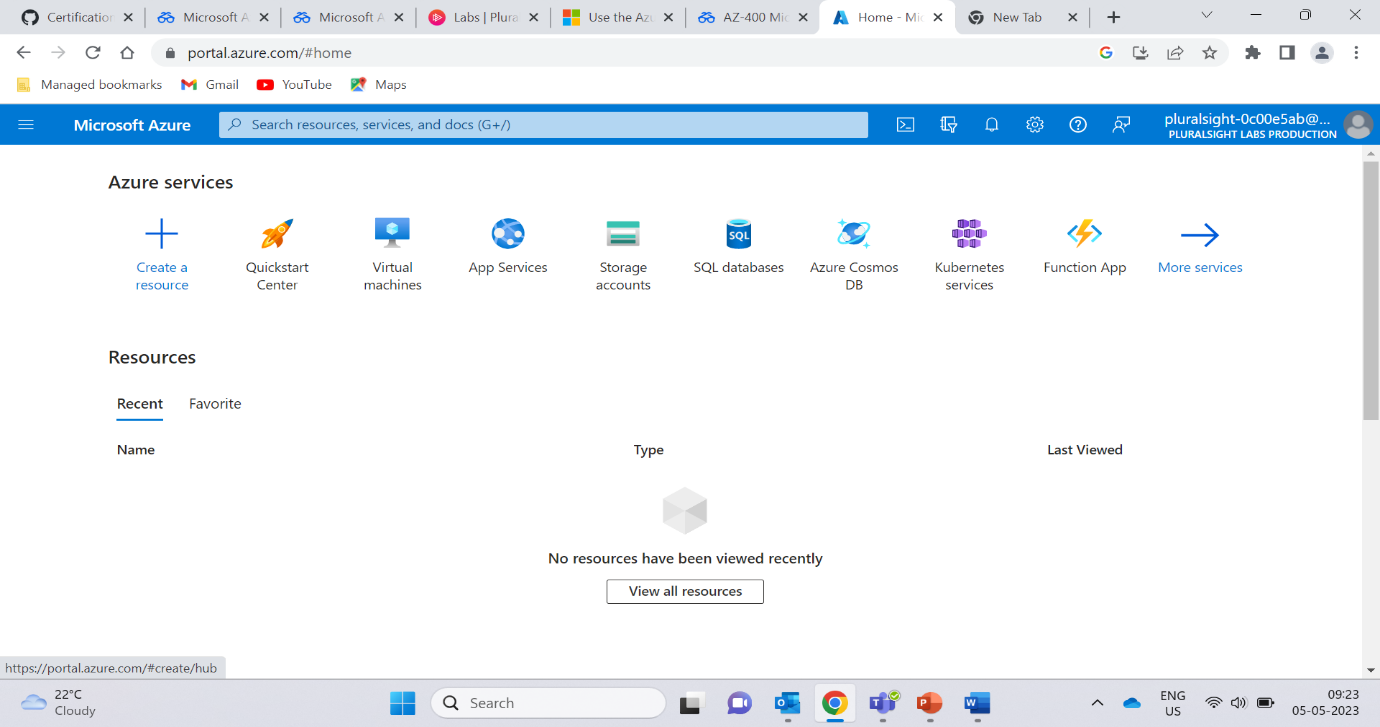
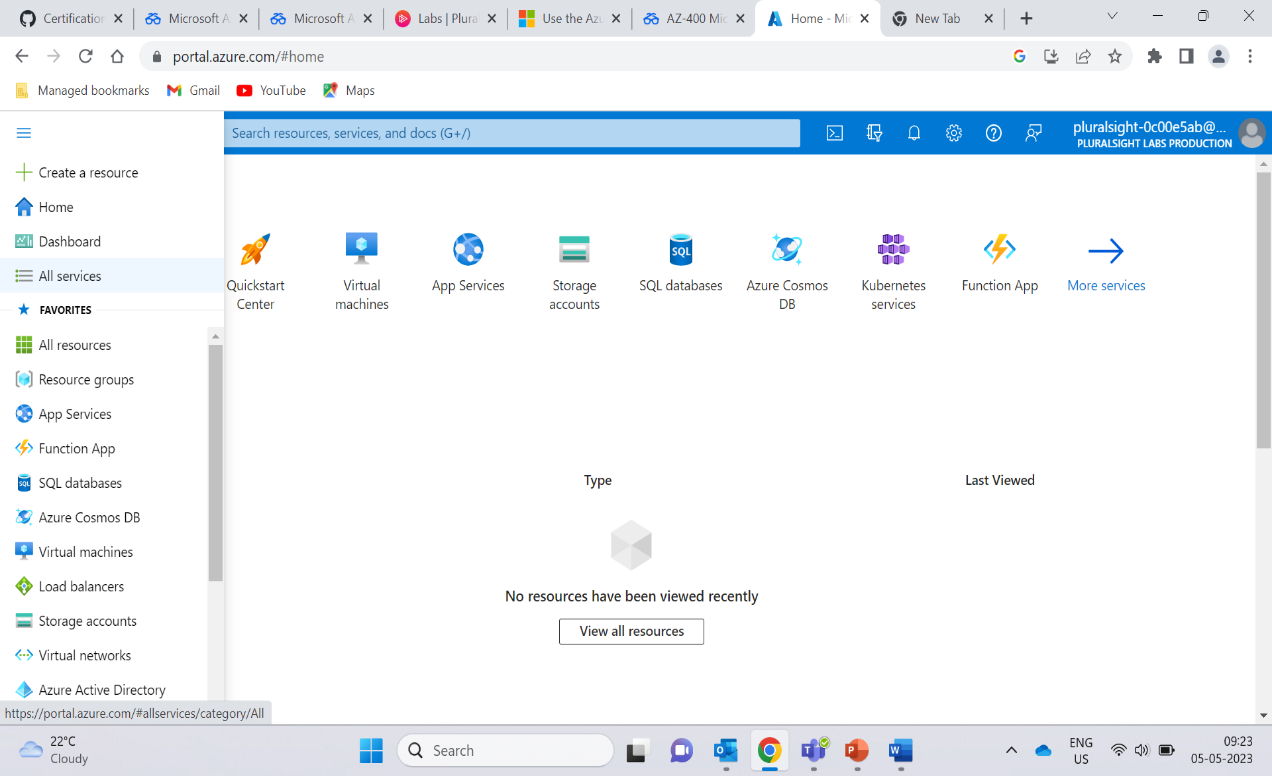
**Create a Service Bus Queue**

1. Log in to the Azure portal using the temporary credentials and link provide

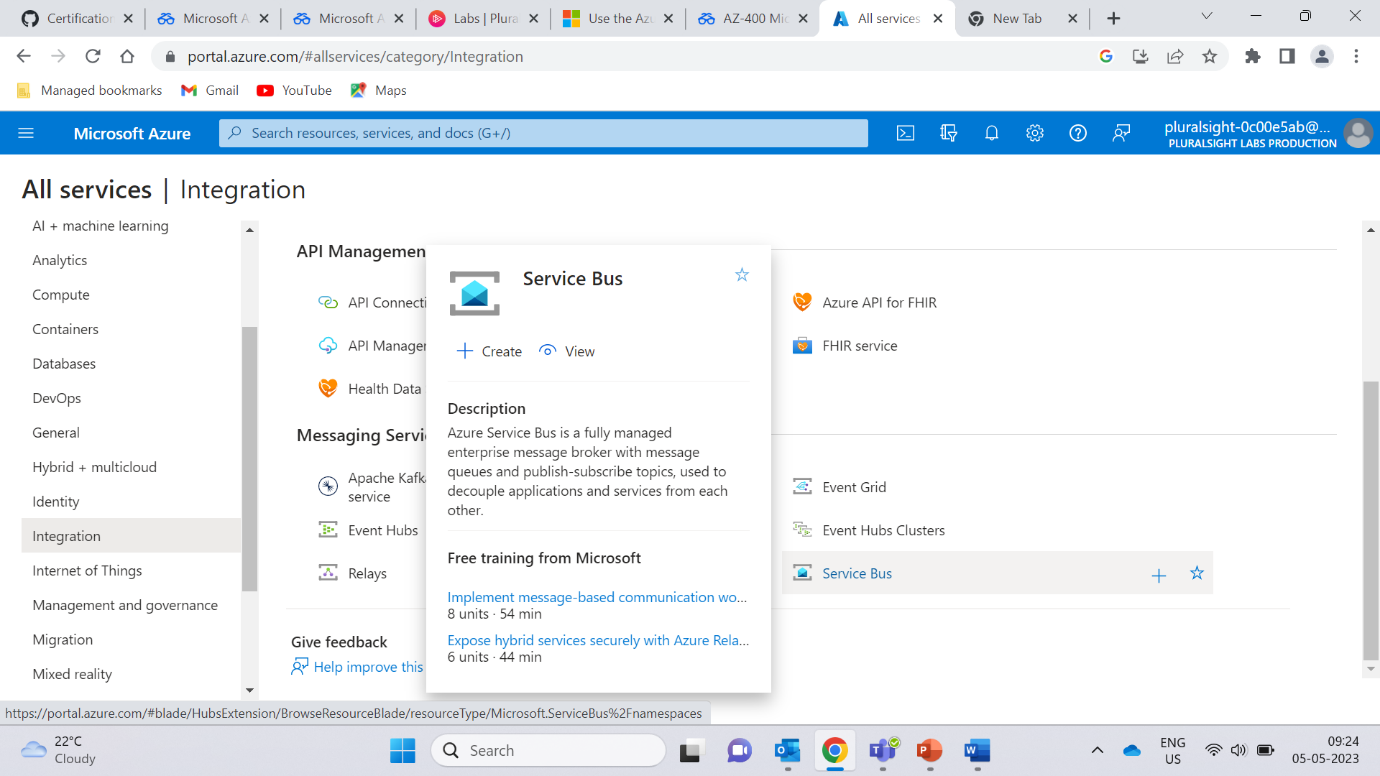


1. Click on the menu button, ☰, in the upper-left corner and select **All services**.

This will show an overview of all the services Microsoft Azure provides, organized by categories.



