an exception. These are two examples where a subscriber service could potentially miss topic messages. A good pattern to apply here is **topic-queue-chaining**. That means that you add a queue, in our case an Amazon SQS queue, between the ride completion Amazon SNS topic and each of the subscriber services.  
As messages are buffered in a persistent manner in an SQS queue, no message will get lost should a subscriber process run into problems for many hours or days, or has exceptions or crashes.

But there is even more to it. By having an Amazon SQS queue in front of each subscriber service, we can leverage the fact that a queue can act as a **buffering load-balancer**. Due to nature that every queue message is delivered to one of potentially many consumer processes, you can easily scale your subscriber services out & in and the message load will be distributed over the available consumer processes. Furthermore, since messages are buffered in the queue, also a scaling event, for instance when you need to wait until an additional consumer process becomes operational, will not make you lose messages.

In this lab, we will develop the architecture below:

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/module-2.png)

## **Lab Objectives**

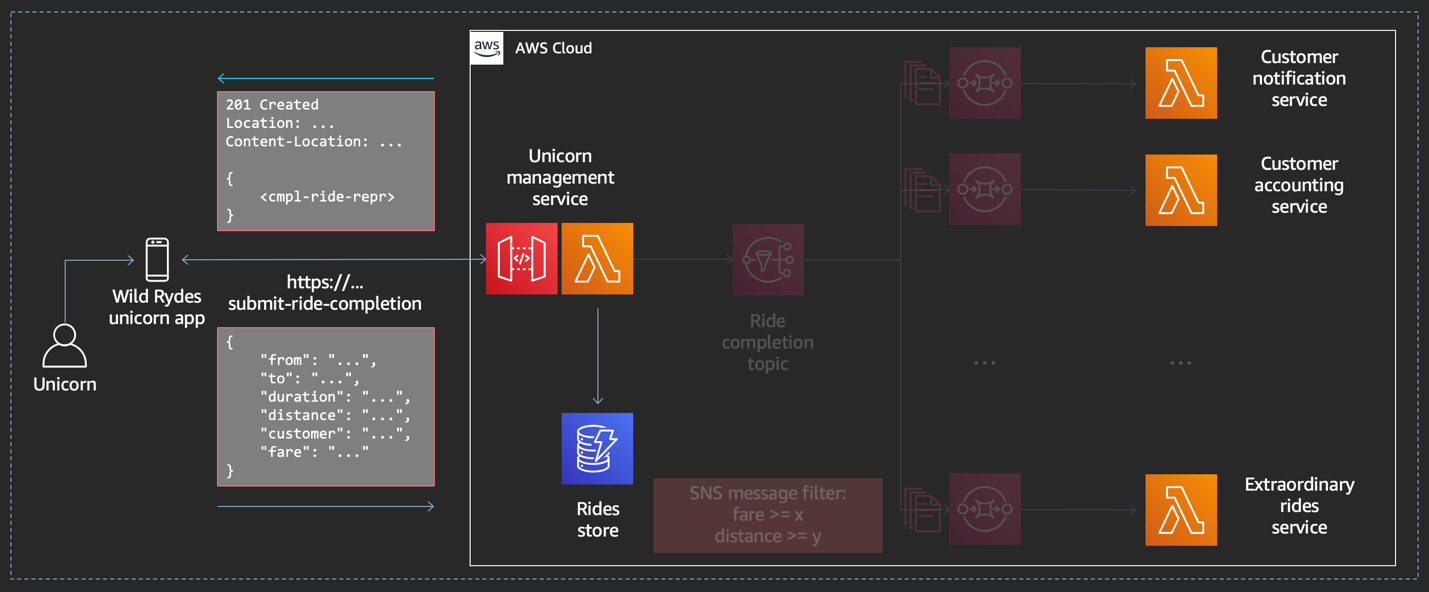
In this lab, you will acquire the following skills:

* **How to create an Amazon SQS queue?**
* **How to leverage Amazon SQS as event source for AWS Lambda?**
* **How to add an Amazon SQS subscription to an Amazon SNS topic?**
* **How to define a subscription filter in an Amazon SNS subscriptions?**
* **How to call Amazon SNS from AWS Lambda?**

**Lab source code**  
If you are curious and would like to dive into the lab’s source code, you are more than welcome to do so. You will find the source code of this lab in our Github repo [**here**](https://github.com/aws-samples/asynchronous-messaging-workshop/tree/master/lab-2).

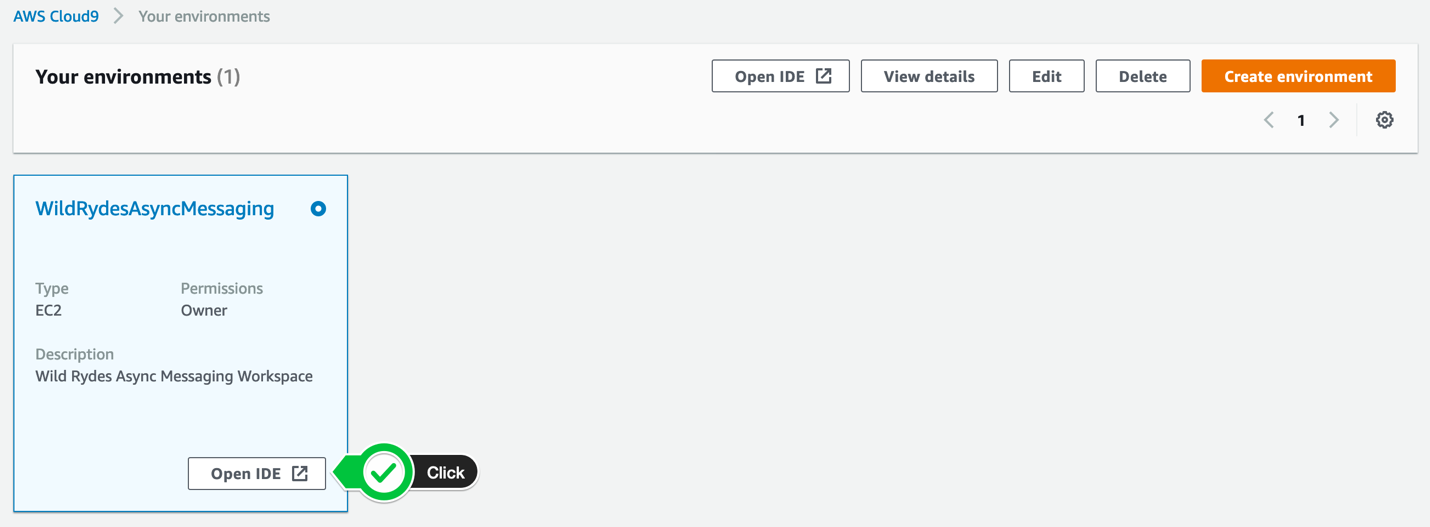
# **BOOTSTRAP THE INITIAL STATE**

First, we will setup the initial state, including the integrating of the **Unicorn Management Service** (leveraging [Amazon API Gateway](https://aws.amazon.com/api-gateway/) and [AWS Lambda](https://aws.amazon.com/lambda/)), the **Rides Store** (leveraging [Amazon DynamoDB](https://aws.amazon.com/dynamodb/)) and all **3 backend services** (leveraging [AWS Lambda](https://aws.amazon.com/lambda/)).

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/bootstrap-initial-state/step-1.png)

#### **1. Browse to your AWS Cloud9 IDE**

Browse to your [AWS Cloud9 Console](https://console.aws.amazon.com/cloud9/home) and select the environment called **WildRydesAsyncMessaging**.

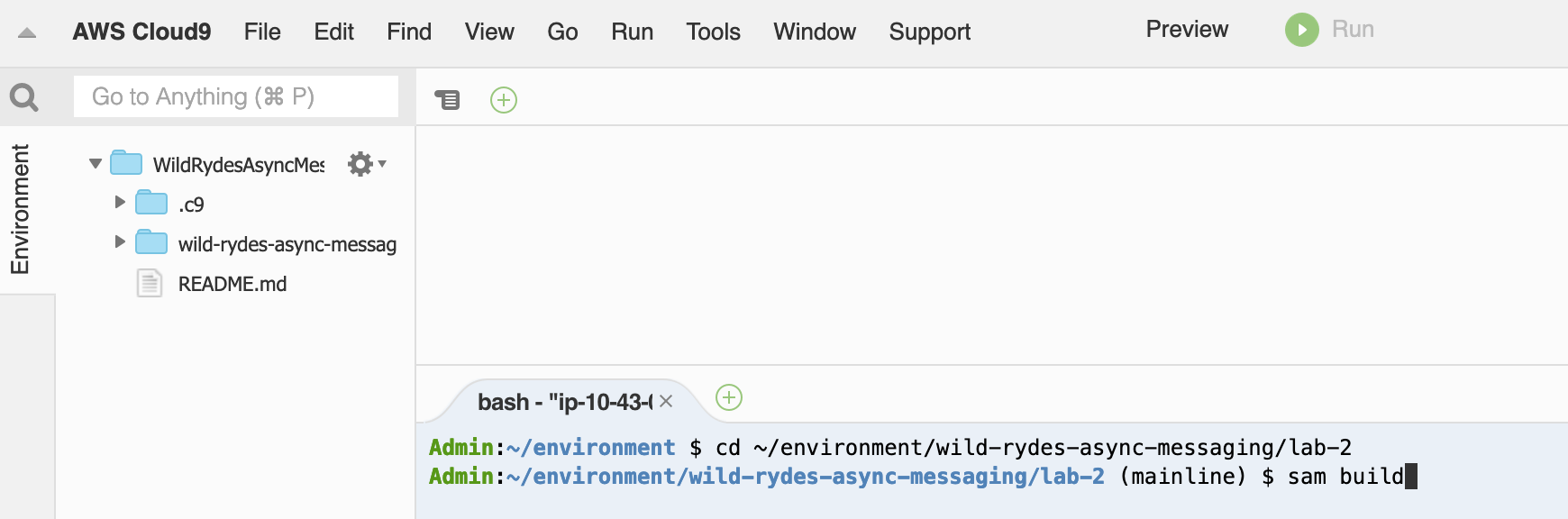
[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/bootstrap-initial-state/step-2.png)

#### **2. Build the lab artifacts from source**

We provide you with an [AWS SAM](https://aws.amazon.com/serverless/sam/) template which we will use to bootstrap the initial state. In the **bash tab** (at the bottom) in you **AWS Cloud9 IDE**, run the following commands to build the lab code:

cd ~/environment/wild-rydes-async-messaging/lab-2

sam build

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/bootstrap-initial-state/step-4.png)

#### **3. Deploy the application**

Now we are ready to deploy the application in the **lab-2** directory.

If this is the first time you deploying this application, run the following command:

export AWS\_REGION=$(aws --profile default configure get region)

sam deploy \

--stack-name wild-rydes-async-msg-2 \

--capabilities CAPABILITY\_IAM \

--region $AWS\_REGION \

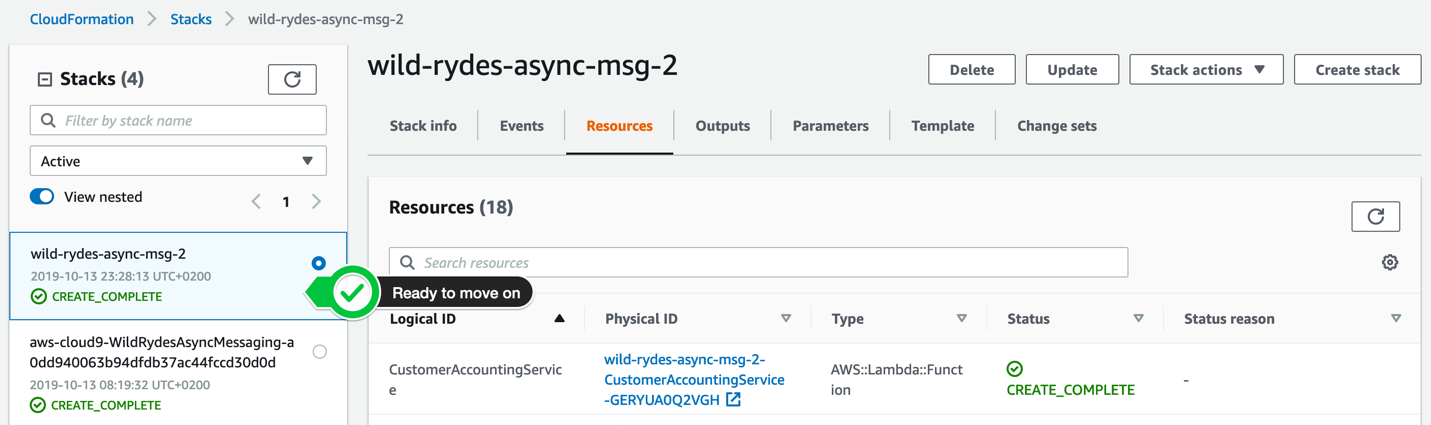
--guided

You will be asked to set the default arguments for your deployment. All input is saved in a SAM configuration file (samconfig.toml), so you won’t be prompted for this information again.

* Confirm the first 4 proposed arguments by hitting **ENTER**.
* When you get asked **SubmitRideCompletionFunction may not have authorization defined, Is this okay? [y/N]:**, enter y and hit **ENTER**.
* Accept the remaining default arguments by hitting **ENTER**.

#### **4. Wait until the stack is successfully deployed**

It takes usually 4 minutes until the stack launched. You can monitor the progress of the **wild-rydes-async-msg-2** stack in your SAM CLI or your [AWS CloudFormation Console](https://console.aws.amazon.com/cloudformation). When the stack is launched, the status will change from **CREATE\_IN\_PROGRESS** to **CREATE\_COMPLETE**.

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/bootstrap-initial-state/step-7.png)

In the meantime while your waiting, you may want to have a look at the AWS SAM template to make yourself familiar with the stack we launched. Just click on the **template.yaml** attachment below to see the content.

 Related files

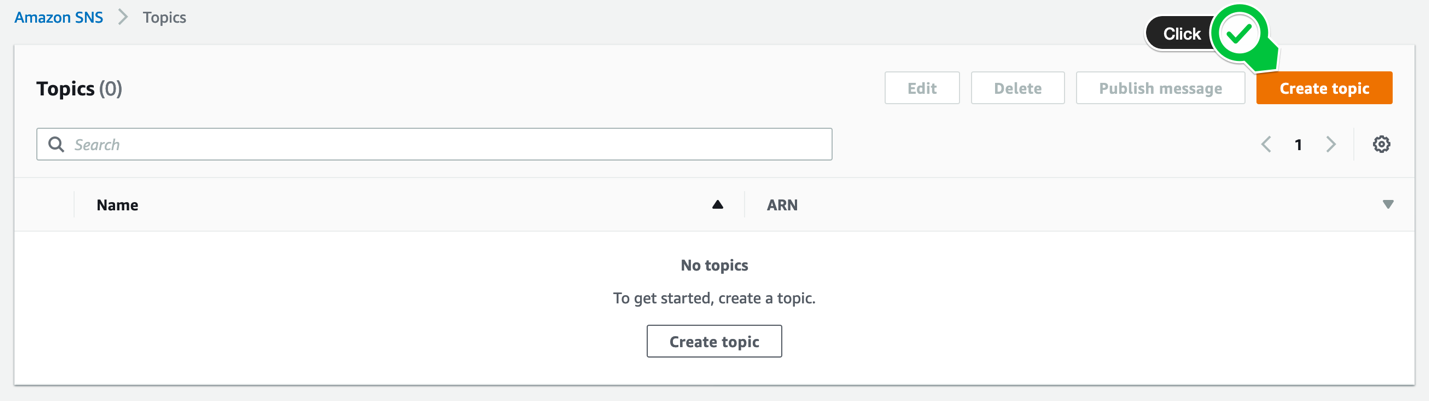
 [template.yaml](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/bootstrap-initial-state/bootstrap-initial-state.files/template.yaml) (8 kb)

# **CREATE THE AMAZON SNS TOPIC**

In this step, you will create the Amazon SNS topic via the AWS console.