1. Select **Integration** on the left sidebar under **Categories**.



1. Select **Service Bus** from the list.
2. Click the **+ Create** button.



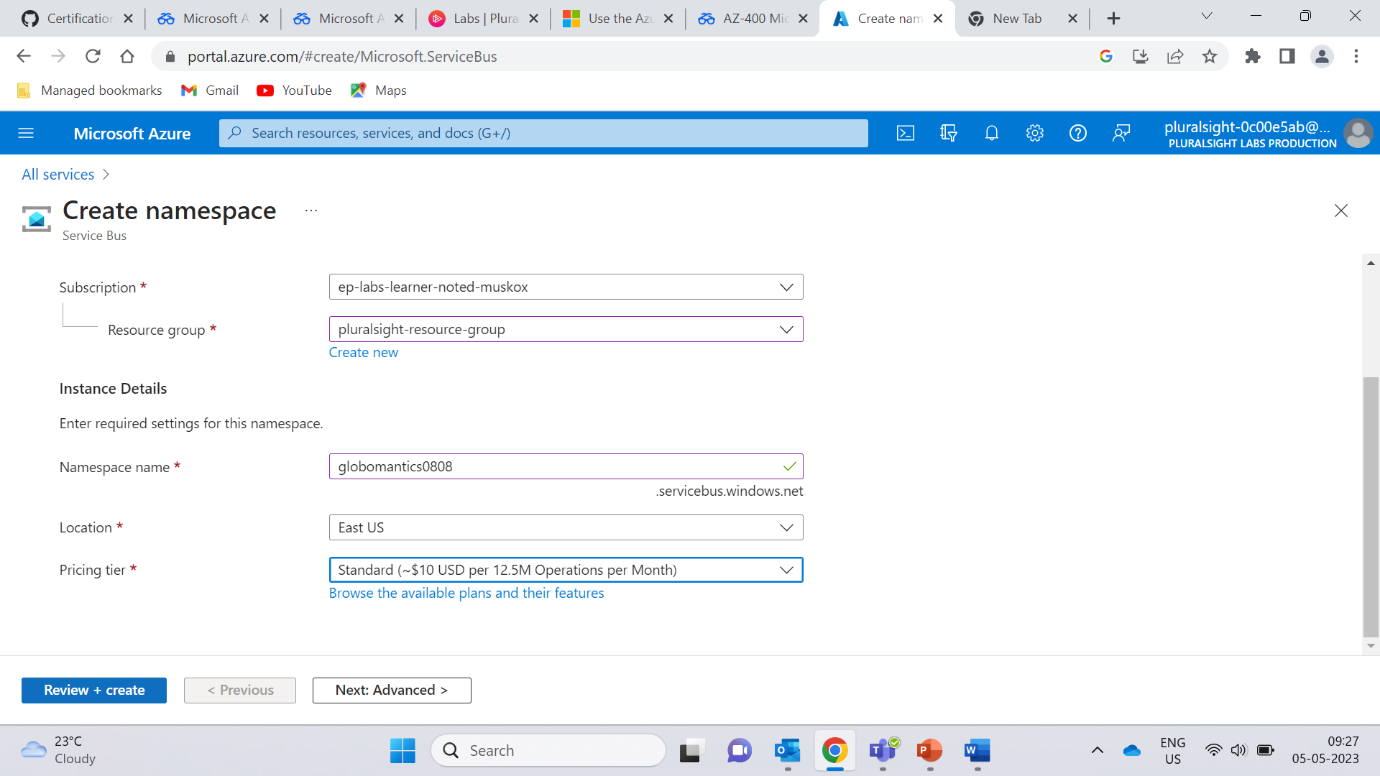
The Pluralsight provided **Subscription** should already be selected; however, you will need to select which **Resource group** the namespace should be a part of.

1. Click on the dropdown next to **Resource group** and select pluralsight-resource-group.
2. **Namespace name** needs to be unique across all of Azure. Ensure the **Location** is East US.
3. Click on the dropdown next to **Pricing tier** and select **Standard**.

Service Bus provides three different pricing tiers. You can see the specific differences by clicking the View full pricing details button.

* + Basic offers the lowest cost, but it only supports Queues.
  + Standard is recommended for non-production environments and supports Topics in addition to Queues.
  + Premium is recommended for production use and offers better performance, more predictable latency, zone redundancy, and larger message sizes; however, it's also more expensive.