#### **1. Browse to the Amazon SNS console**

In your [Amazon SNS console](https://console.aws.amazon.com/sns/v3/home?#/topics), select **Topic** in the left navigation pane and click the **Create topic** button in the top right corner.

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-sns-topic/step-1-console.png)

#### **2. Create the Ride Completion Topic**

Enter the topic name **RideCompletionTopic** and leave the default values. Scroll to the bottom of the page and click **Create topic**.

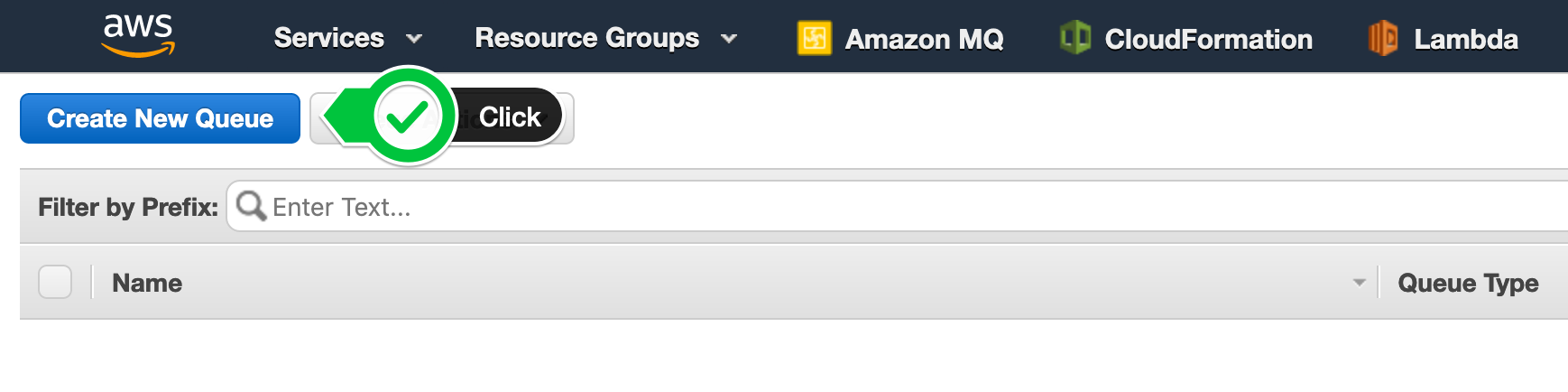
[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-sns-topic/step-2-console.png)

CREATE CUSTOMER NOTIFICATION SERVICE SUBSCRIPTION

In this step, we will create an **Amazon SQS** queue for the **Customer Notification Service** and add a subscription to the Amazon SNS topic we created before:

#### **1. Create a new Amazon SQS queue**

In your [**Amazon SQS console**](https://console.aws.amazon.com/sqs/home?), select **Create New Queue** in top left corner or click **Get Started Now** in the center of the page, if it’s your first queue in this region.

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-1-console.png)

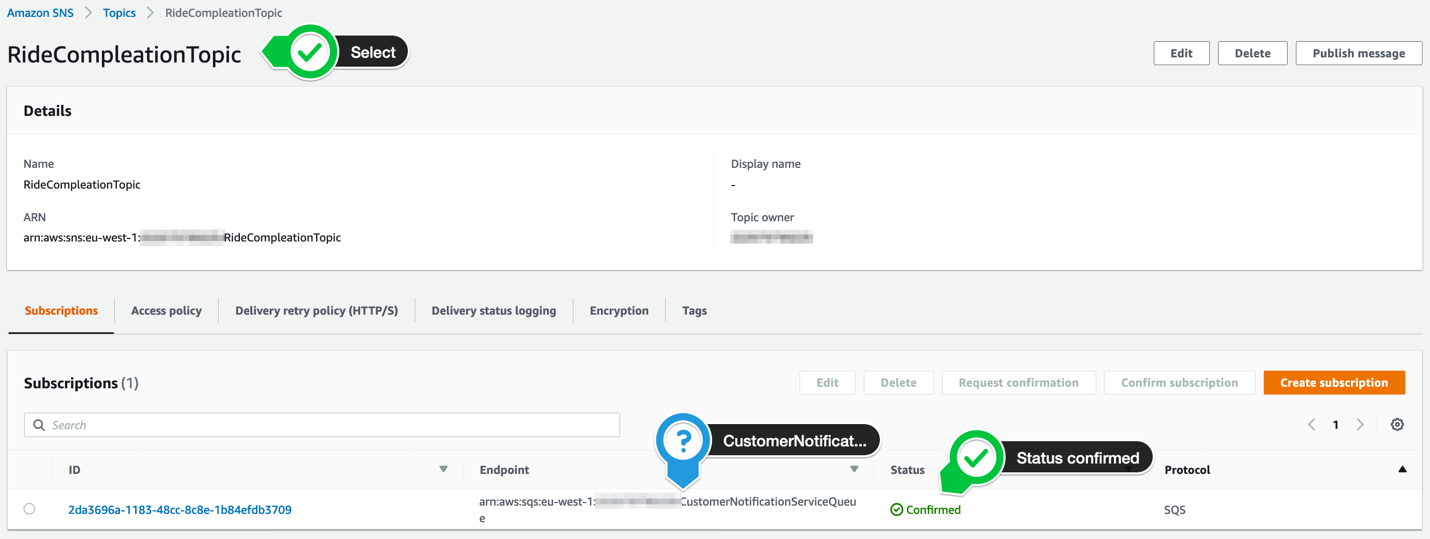
Enter CustomerNotificationServiceQueue as **Queue Name**, make sure **Standard Queue** is highlighted and click **Create Queue**.

#### **2. Create a new subscription**

After queue is created **Subscribe Queue to SNS Topic**. n the pop up window, select the **RideCompletionTopic** and click **Subscribe**. Click **save** in the confirmation dialog.

#### **3. Validate the subscription confirmation**

Browse to your [**Amazon SNS console**](https://console.aws.amazon.com/sns/v3/home?#/topics) to list your existing topics. Select the **RideCompletionTopic** and verify, the subscription has the status **Confirmed**.

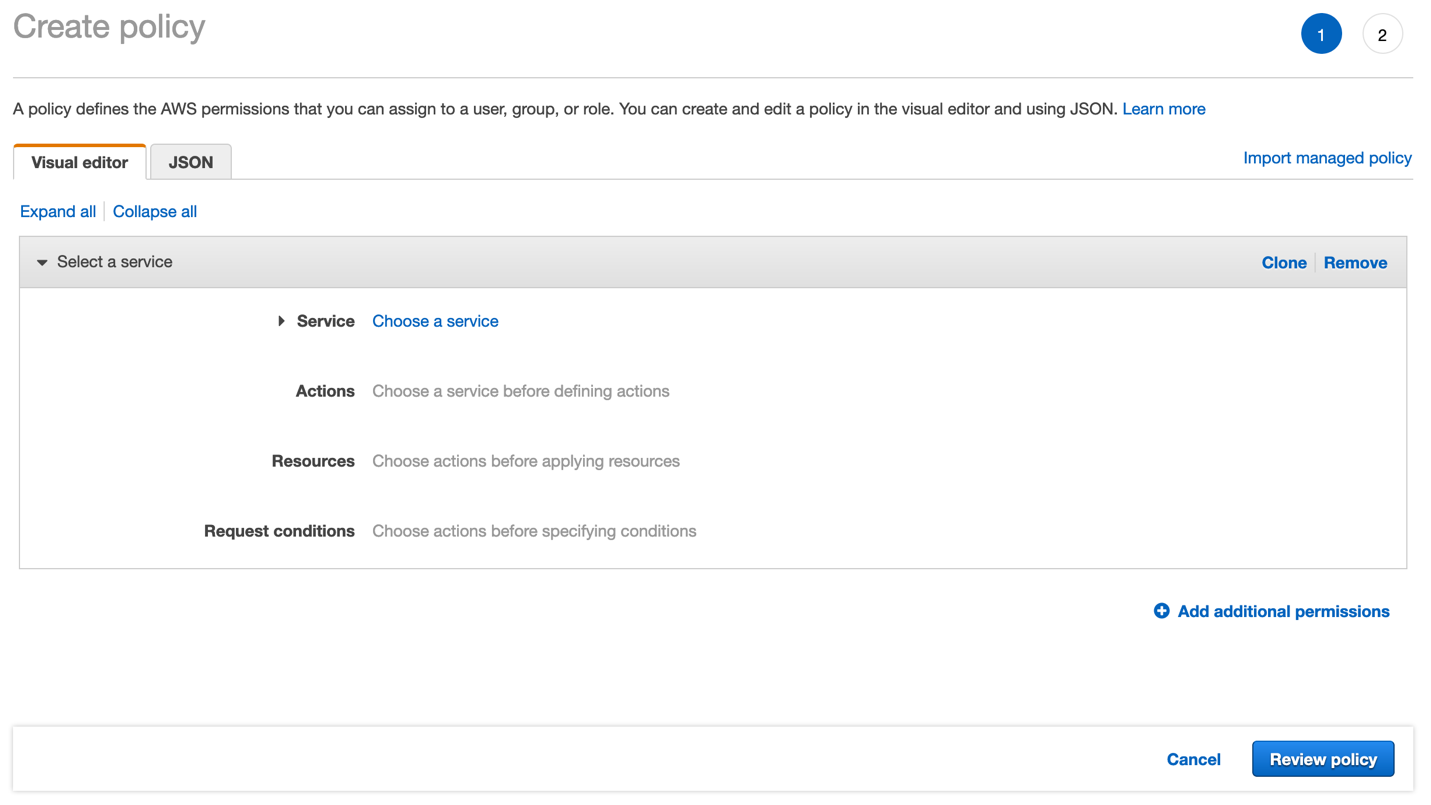
[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-5-console.png)

#### **4. Grant permissions to our function to access the Amazon SQS queue**

In your [**Amazon IAM console**](https://console.aws.amazon.com/iam), select **Roles** in the left navigation. Use the filter text box to find the role with the name **wild-rydes-async-msg-2-CustomerNotificationService-…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name).

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-6-console.png)

Click on the role name and click **Add inline policy** to attache another one.

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-7-console.png)

Select the **JSON** tab and passed the following policy statement into it, after you have substitute «…» with the correct values. It will add the permission to your Lambda function to access the Amazon SQS queue:

{

"Statement": [

{

"Effect": "Allow",

"Action": [

"sqs:ChangeMessageVisibility",

"sqs:ChangeMessageVisibilityBatch",

"sqs:DeleteMessage",

"sqs:DeleteMessageBatch",

"sqs:GetQueueAttributes",

"sqs:ReceiveMessage"

],

"Resource": "arn:aws:sqs:<<AWS REGION>>:<<AWS ACCOUNT ID>>:CustomerNotificationServiceQueue"

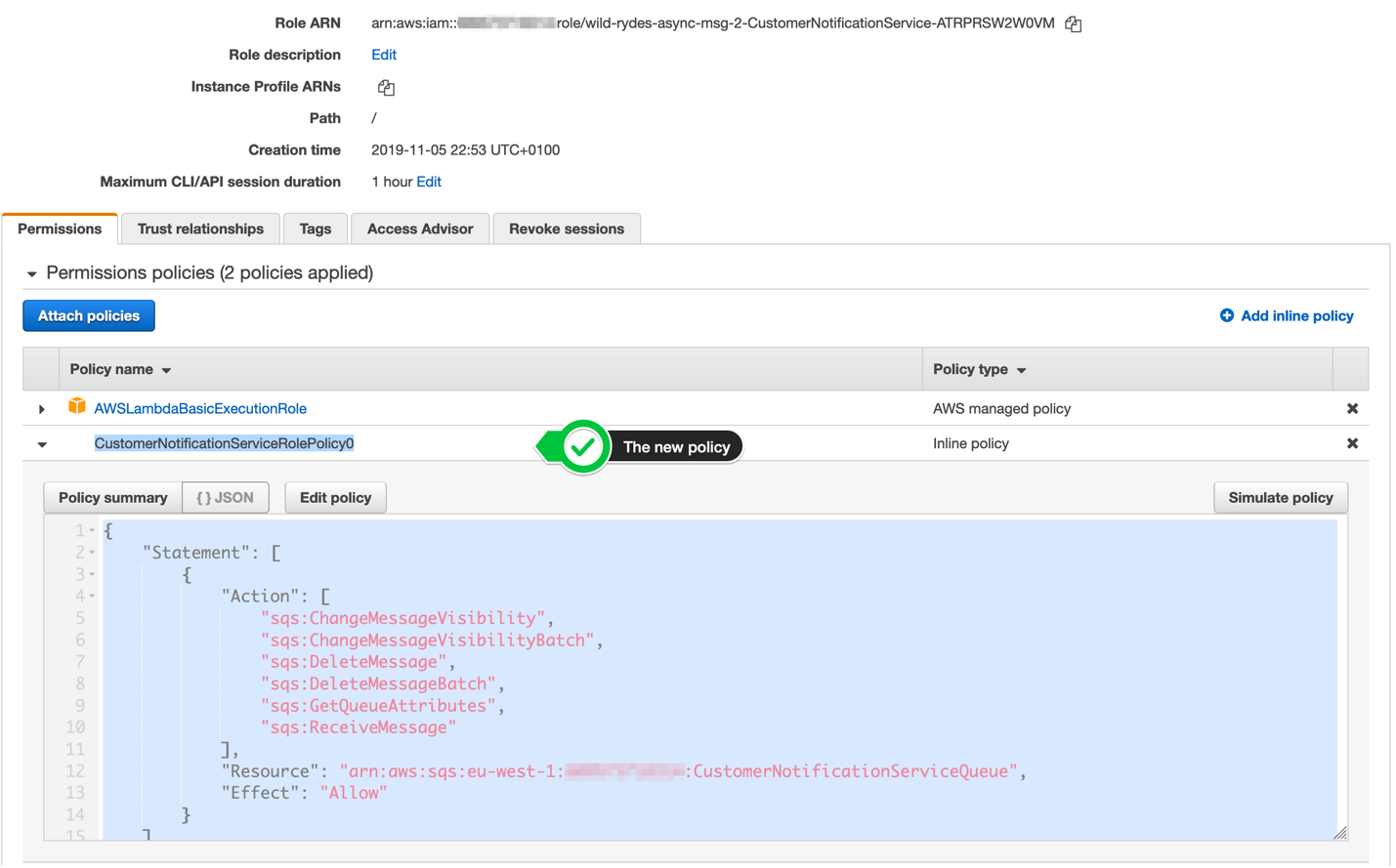
}

]