1. Click the **Review + create** button.

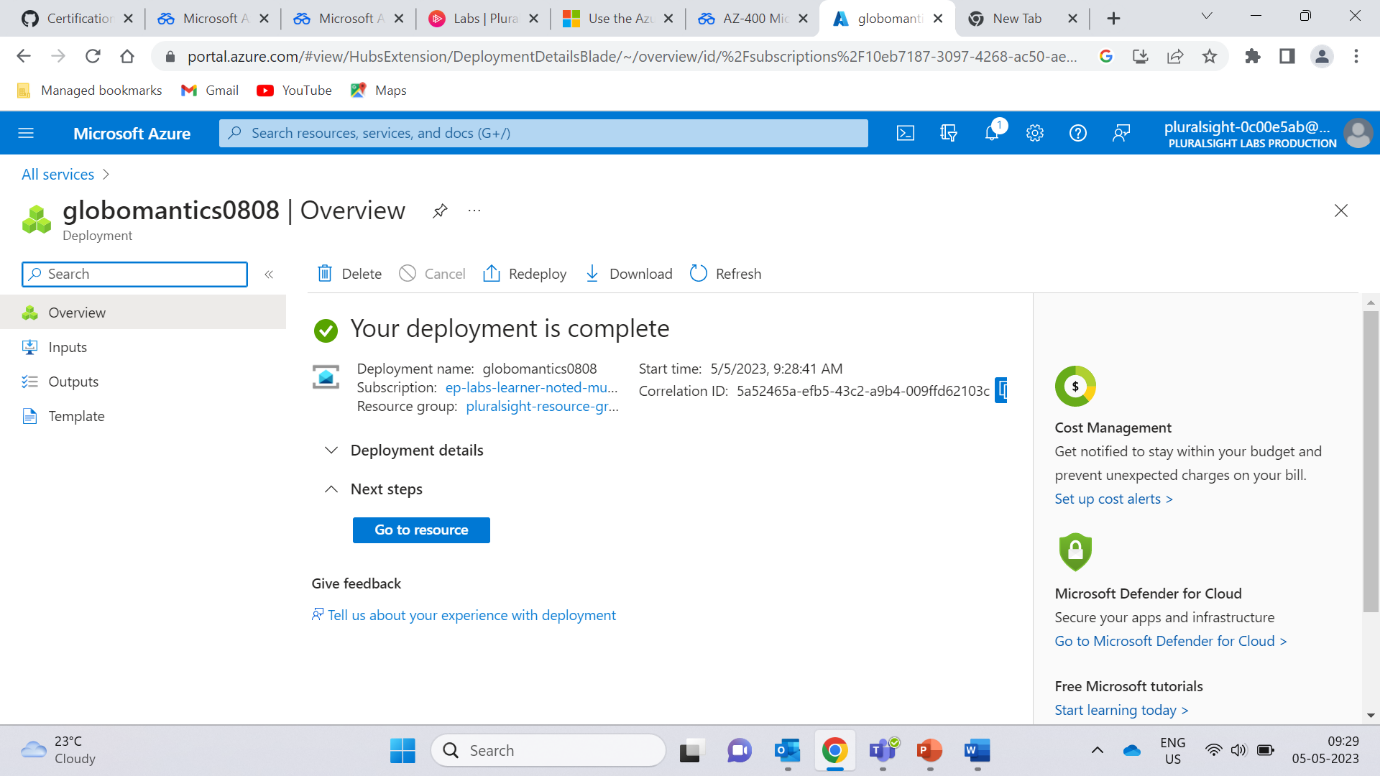


1. Click the blue **Create** button.

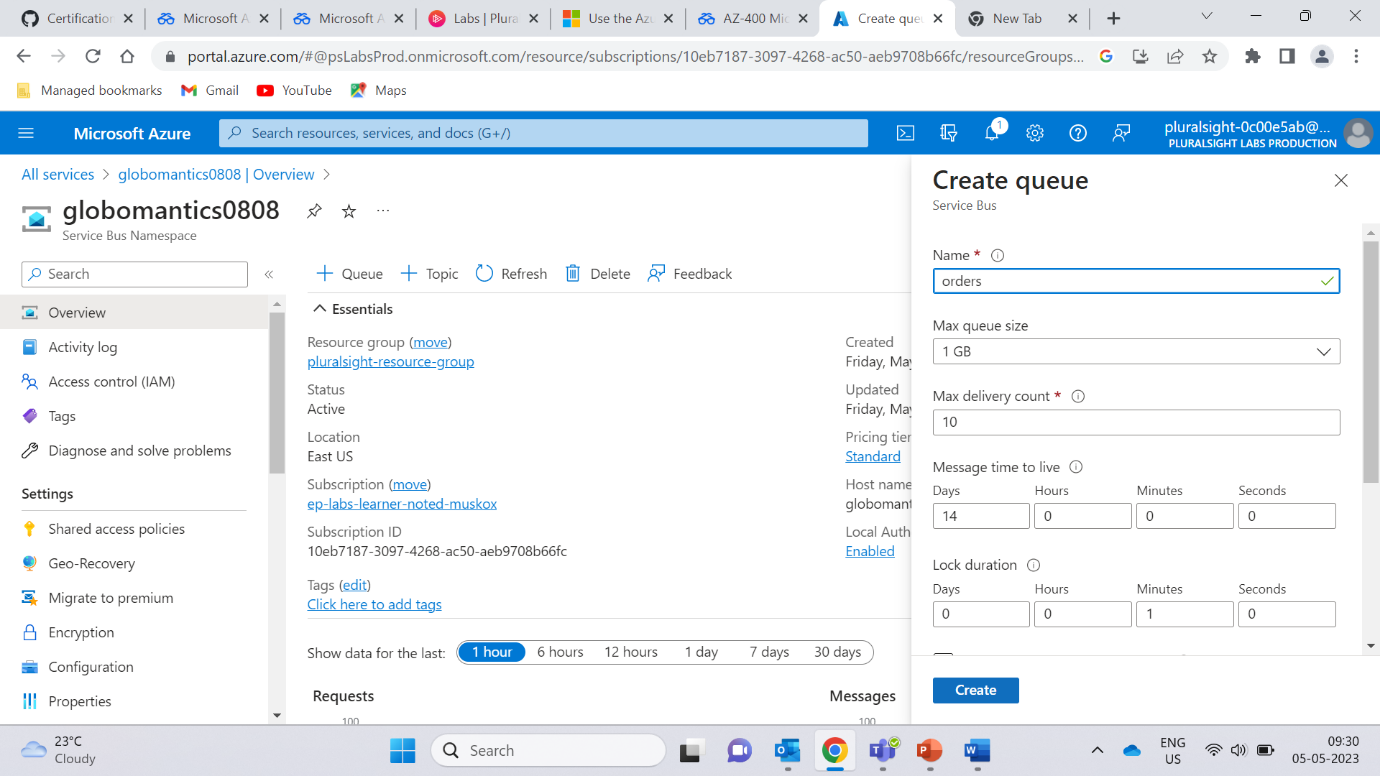


1. Click the blue **Go to resource** button.

Now that the namespace has been created, queues and topics can be created within it.

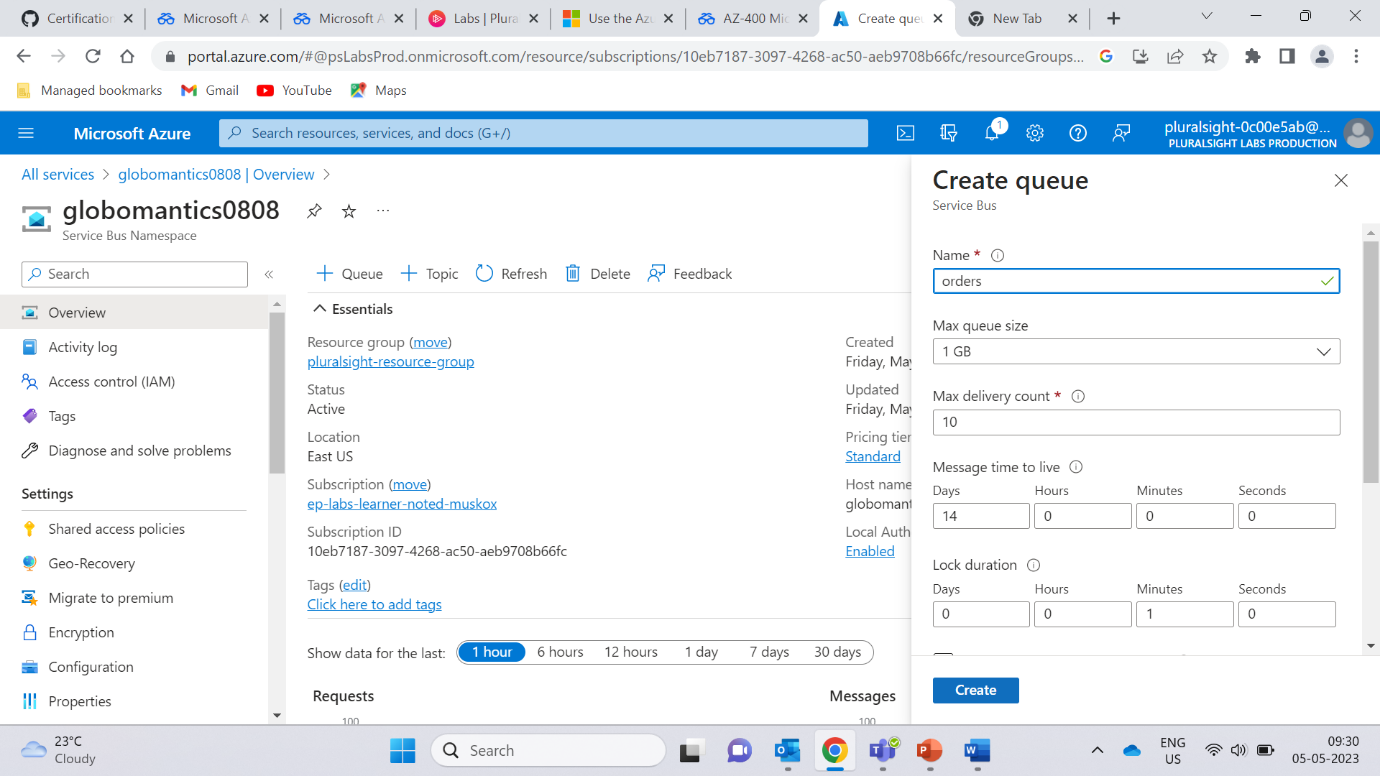


1. Click the **+ Queue** button.
2. Enter orders for **Name**.



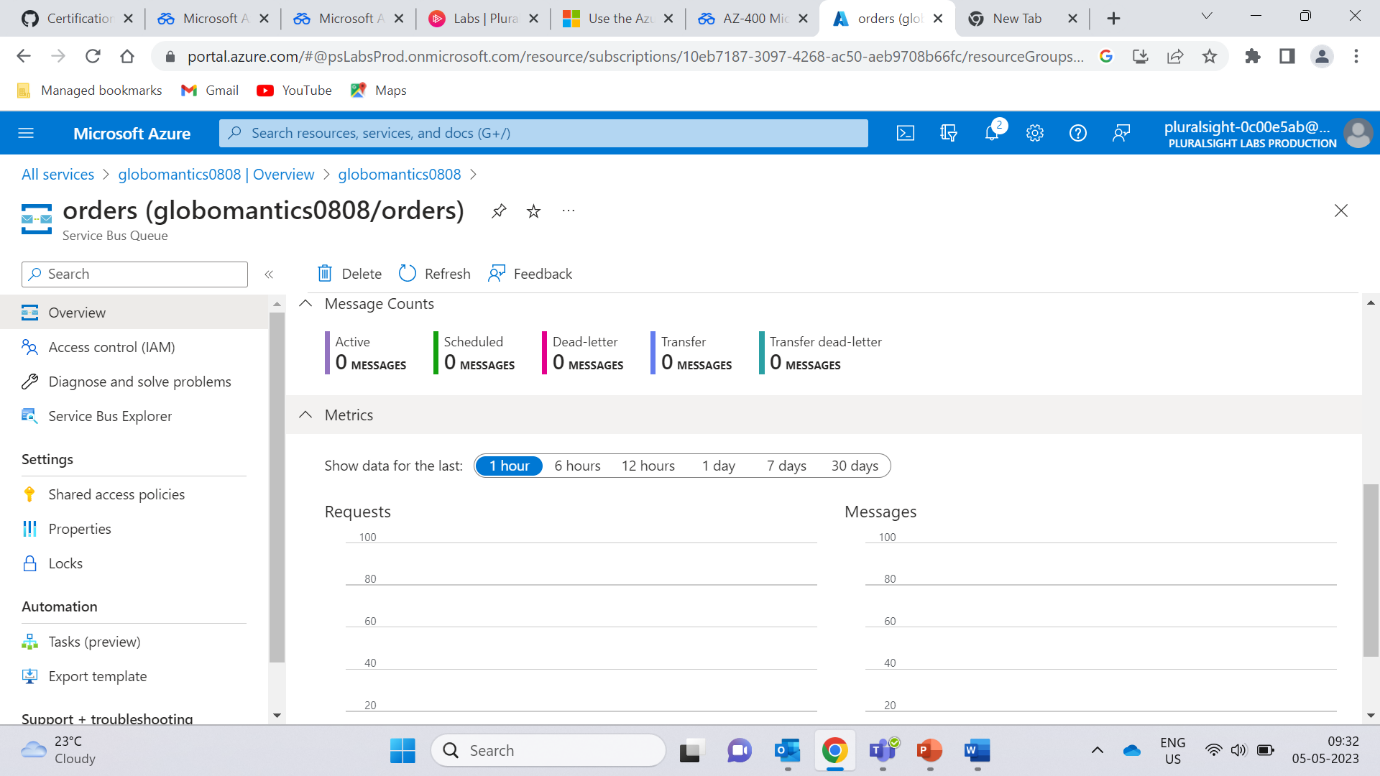
There are a number of properties that can be configured on a queue, but they can all be left on their defaults for now.

1. Click the blue **Create** button.

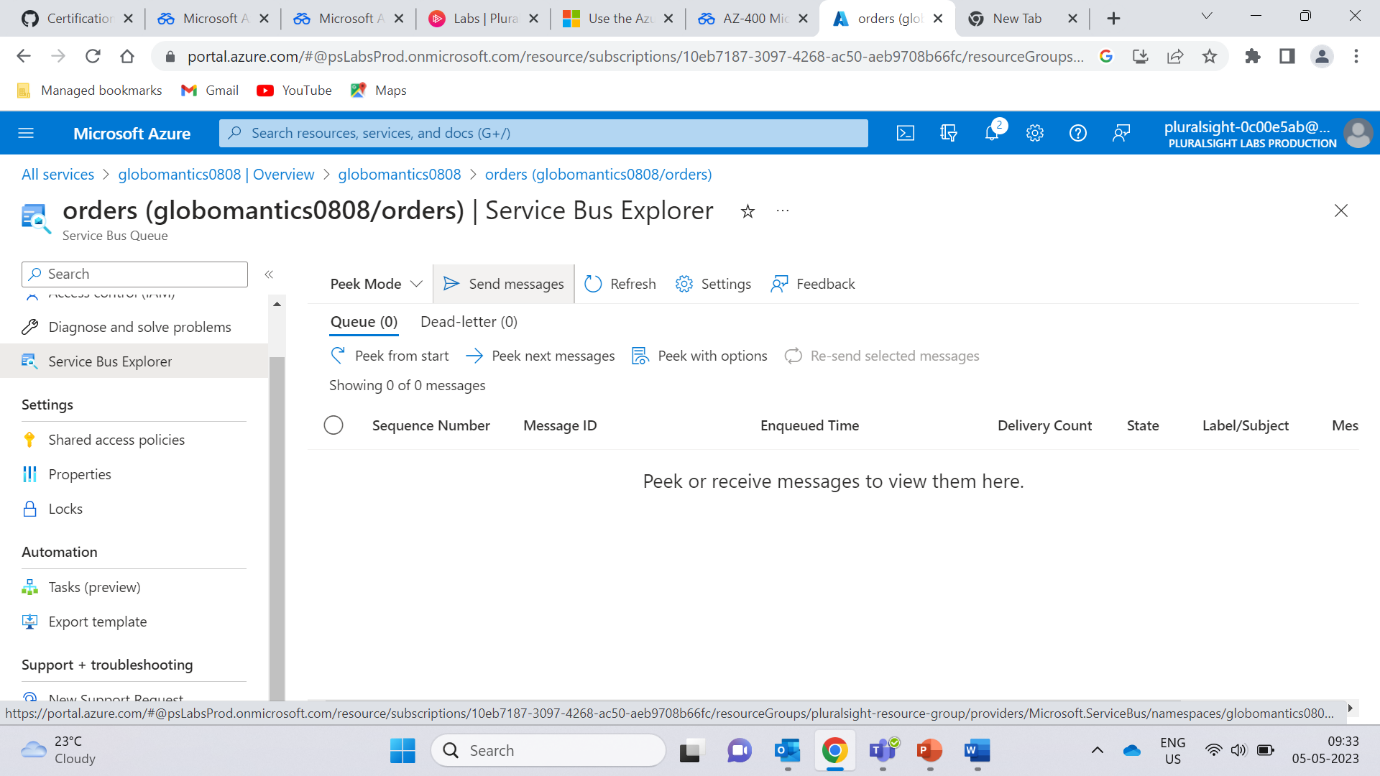


1. Click on the **orders** queue in the **Queues** list at the bottom of the screen.

There aren't currently any messages on this queue, so you'll see everything under **Message Counts** is 0 and the **Metric** won't show anything.



1. In the left-hand menu, click **Service Bus Explorer**.
2. Click on **Send messages**.



1. In the **Send messages** pane, copy and paste the following JSON into the **Message Body**:

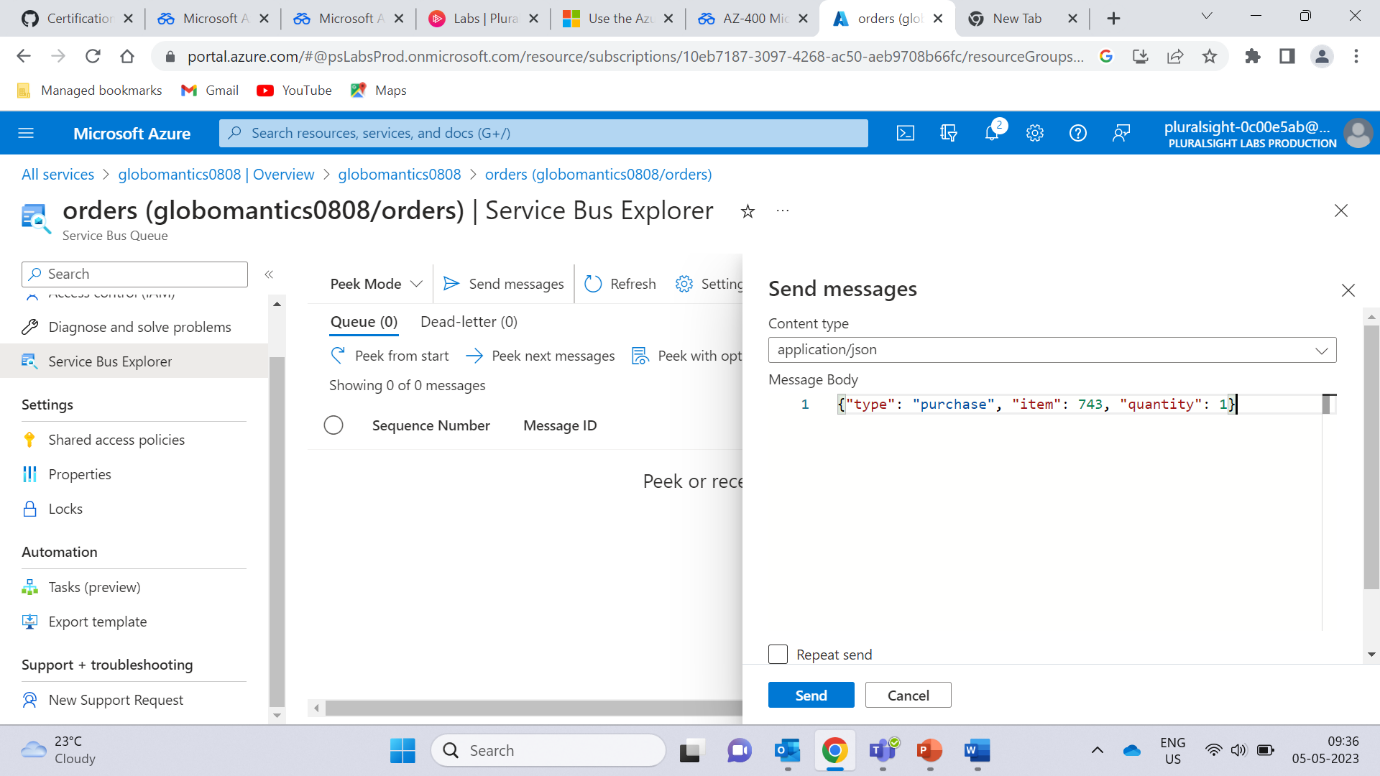
{"type": "purchase", "item": 743, "quantity": 1}

Before pasting, under **Message Body**, click to the right of the **1**. A cursor will appear, and you can then paste in the JSON.