}

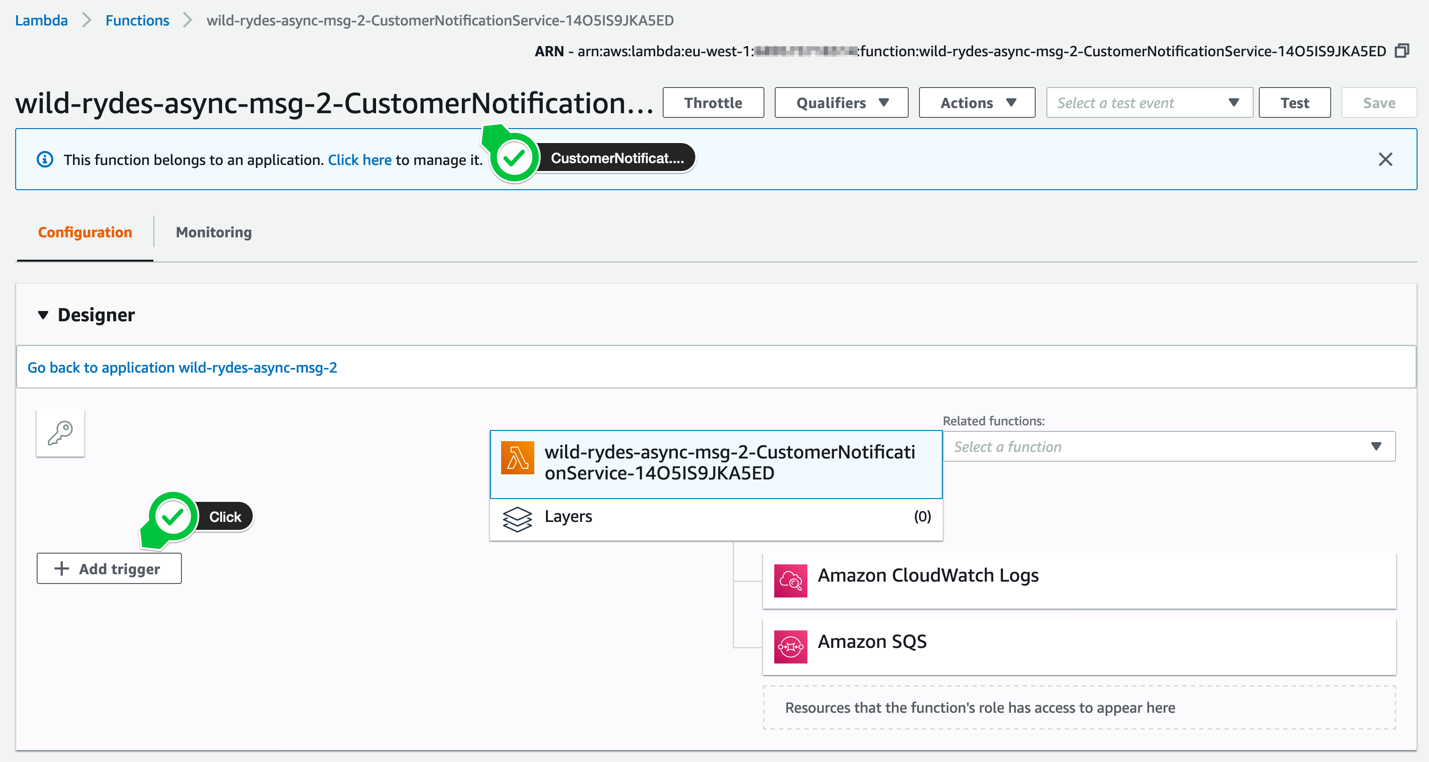
Make sure you provide the AWS ACCOUNT ID in the form of XXXXXXXXXXXX and not XXXX-XXXX-XXXX!

Click **Review policy** and enter the **Name** CustomerNotificationServiceRolePolicy0. Click **Create policy**. To validate this step, select on the role again and your should see 3 policies attached to your role, including the one you just have created:

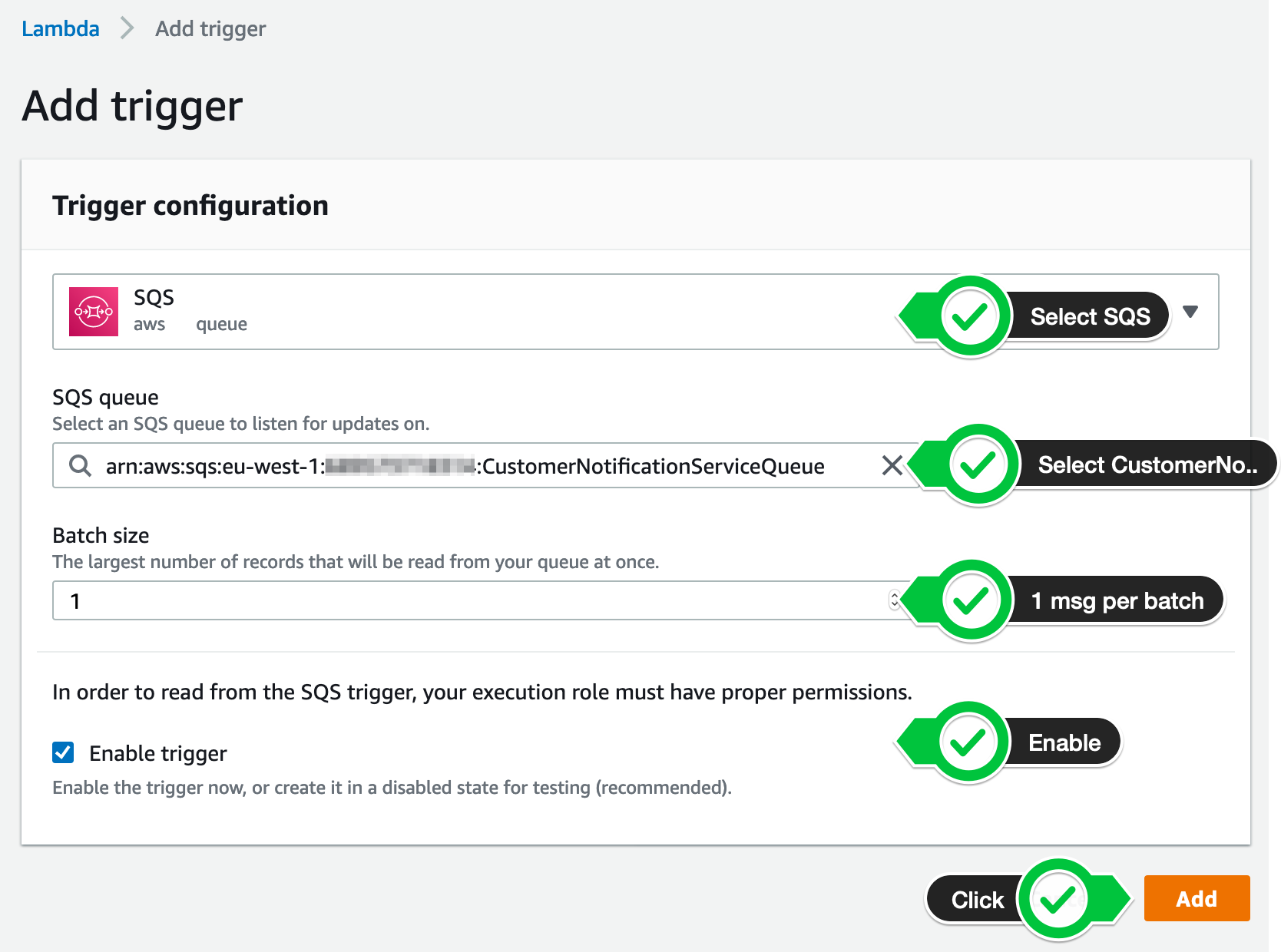
[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-8-console.png)

5. Add the Amazon SQS queue as event source for your Customer Notification Service AWS Lambda function

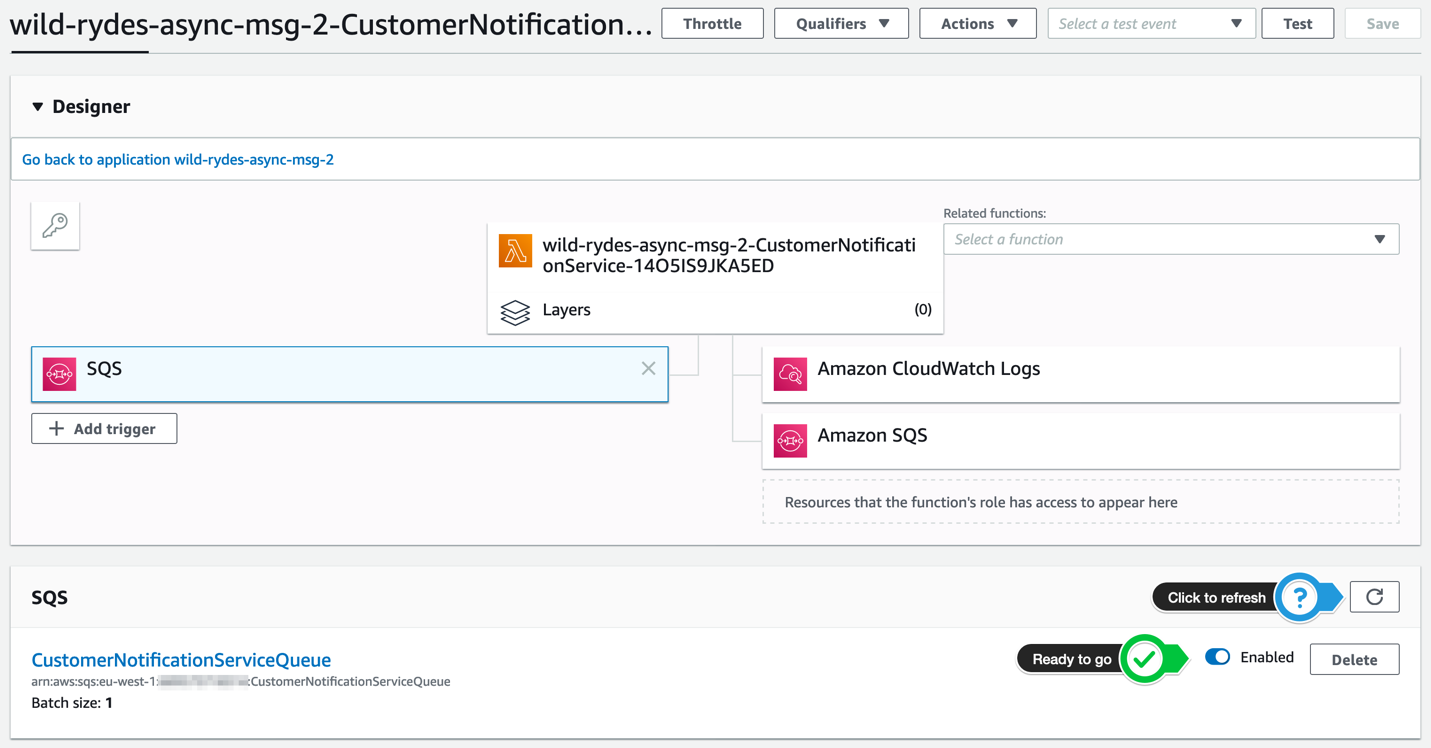
Open your [**AWS Lambda console**](https://console.aws.amazon.com/lambda/home?#/functions) and select **Functions** in the left navigation. Click on the function with the name **wild-rydes-async-msg-2-CustomerNotification…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name). Click on the **+ Add Trigger** button on the left side of the page:

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-9-console.png)

On the following page, select SQS as the event source for this AWS Lambda function. For the **SQS queue**, select the CustomerNotificationServiceQueue and set the **batch size** to 1. Don’t forget to **enable the trigger**, before you click the **Add** button in the lower right corner.

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-10-console.png)

After some seconds, the trigger will be enabled and and you are ready to go (you may have to refresh the site a few times).

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-notification-service-subscription/step-11-console.png)

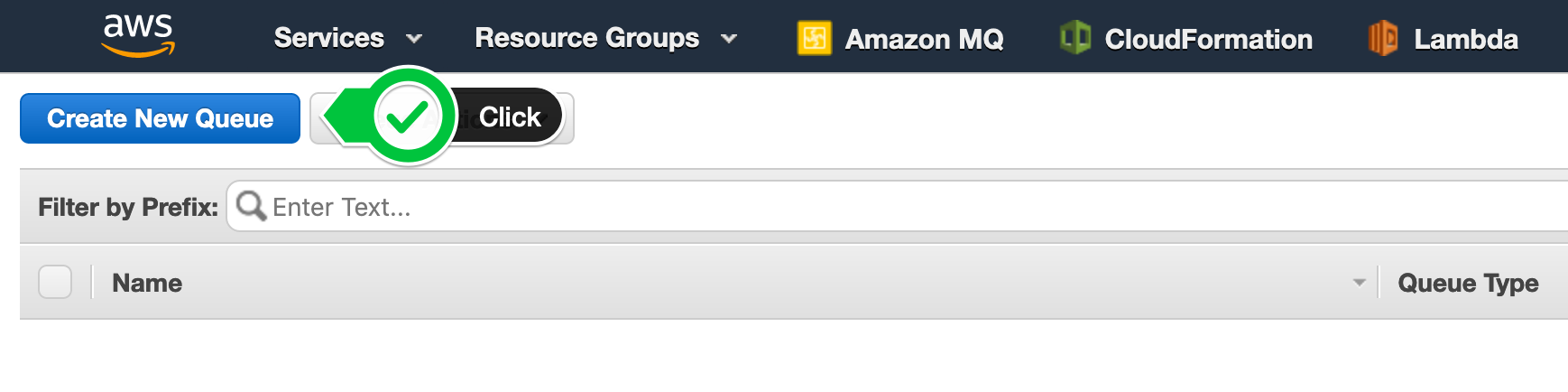
CREATE CUSTOMER ACCOUNTING SERVICE SUBSCRIPTION

In this step, we will create an Amazon SQS queue for the **Customer Accounting Service** and add a subscription to the Amazon SNS topic we created before:

#### **1. Create a new Amazon SQS queue**

In your [**Amazon SQS console**](https://console.aws.amazon.com/sqs/home?), select **Create New Queue** in top left corner or click **Get Started Now** in the center of the page, if it’s your first queue in this region.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-1-console.png)

Enter CustomerAccountingServiceQueue as **Queue Name**, make sure **Standard Queue** is highlighted and click **Create Queue**.

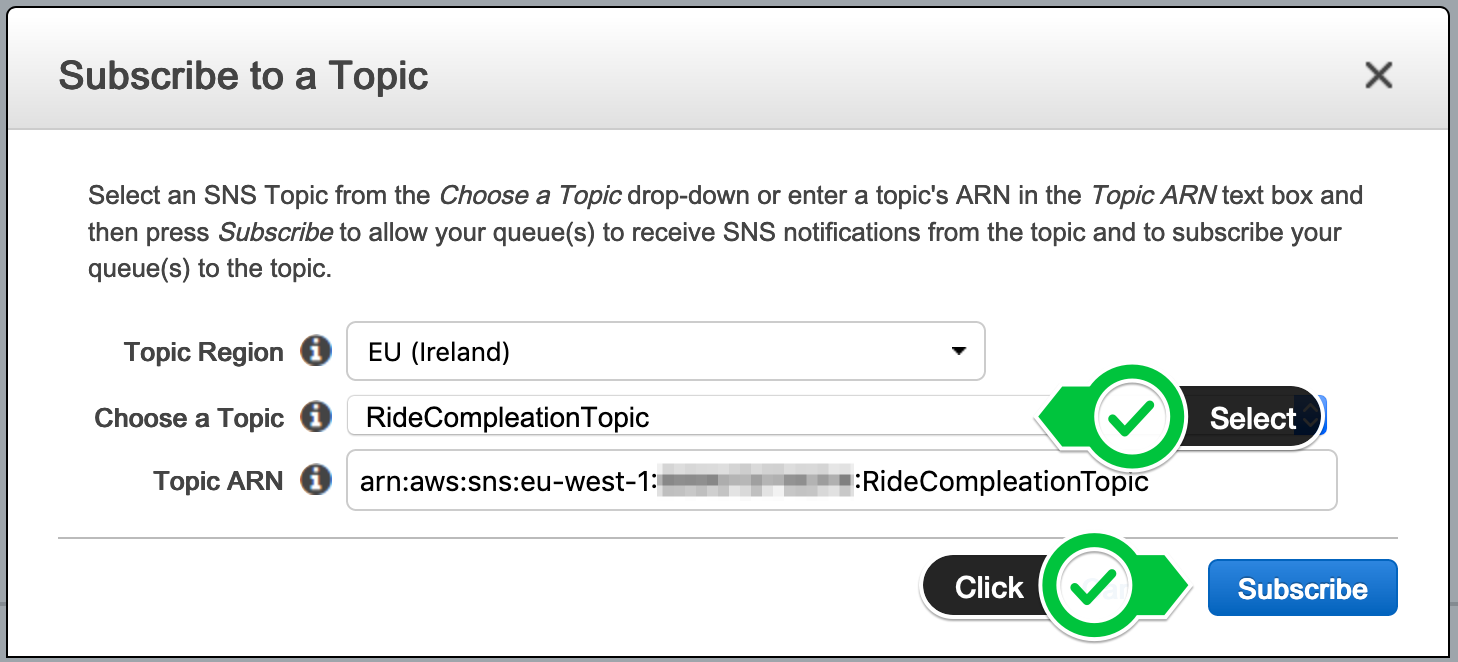
 Screenshot

#### **2. Create a new subscription**

After queue **CustomerAccountingServiceQueue is created**, select **Subscribe Queue to SNS Topic**.

In the pop up window, select the **RideCompletionTopic** and click **Subscribe**. Click **Ok** in the confirmation dialog.

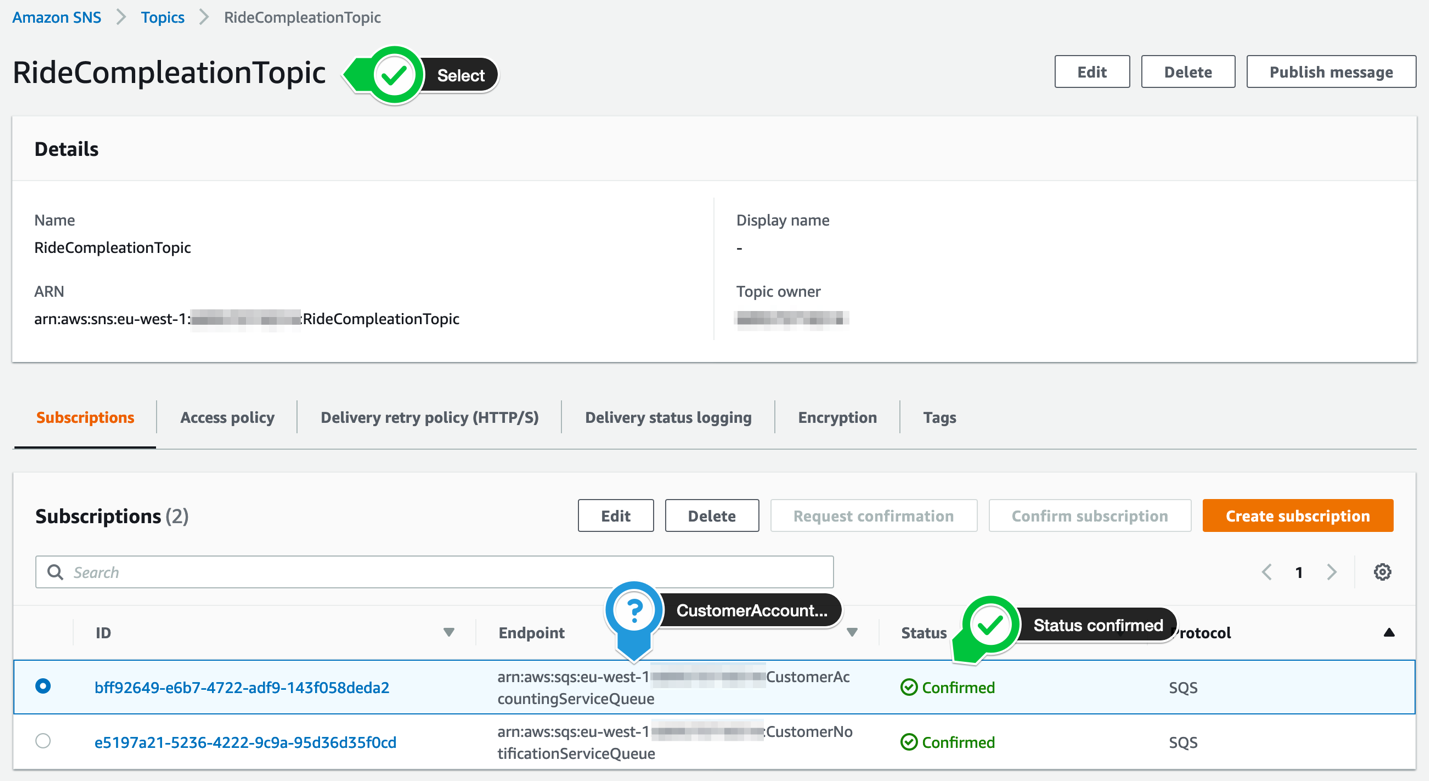
 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-4-console.png)

#### **3. Validate the subscription confirmation**

Browse to your [**Amazon SNS console**](https://console.aws.amazon.com/sns/v3/home?#/topics) to list your existing topics. Select the **RideCompletionTopic** and verify, the subscription has the status **Confirmed**.

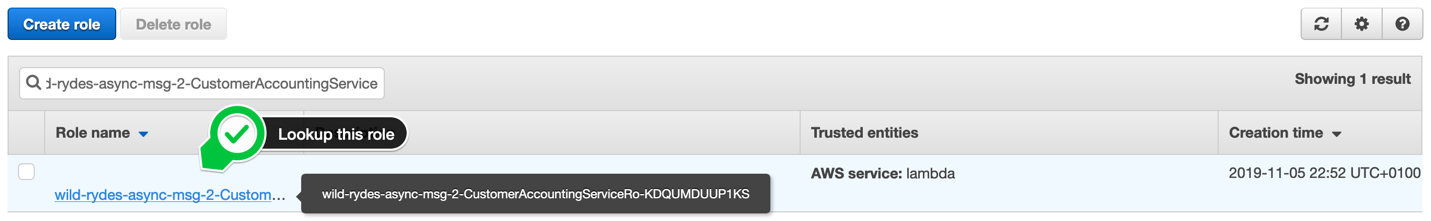
 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-5-console.png)

#### **4. Grant permissions to our function to access the Amazon SQS queue**

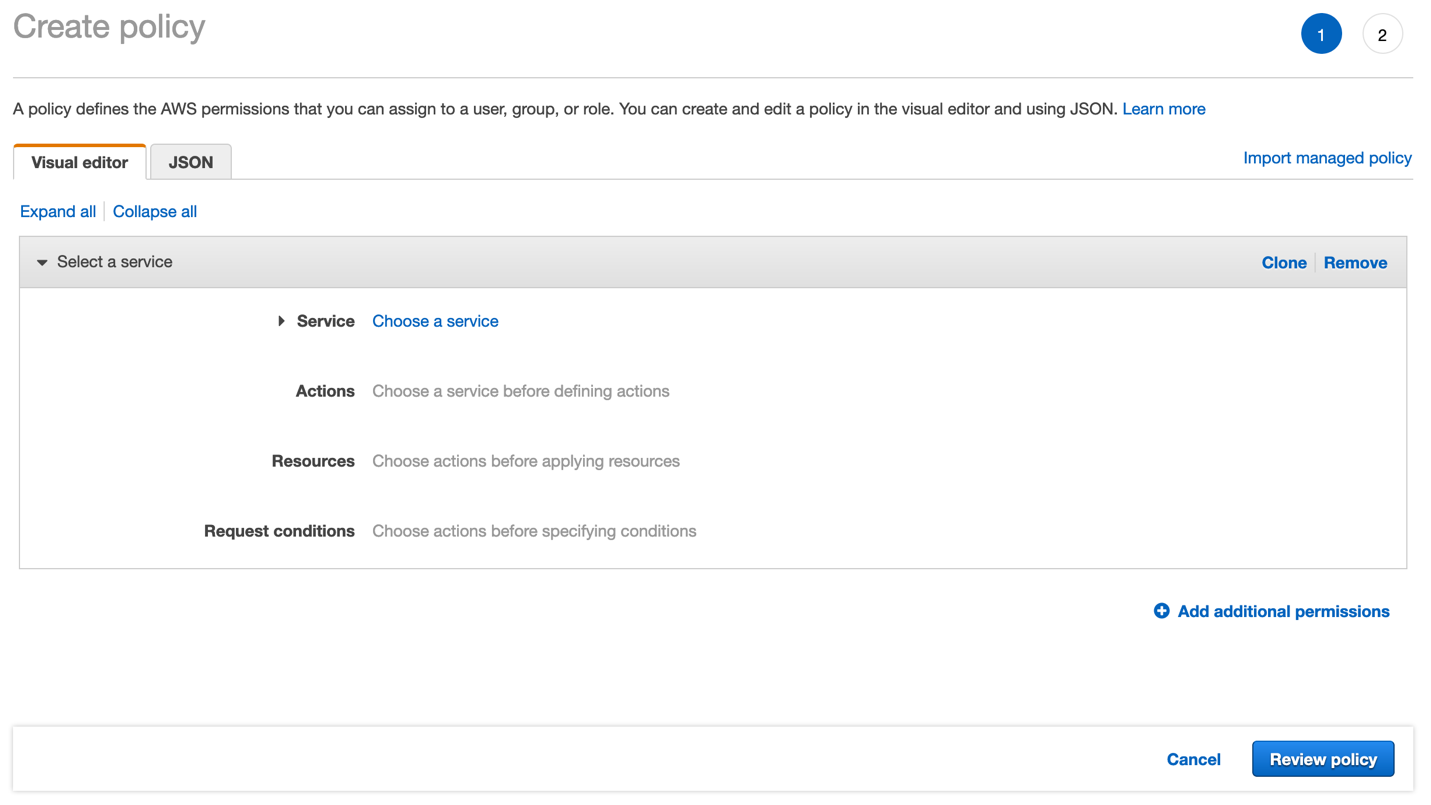
In your [**Amazon IAM console**](https://console.aws.amazon.com/iam), select **Roles** in the left navigation. Use the filter text box to find the role with the name **wild-rydes-async-msg-2-CustomerAccountingService-…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name).

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-6-console.png)

Click on the role name and click **Add inline policy** to attache another one.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-7-console.png)

Select the **JSON** tab and passed the following policy statement into it, after you have substitute «…» with the correct values. It will add the permission to your Lambda function to access the Amazon SQS queue:

 policy

{

"Statement": [

{

"Effect": "Allow",

"Action": [

"sqs:ChangeMessageVisibility",

"sqs:ChangeMessageVisibilityBatch",

"sqs:DeleteMessage",

"sqs:DeleteMessageBatch",

"sqs:GetQueueAttributes",

"sqs:ReceiveMessage"

],

"Resource": "arn:aws:sqs:<<AWS REGION>>:<<AWS ACCOUNT ID>>:CustomerAccountingServiceQueue"

}

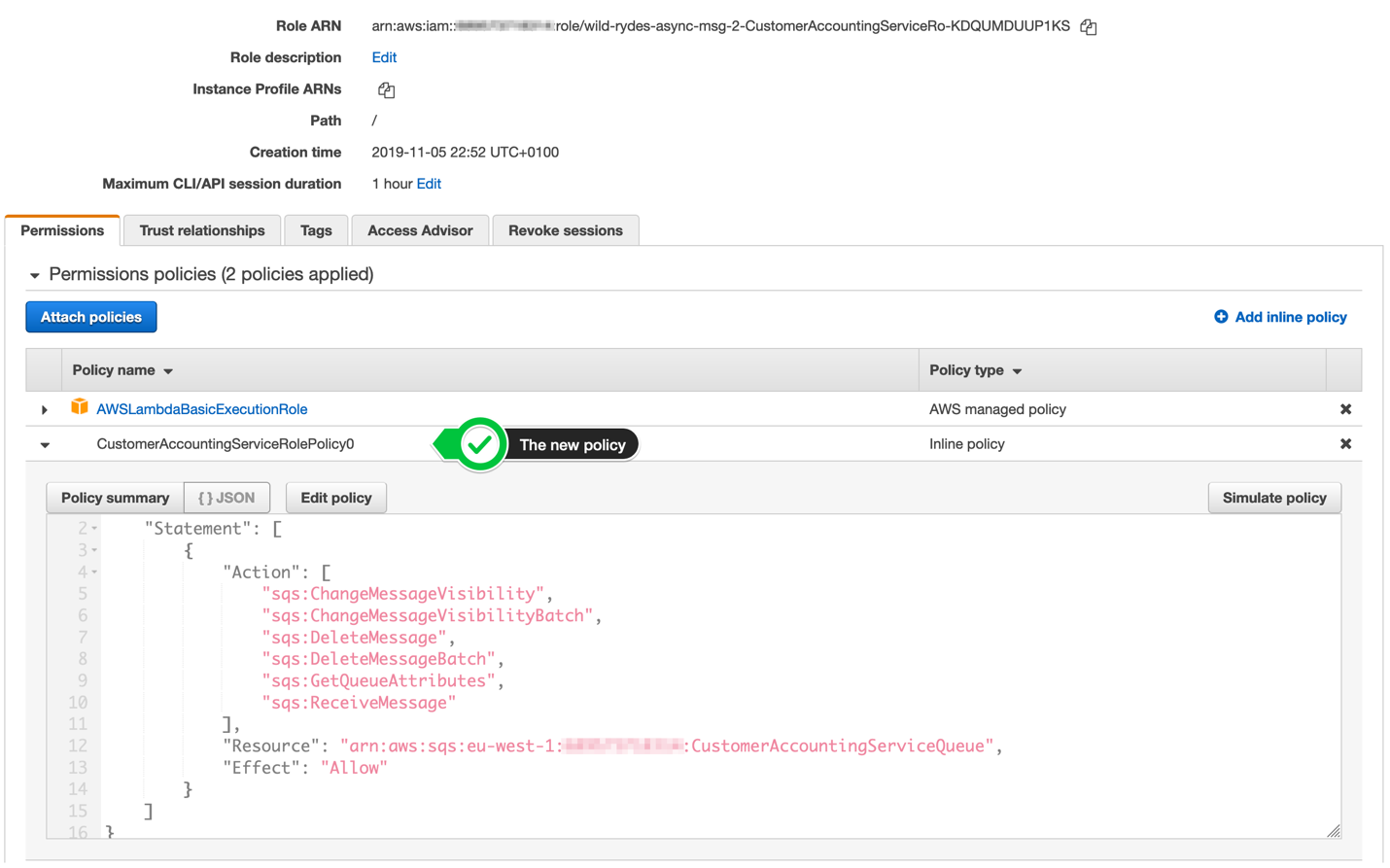
]

}

Make sure you provide the AWS ACCOUNT ID in the form of XXXXXXXXXXXX and not XXXX-XXXX-XXXX!