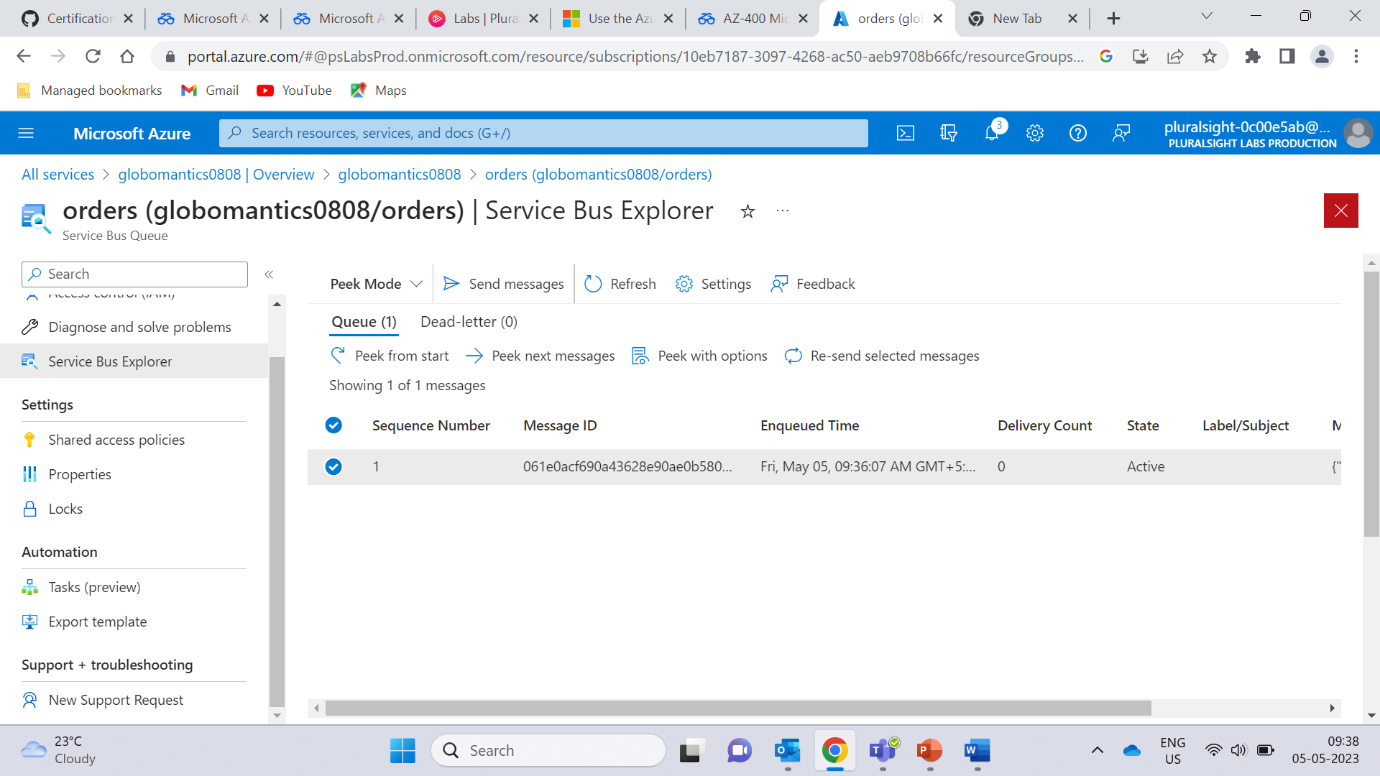
1. Still in the **Send messages** pane, under **Content type**, select **application/json**.

Message data on a queue or topic can be stored in any binary format. In this case, JSON will be used; however, other formats such as XML or plain text could be used as well.

1. Click the blue **Send** button.



1. Click **Peek from start**.
2. Select the single message from the list with a **Sequence Number** of **1**.



You should now see the content of the message you just sent. Since the Bus Explorer by default is in Peek Mode (as opposed to Receive Mode), this message is still on the queue waiting for a receiver to process it.

You've just created a Service Bus queue and sent a message to it. In the next challenge, you'll create an Azure Function to send more randomly generated messages to the queue.

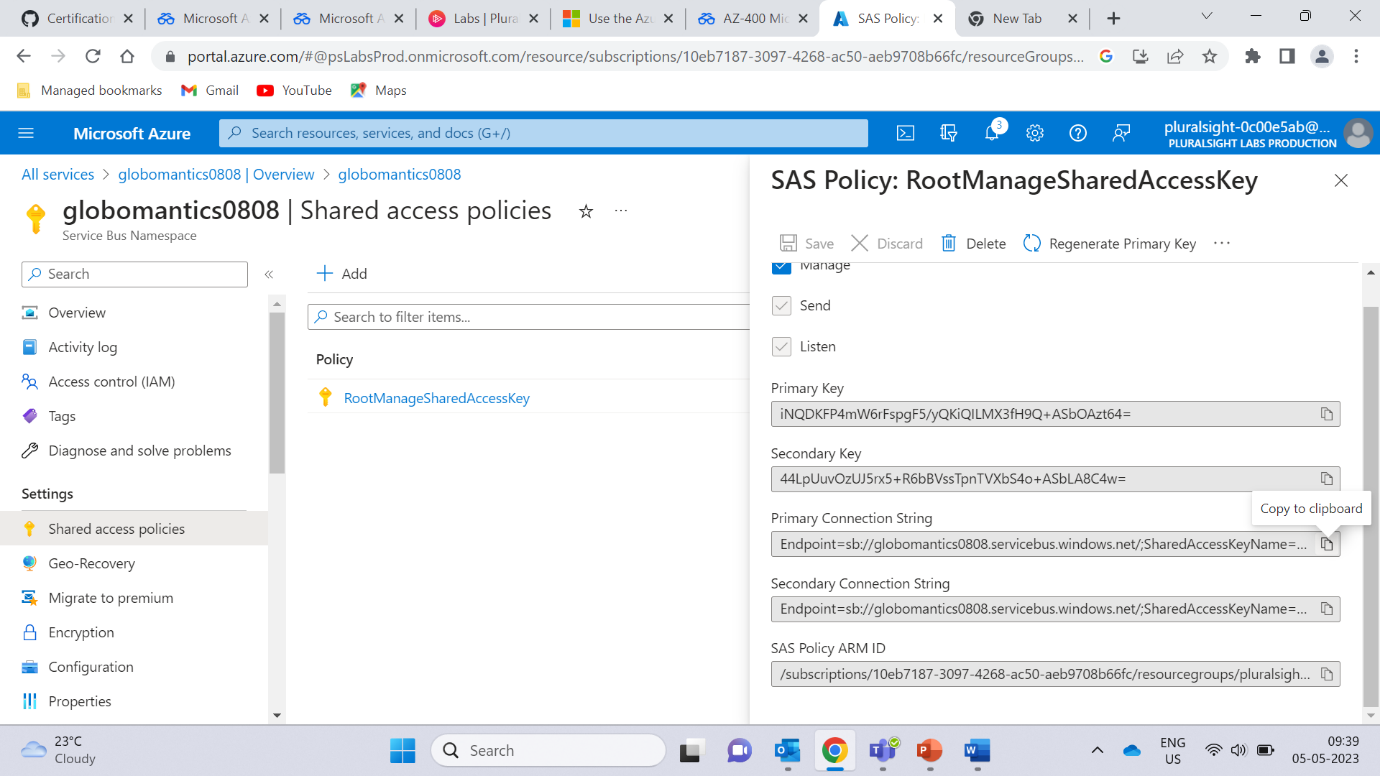
# Send Messages to the Queue

With the queue created, it's time to configure something to send messages to the queue and receive messages from the queue. In the future, messages will be sent by the order processing system; however, you'll be creating an Azure Function to generate random messages for testing purposes here.

1. Click on the **X** button in the upper-right corner to close the queue.
2. At the **globomantics-**prefixed **Service Bus Namespace** page, from the left sidebar under **Settings**, click **Shared access policies**.
3. Select the **RootManageSharedAccessKey** policy from the list.