Click **Review policy** and enter the **Name** CustomerAccountingServiceRolePolicy0. Click **Create policy**. To validate this step, select on the role again and your should see 3 policies attached to your role, including the one you just have created:

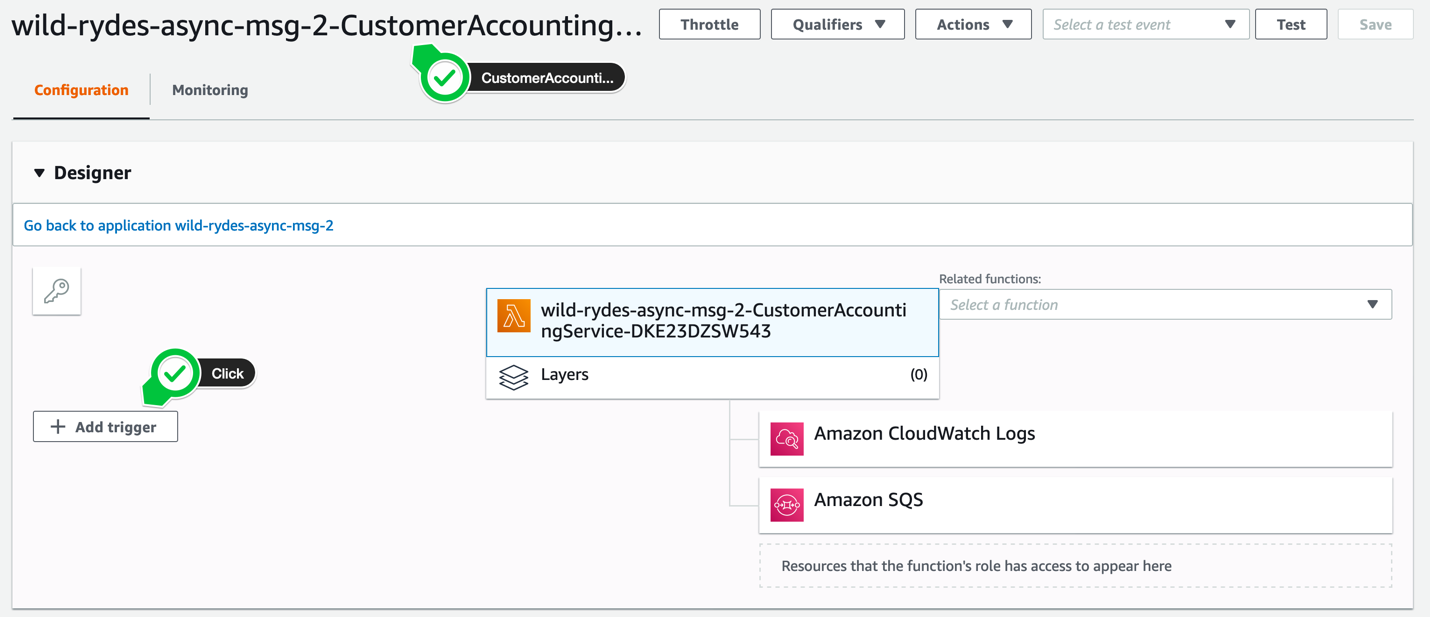
 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-8-console.png)

#### **5. Add the Amazon SQS queue as event source for your Customer Notification Service AWS Lambda function**

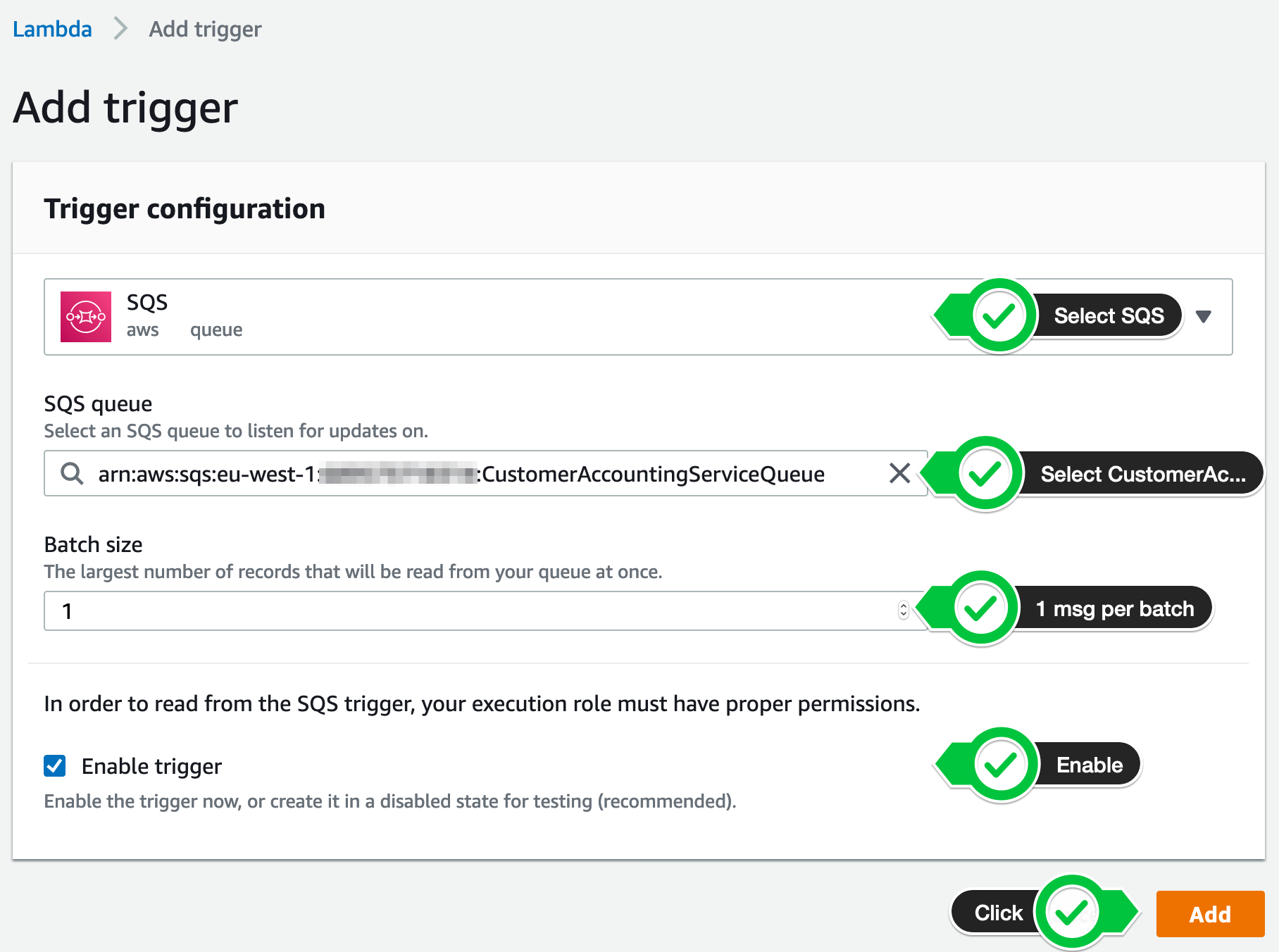
Open your [**AWS Lambda console**](https://console.aws.amazon.com/lambda/home?#/functions) and select **Functions** in the left navigation. Click on the function with the name **wild-rydes-async-msg-2-CustomerAccounting…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name). Click on the **+ Add Trigger** button on the left side of the page:

 Detailed description

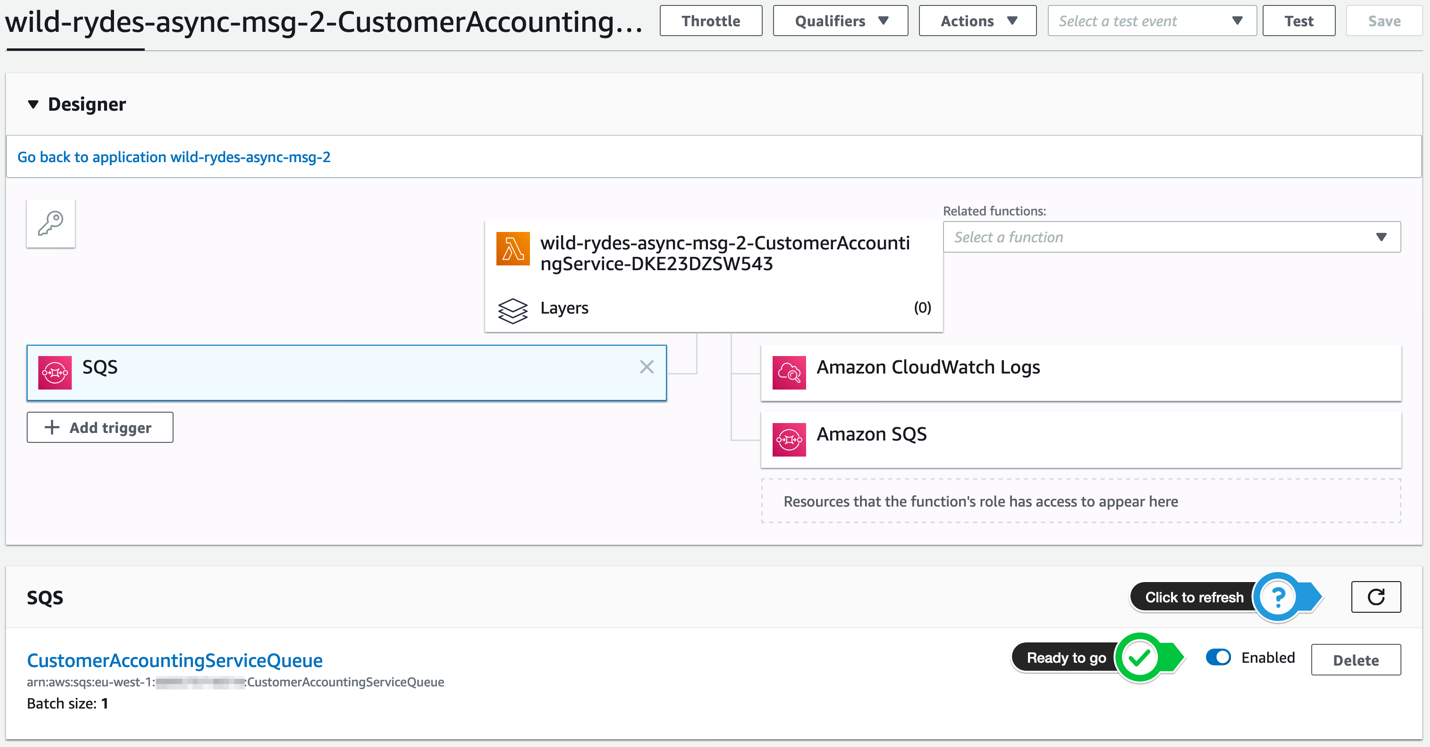
[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-9-console.png)

On the following page, select SQS as the event source for this AWS Lambda function. For the **SQS queue**, select the CustomerAccountingServiceQueue and set the **batch size** to 1. Don’t forget to **enable the trigger**, before you click the **Add** button in the lower right corner.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-10-console.png)

After some seconds, the trigger will be enabled and and you are ready to go (you may have to refresh the site a few times).

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-customer-accounting-service-subscription/step-11-console.png)

# **CREATE EXTRAORDINARY RIDES SERVICE SUBSCRIPTION**

In this step, we will create an Amazon SQS queue for the **Extraordinary Rides Service** and add a subscription to the Amazon SNS topic we created before:

#### **1. Create a new Amazon SQS queue**

In your [**Amazon SQS console**](https://console.aws.amazon.com/sqs/home?), select **Create New Queue** in top left corner or click **Get Started Now** in the center of the page, if it’s your first queue in this region.

Enter ExtraordinaryRidesServiceQueue as **Queue Name**, make sure **Standard Queue** is highlighted and click **Quick Queue**.

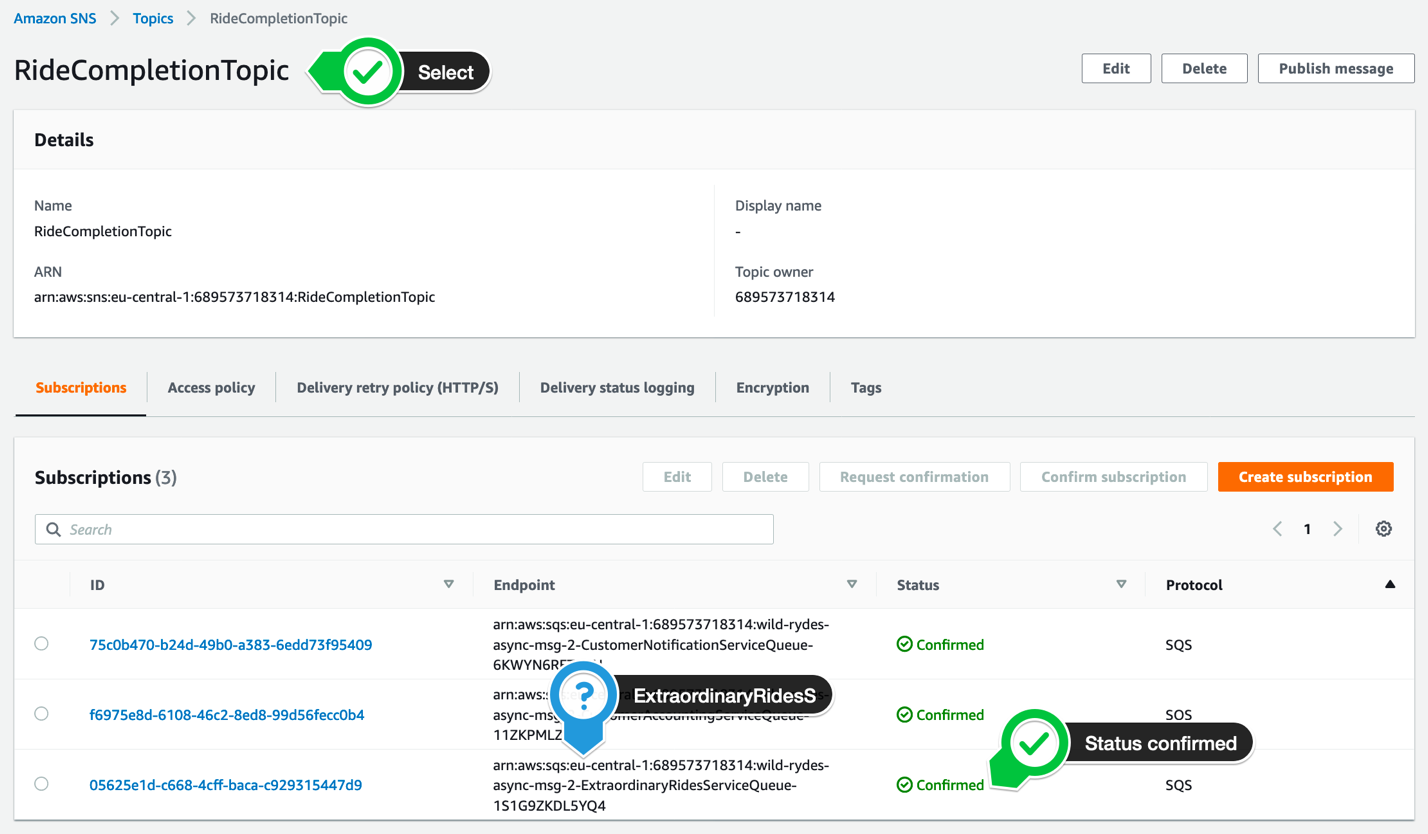
#### **2. Create a new subscription**

After the **ExtraordinaryRidesServiceQueue** is created, click **Subscribe to SNS Topic**.

In the pop up window, select the **RideCompletionTopic** and click **Subscribe**. Click **Save** in the confirmation dialog.

#### **3. Validate the subscription confirmation**

Browse to your [**Amazon SNS console**](https://console.aws.amazon.com/sns/v3/home?#/topics) to list your existing topics. Select the **RideCompletionTopic** and verify, the subscription has the status **Confirmed**.

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-5-console.png)

#### **4. Add the filter to the subscription**

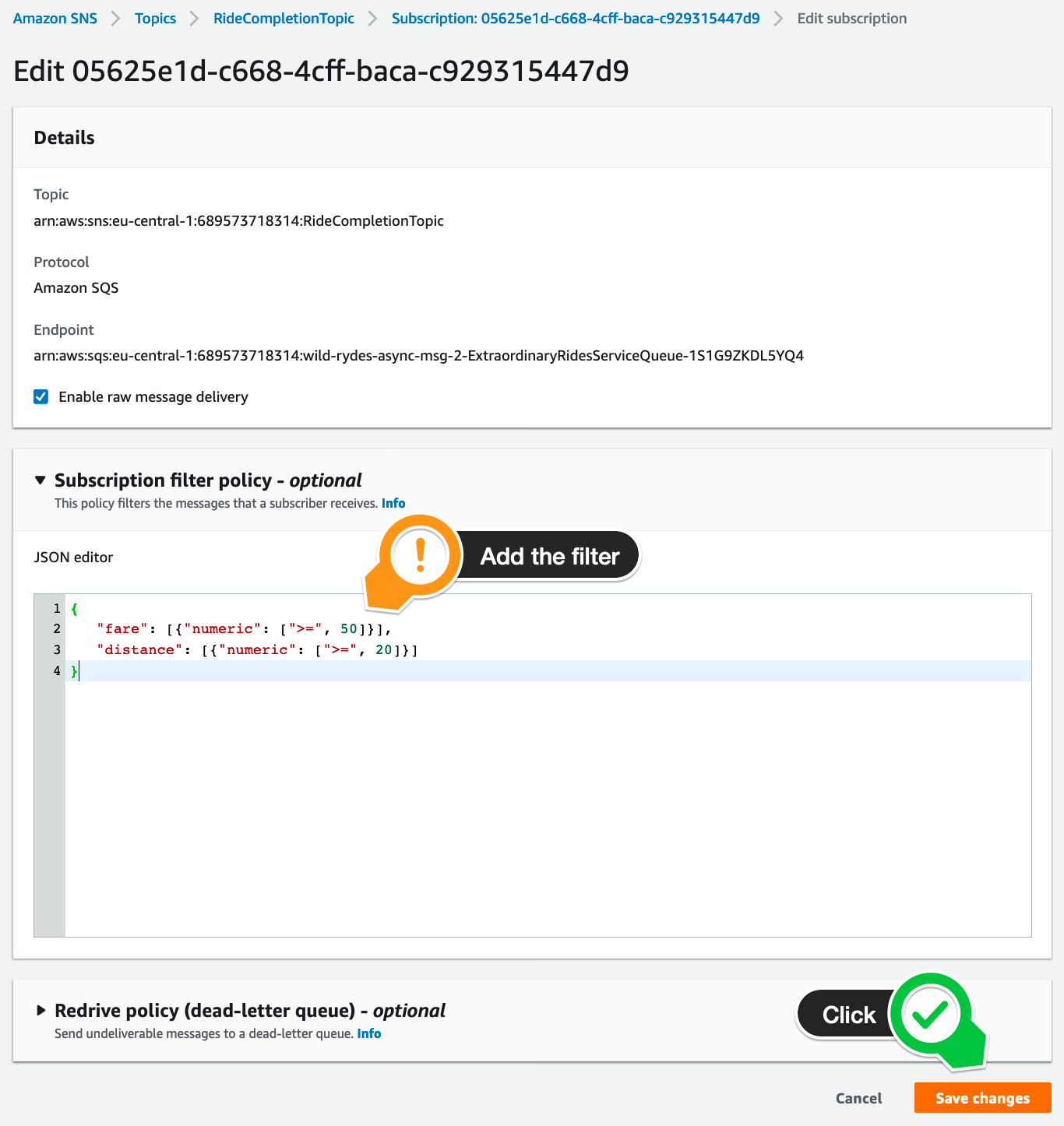
Click on the subscription id for the **ExtraordinaryRidesServiceQueue subscription** and click on **Edit** to add our subscription filter policy. In the subscription filter policy section, add out filter policy. Click **Save changes**.

{

"fare": [{"numeric": [">=", 50]}],

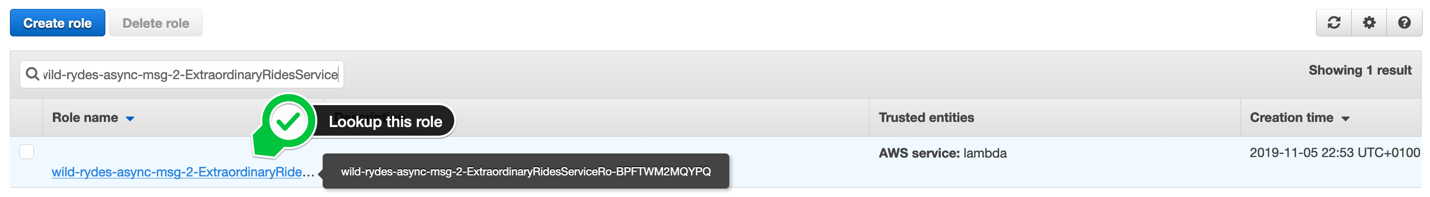
"distance": [{"numeric": [">=", 20]}]

}

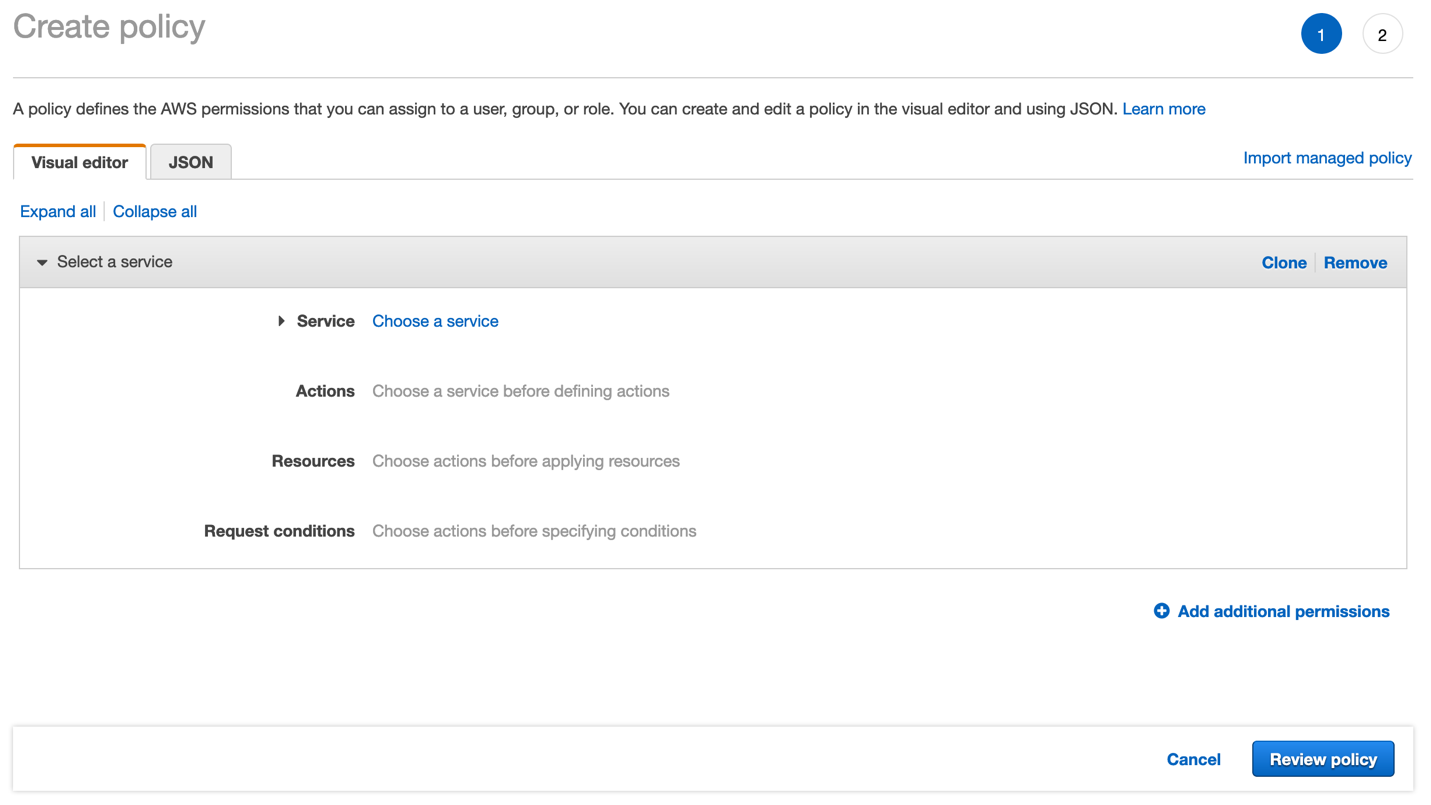
[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-6-console.png)

5. Grant permissions to our function to access the Amazon SQS queue

In your [**Amazon IAM console**](https://console.aws.amazon.com/iam), select **Roles** in the left navigation. Use the filter text box to find the role with the name **wild-rydes-async-msg-2-ExtraordinaryRidesService-…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name).

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-7-console.png)

Click on the role name and click **Add inline policy** to attach another one.

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-8-console.png)

Select the **JSON** tab and passed the following policy statement into it, after you have substitute «…» with the correct values. It will add the permission to your Lambda function to access the Amazon SQS queue:

 policy

{

"Statement": [

{

"Effect": "Allow",

"Action": [

"sqs:ChangeMessageVisibility",

"sqs:ChangeMessageVisibilityBatch",

"sqs:DeleteMessage",

"sqs:DeleteMessageBatch",

"sqs:GetQueueAttributes",

"sqs:ReceiveMessage"

],

"Resource": "arn:aws:sqs:<<AWS REGION>>:<<AWS ACCOUNT ID>>:ExtraordinaryRidesServiceQueue"

}

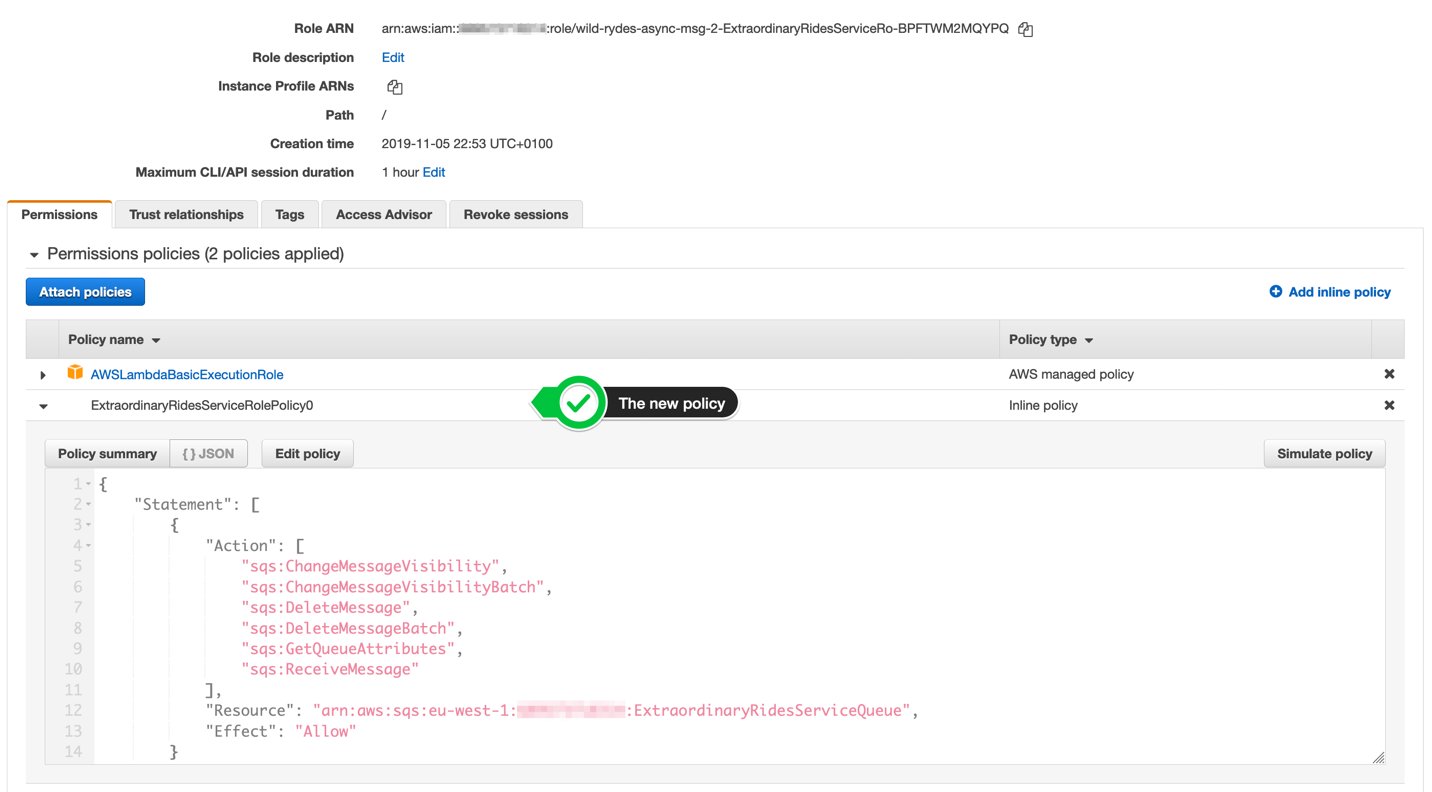
]

}

Make sure you provide the AWS ACCOUNT ID in the form of XXXXXXXXXXXX and not XXXX-XXXX-XXXX!