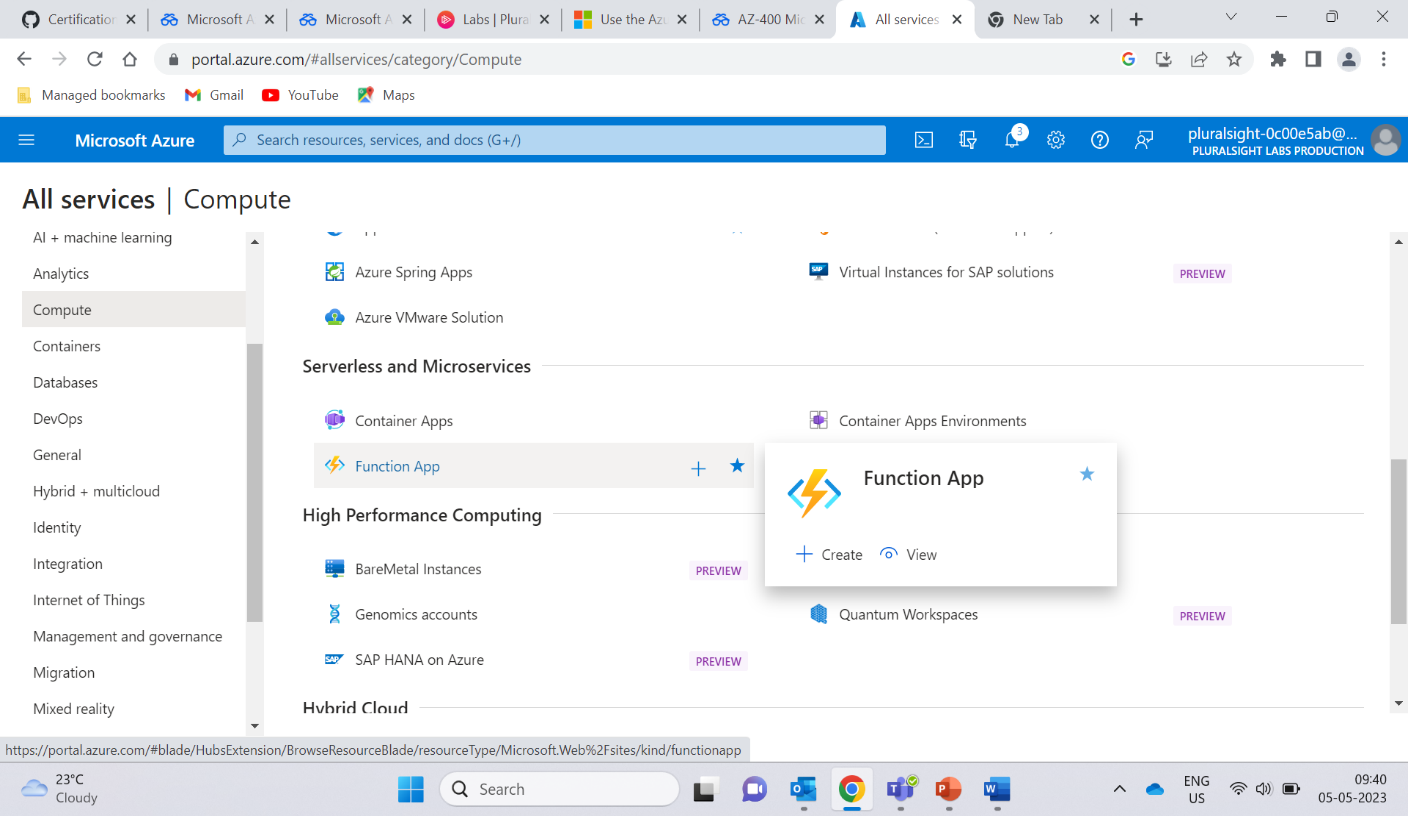
RootManageSharedAccessKey is a policy that's automatically created for all Service Bus namespaces with Manage permission. It's usually a good idea to create more restrictive policies with either Send or Receive permissions in case the connection string is leaked, though this will suffice for this example.

1. Copy the **Primary Connection String**, and store it locally for later use in this lab.



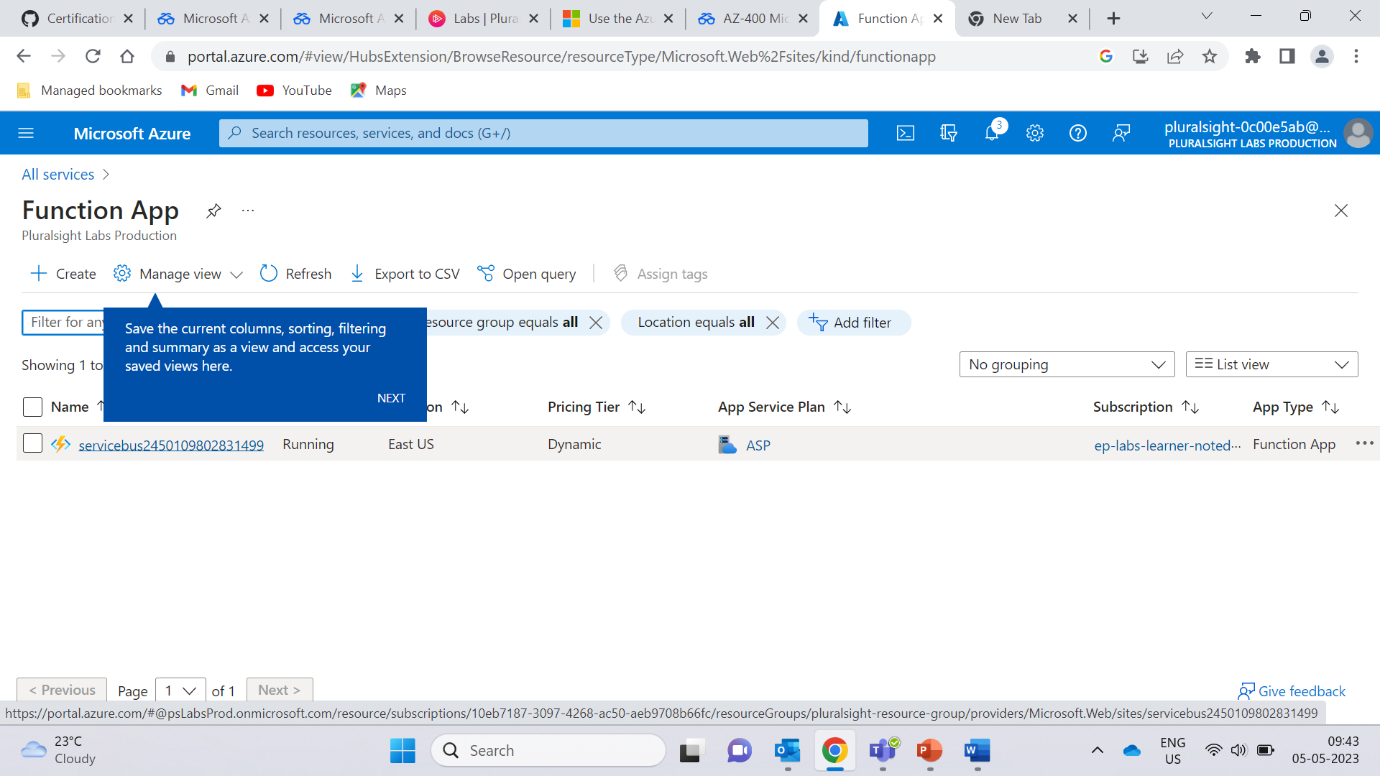
This namespace connection string will be used in the Azure Function by the Azure SDK to access the queue that was created in the last challenge.

1. Click on the menu button, ☰, in the upper-left corner and select **All services**.
2. Select **Compute** on the left sidebar under **Categories**.
3. Select **Function App** from the list.