Click **Review policy** and enter the **Name** ExtraordinaryRidesServiceRolePolicy0. Click **Create policy**. To validate this step, select on the role again and your should see 3 policies attached to your role, including the one you just have created:

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-9-console.png)

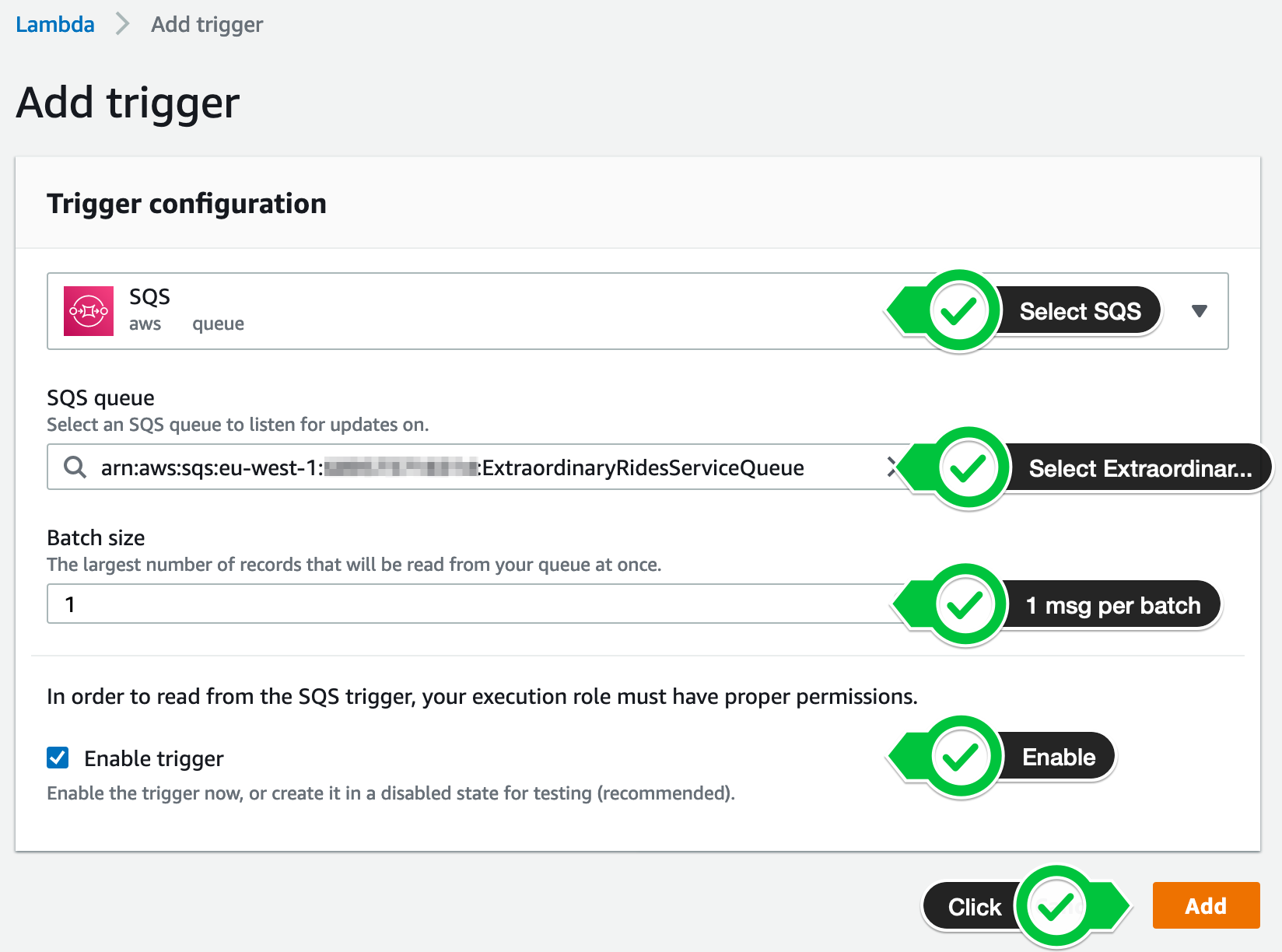
6. Add the Amazon SQS queue as event source for your Customer Notification Service AWS Lambda function

Open your [**AWS Lambda console**](https://console.aws.amazon.com/lambda/home?#/functions) and select **Functions** in the left navigation. Click on the function with the name **wild-rydes-async-msg-2-ExtraordinaryRides…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name). Click on the **+ Add Trigger** button on the left side of the page:

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-10-console.png)

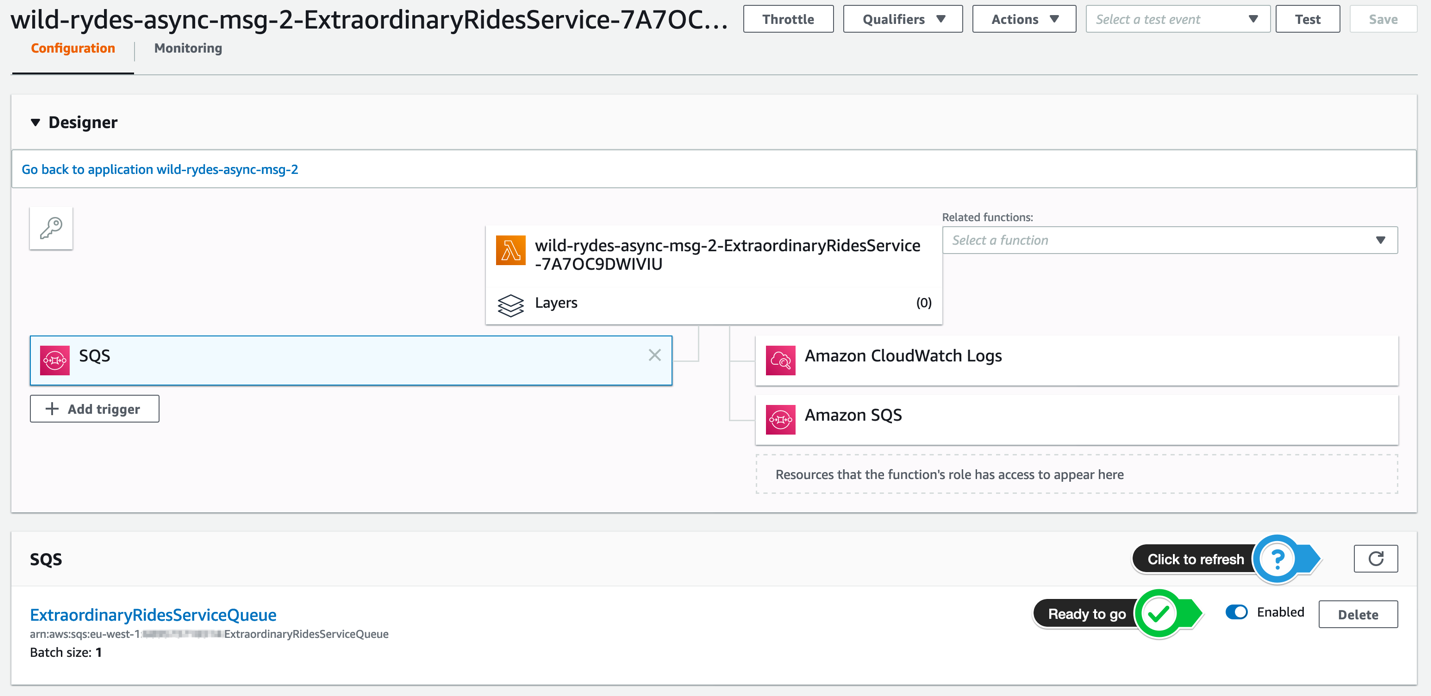
On the following page, select SQS as the event source for this AWS Lambda function. For the **SQS queue**, select the ExtraordinaryRidesServiceQueue and set the **batch size** to 1. Don’t forget to **enable the trigger**, before you click the **Add** button in the lower right corner.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-11-console.png)

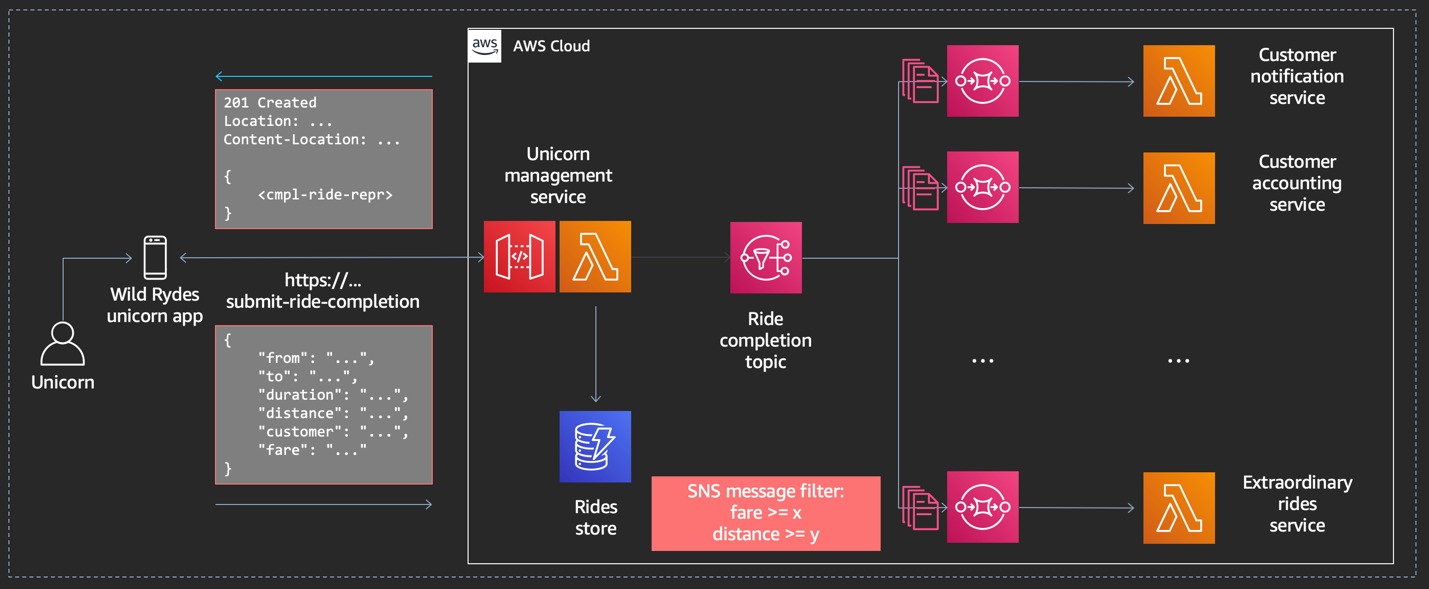
After some seconds, the trigger will be enabled and you are ready to go (you may have to refresh the site a few times).

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/create-extraordinary-rides-service-subscription/step-12-console.png)

UPDATE UNICORN MANAGEMENT SERVICE

After creating the Amazon SNS topic, all the Amazon SQS queues and the subscriptions, the current architecture looks like the following on:

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/update-unicorn-management-service/step-1.png)

The last missing part to complete the architecture is calling our **Amazon SNS topic** from our **Unicorn Management Service**

#### **1. Grant additional IAM permissions to Lambda**

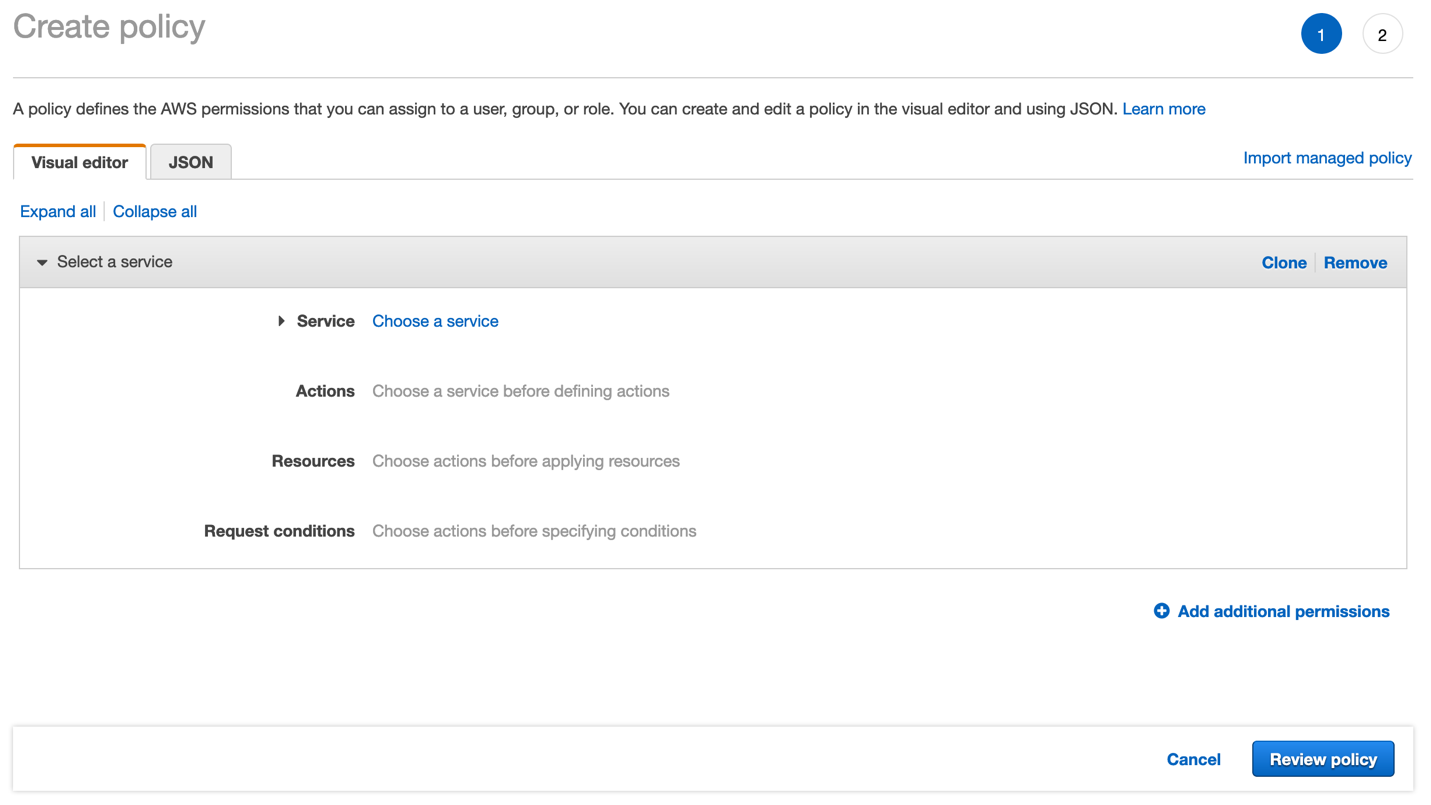
In your [**Amazon IAM console**](https://console.aws.amazon.com/iam), select **Roles** in the left navigation. Use the filter text box to find the role with the name **wild-rydes-async-msg-2-SubmitRideCompletionFunctio-…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name).

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/update-unicorn-management-service/step-1-console.png)

Click on the role name and click **Add inline policy** to attache another one.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/update-unicorn-management-service/step-2-console.png)

Select the **JSON** tab and passed the following policy statement into it, after you have substitute «…» with the correct values. It will add the permission to your Lambda function to publish messages to this particular Amazon SNS topic:

 Policy

{

"Version": "2012-10-17",

"Statement": [

{

"Action": [

"sns:Publish"

],

"Resource": "arn:aws:sns:<<AWS REGION>>:<<AWS ACCOUNT ID>>:<<SNS TOPIC NAME>>",

"Effect": "Allow"

}

]

}

Make sure you provide the AWS ACCOUNT ID in the form of XXXXXXXXXXXX and not XXXX-XXXX-XXXX!

Click **Review policy** and enter the **Name** SubmitRideCompletionFunctionRolePolicy1. Click **Create policy**. To validate this step, select on the role again and you should see 3 policies attached to your role, including the one you just have created:

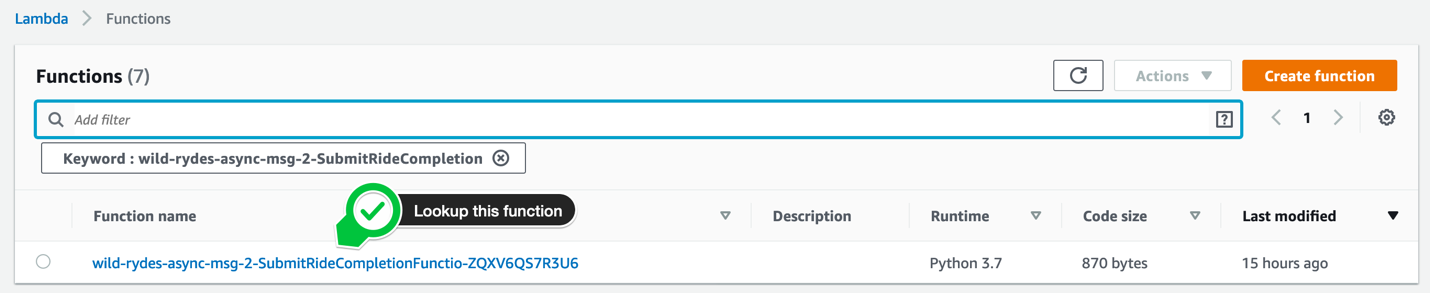
 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/update-unicorn-management-service/step-3-console.png)

#### **2. Provide the Amazon SNS topic ARN to Lambda**

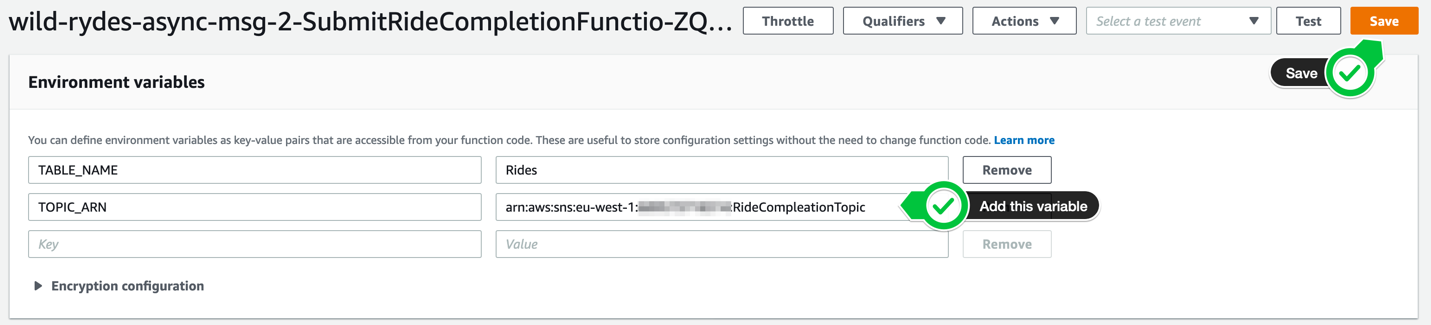
In your [**AWS Lambda console**](https://console.aws.amazon.com/lambda/home?#/functions), select **Functions** in the left navigation. Use the filter text box to find the function with the name **wild-rydes-async-msg-2-SubmitRideCompletionFunctio-…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name).