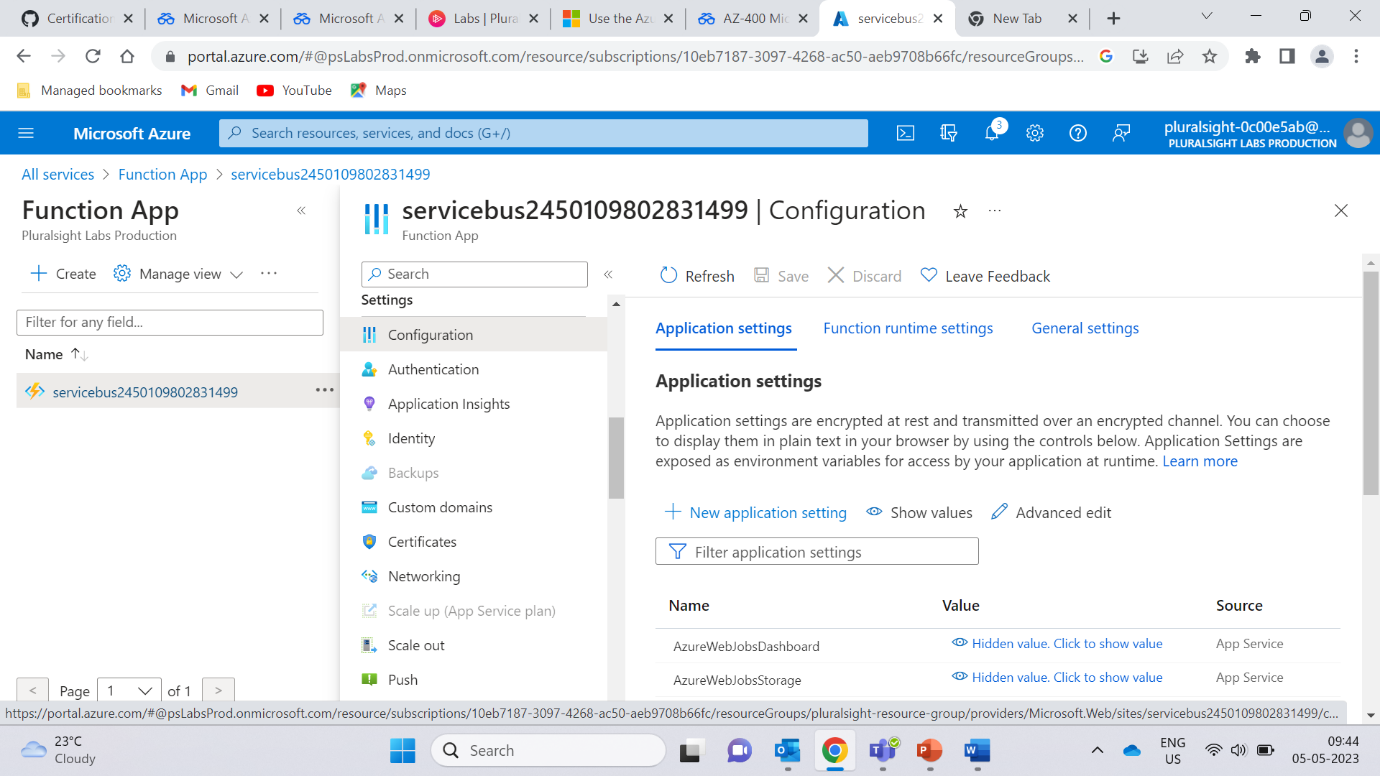


Azure Functions are used to run small amounts of code whenever specific triggers happen. In this case, two triggers will be used: messages will be sent to the queue on a timer trigger, and the queue will trigger another function to receive the messages.

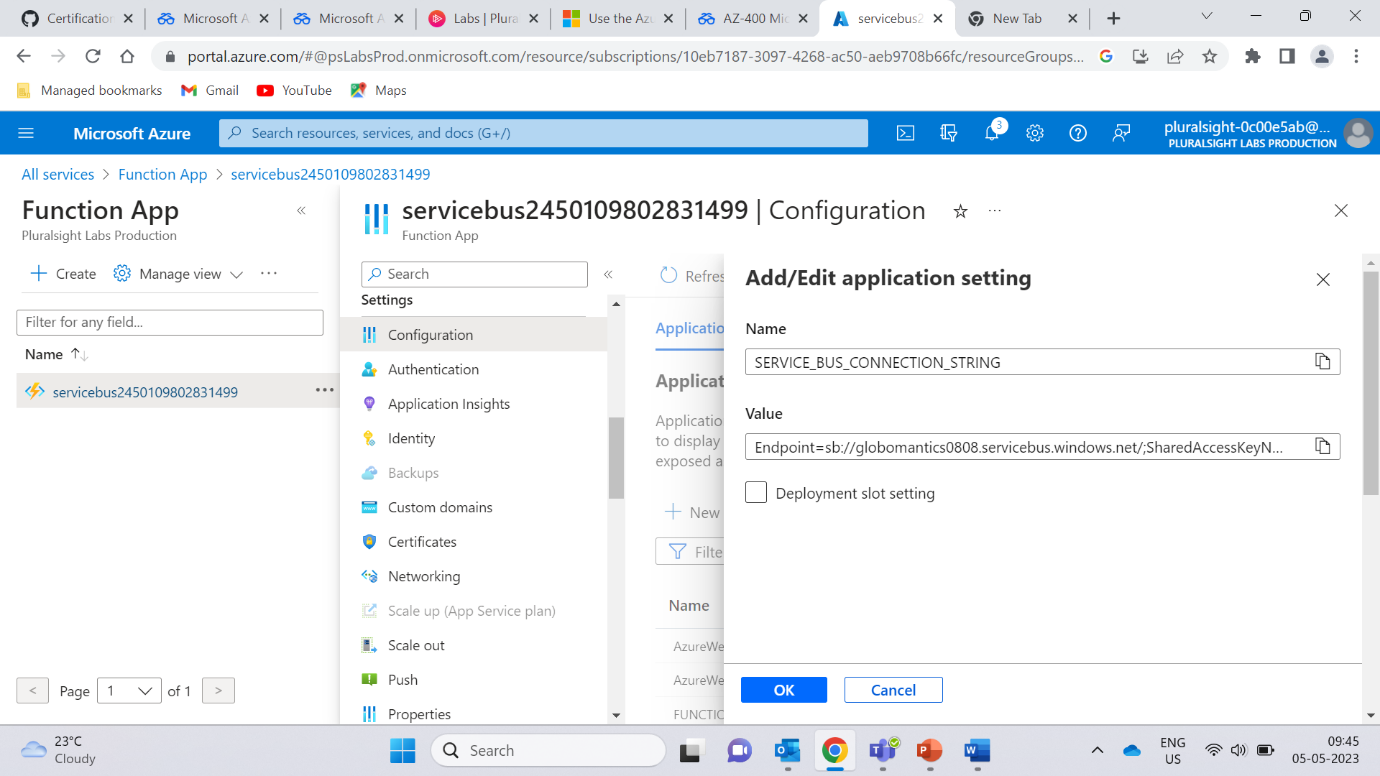
1. Click on the name of the **servicebus**-prefixed function app.



1. Click on **Configuration** under **Settings** on the left sidebar.
2. Click **+ New application setting.**

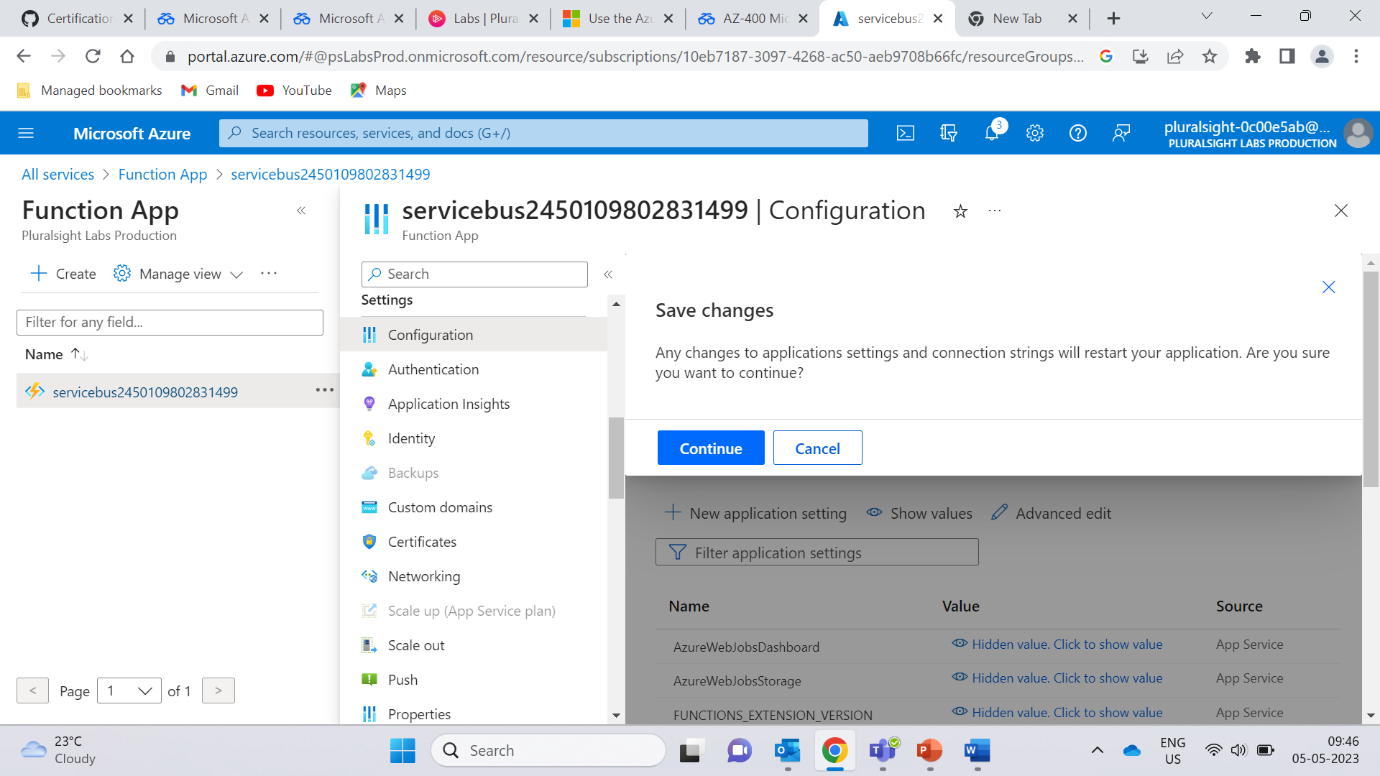


1. Enter SERVICE\_BUS\_CONNECTION\_STRING for **Name**.
2. Paste the Service Bus namespace connection string you copied earlier as the **Value**.