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/update-unicorn-management-service/step-4-console.png)

Click on the function name and scroll down to the section **Environment variables**. Our Lambda function expects an environment variable with the **Name** TOPIC\_ARN. It uses this Amazon SNS topic to publish all messages to. Lookup your Amazon SNS topic name in the [Amazon SNS console](https://console.aws.amazon.com/sns) and add this variable. Click the **Save** button in the top right corner to save the change.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/update-unicorn-management-service/step-5-console.png)

#### **3. Update your Lambda function to call Amazon SNS**

Open your [**AWS Lambda console**](https://console.aws.amazon.com/lambda/home?#/functions) and select **Functions** in the left navigation. Select the function with the name **wild-rydes-async-msg-2-SubmitRideCompletionFunctio-…** (assuming your have chosen wild-rydes-async-msg-2 as your stack name). Scroll a bit down to the section **Function code**. Add the definition of the sns client directly after the dynamodb client:

 Cheat Sheet

sns = boto3.client('sns', config=config)

After the put item DynamoDB statement and before we are sending the response back to the caller, add the code to publish a message to Amazon SNS:

 Cheat Sheet

response = sns.publish(

TopicArn=TOPIC\_ARN,

Message=json.dumps(request),

MessageAttributes = {

'fare': {

'DataType': 'Number',

'StringValue': str(request['fare'])

},

'distance': {

'DataType': 'Number',

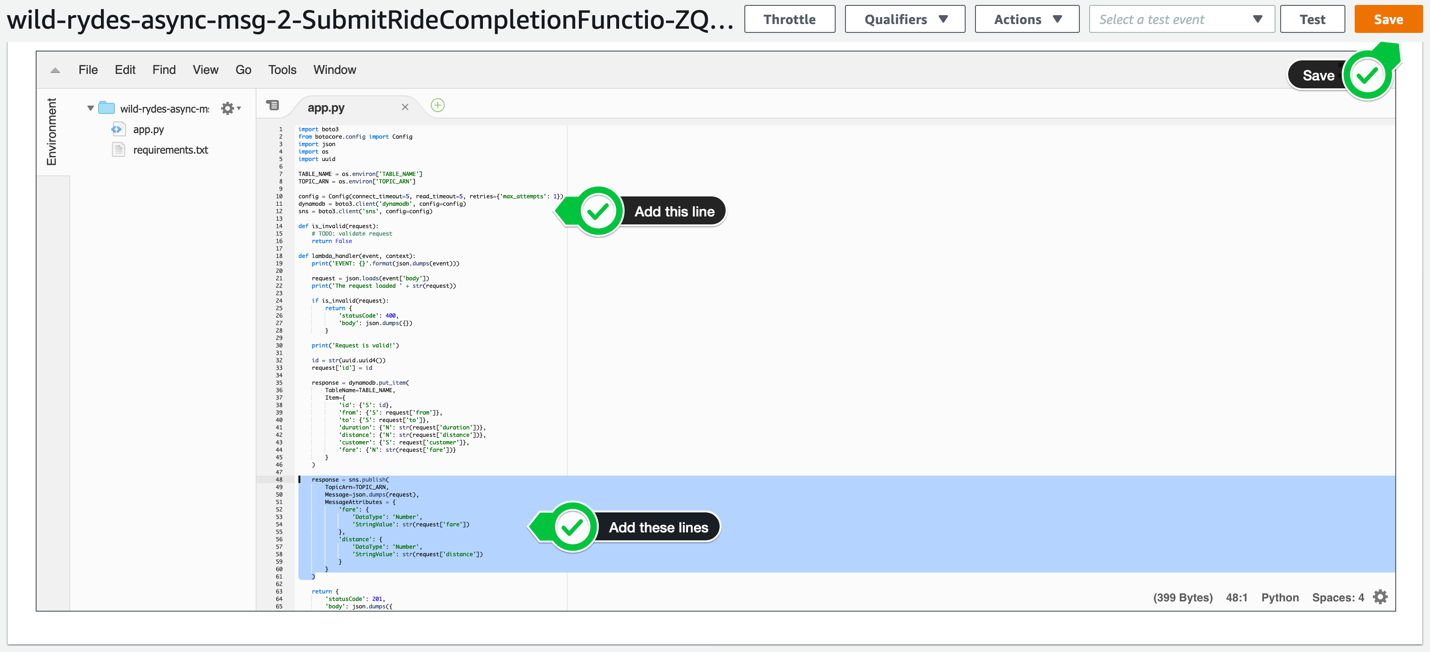
'StringValue': str(request['distance'])

}

}

)

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/update-unicorn-management-service/step-6-console.png)

# **TEST TOPIC-QUEUE CHAINING & LOAD BALANCING**

In this step, we will validate that the Amazon SNS topic is publishing all messages to all subscribers. Because a subscriber can also fail processing a message, we also want to validate that Amazon SNS is redelivering the message, so that we will not miss a single message.

#### **1. Look up the API Gateway endpoint**

To look-up the API Gateway endpoint URL for the submit-ride-completion function, run the following command in your Cloud9 instance.

aws cloudformation describe-stacks \

--stack-name wild-rydes-async-msg-2 \

--query 'Stacks[].Outputs[?OutputKey==`UnicornManagementServiceApiSubmitRideCompletionEndpoint`].OutputValue' \

--output text

#### **2. Send a couple requests to the Unicorn Management Service**

Let’s store this API Gateway endpoint URL in an environment variable, so we don’t have to repeat it all the time:

export ENDPOINT=$(aws cloudformation describe-stacks \

--stack-name wild-rydes-async-msg-2 \

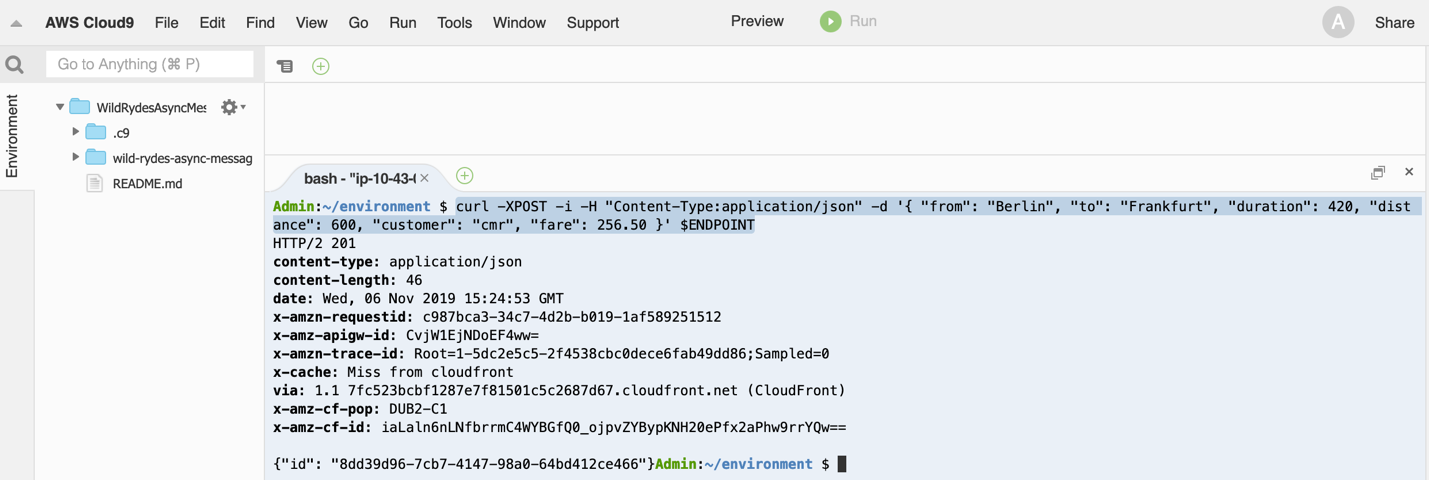
--query 'Stacks[].Outputs[?OutputKey==`UnicornManagementServiceApiSubmitRideCompletionEndpoint`].OutputValue' \

--output text)

To send a couple requests to the **submit ride completion endpoint**, execute the command below 5 or more times and change the request payload to test the filter criteria for the **Extraordinary Rides Service**:

curl -XPOST -i -H "Content-Type:application/json" -d '{ "from": "Berlin", "to": "Frankfurt", "duration": 420, "distance": 600, "customer": "cmr", "fare": 256.50 }' $ENDPOINT

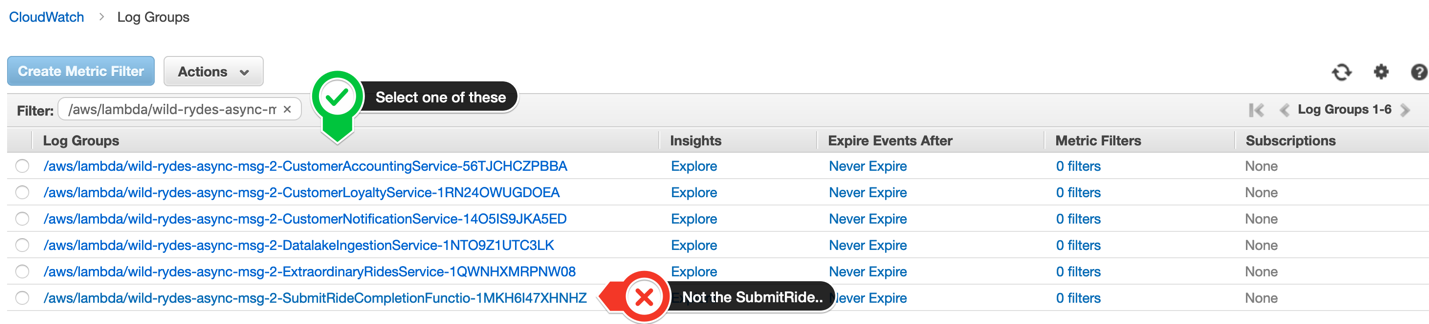
 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/test-topic-queue-chaining-and-load-balancer/step-1.png)

#### **3. Validate the message reception**

Go to your [Amazon CloudWatch Log console](https://console.aws.amazon.com/cloudwatch/home?#logs:prefix=/aws/lambda/wild-rydes-async-msg-2) and lookup all **Log Groups** with the prefix /aws/lambda/wild-rydes-async-msg-2.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/test-topic-queue-chaining-and-load-balancer/step-2.png)

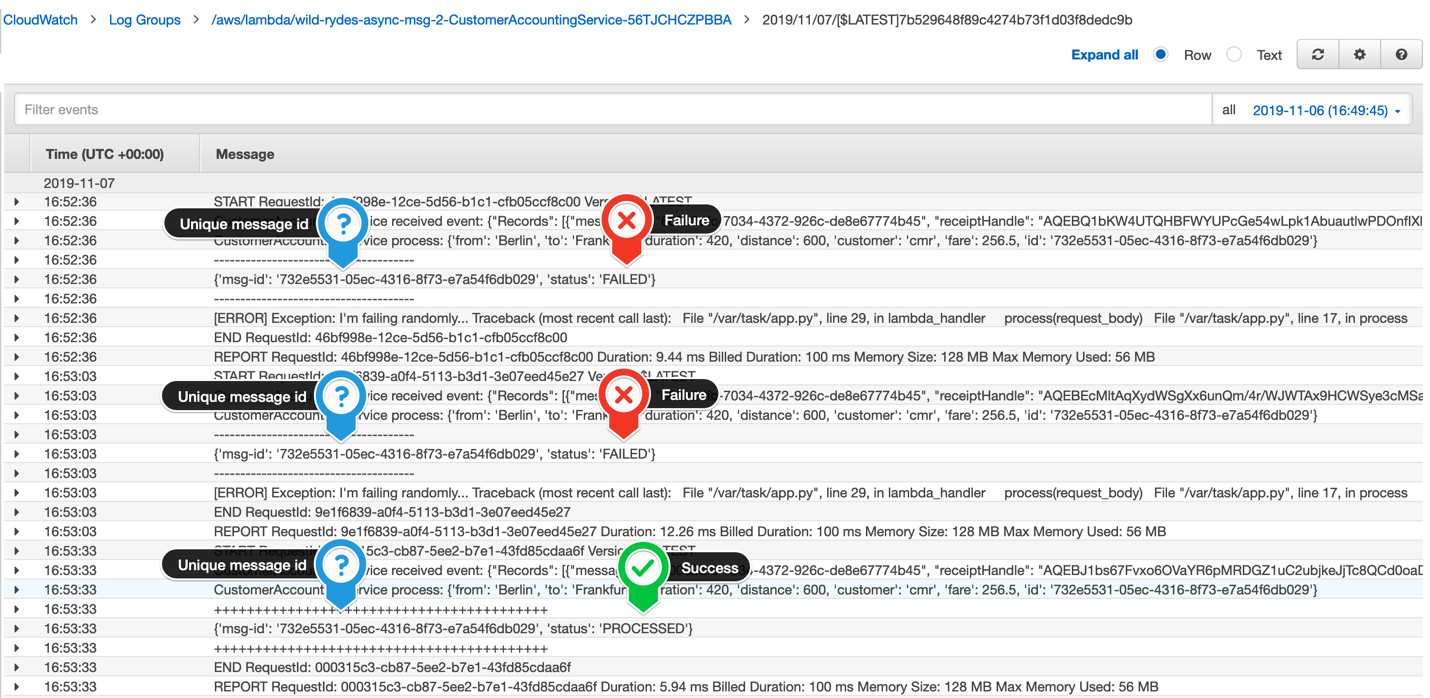
Click one the name of the Log Groups to see all **Log Streams** available for this Log Group.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/test-topic-queue-chaining-and-load-balancer/step-3.png)

Browse the most recent Log Streams to validate, that it could successfully process the message. You should also see some log entries, indicating a failed message processing. Shortly after, you should see the message redelivery from Amazon SNS and the successful message processing log entry.

 Detailed description

[](https://async-messaging.workshop.aws/topic-queue-chaining-and-load-balancer/test-topic-queue-chaining-and-load-balancer/step-4.png)

Browse all Log Groups to validate, that each of our 5 backend service could successfully process the message.