1. Click the blue **OK** button, then click the **Save** and **Continue** buttons.

This will cause the function to restart with the new setting defined.



1. Select **Functions** from the left sidebar.
2. Click the **+** **Create** button.
3. In the **Select a template** section, filter for and select **Timer trigger: C#**.
4. Enter RandomOrders for the name under **New Function**.
5. Replace the **Schedule** with 0/5 \* \* \* \* \*.



This will cause the function to be triggered once every 5 seconds.

1. Click the blue **Create** button.

Once the function has been created, you'll be redirected to the function's details page.

1. Click on the **Code + Test** button under **Developer** on the left sidebar.
2. Replace the contents of the run.csx file with the following code:

#r "Microsoft.ServiceBus"

#r "Newtonsoft.Json"

using System.Text;

using Microsoft.ServiceBus.Messaging;

using Newtonsoft.Json.Linq;

public async static Task Run(TimerInfo myTimer, TraceWriter log)

{

var random = new Random();

var client = QueueClient.CreateFromConnectionString(System.Environment.GetEnvironmentVariable("SERVICE\_BUS\_CONNECTION\_STRING"), "orders");

var json = JObject.FromObject(new {

type = "purchase",

item = random.Next(1000),

quantity = random.Next(20)

}).ToString();

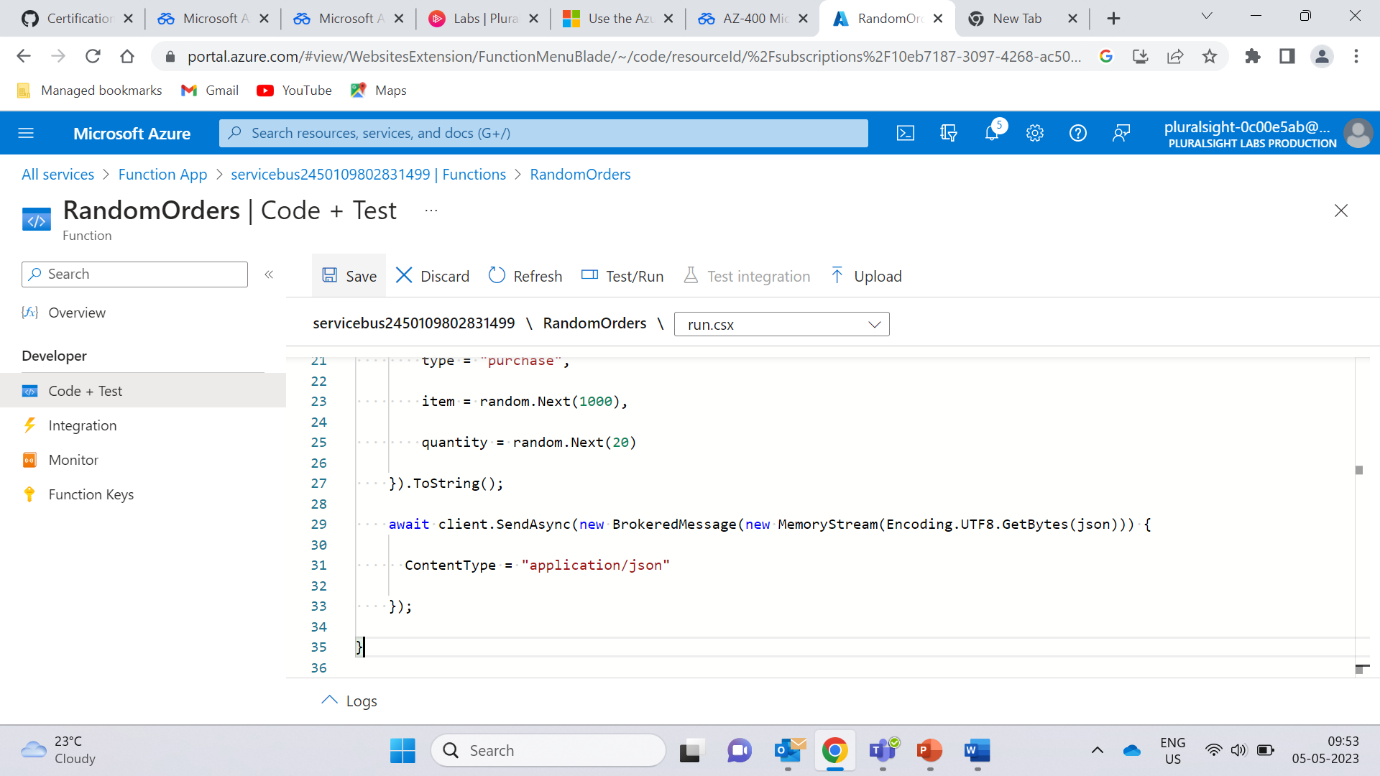
await client.SendAsync(new BrokeredMessage(new MemoryStream(Encoding.UTF8.GetBytes(json))) {

ContentType = "application/json"

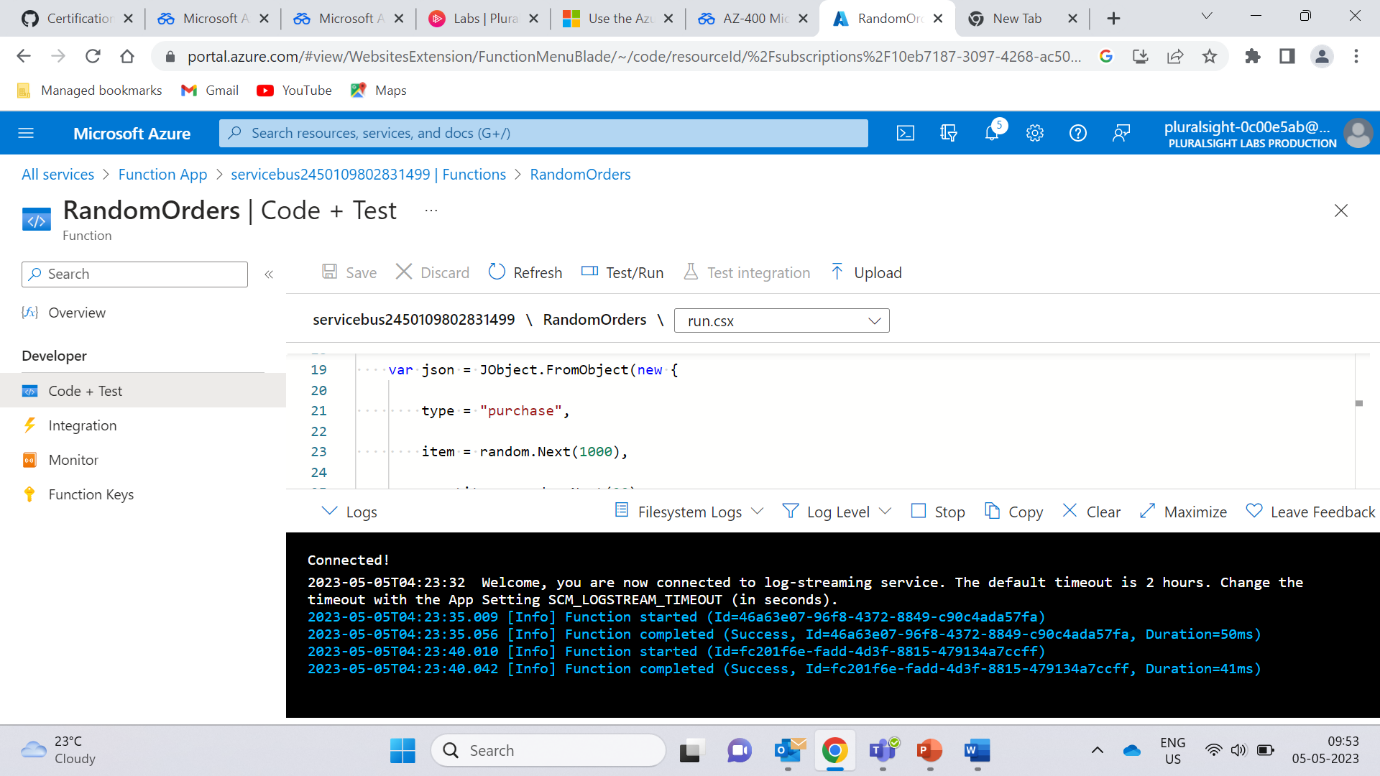
});

}

This code will run every 5 seconds. Each time it will generate a new JSON string with a random item and quantity. It will then use the QueueClient from Microsoft.ServiceBus.Messaging to send a message to the orders queue using the SERVICE\_BUS\_CONNECTION\_STRING environment variable as the connection string.



1. Click the **Save** button.
2. Expand the **Logs** tray at the bottom of the screen.



If you scroll to the bottom of the logs, you should see that the function is being run every 5 seconds. Each time it runs, it generates a random message and sends it to the queue. You may need to wait for a little bit for the logs to show. You can click the **Refresh** button, if needed, to get them to show after waiting for a bit.

# Receive Messages from the Queue

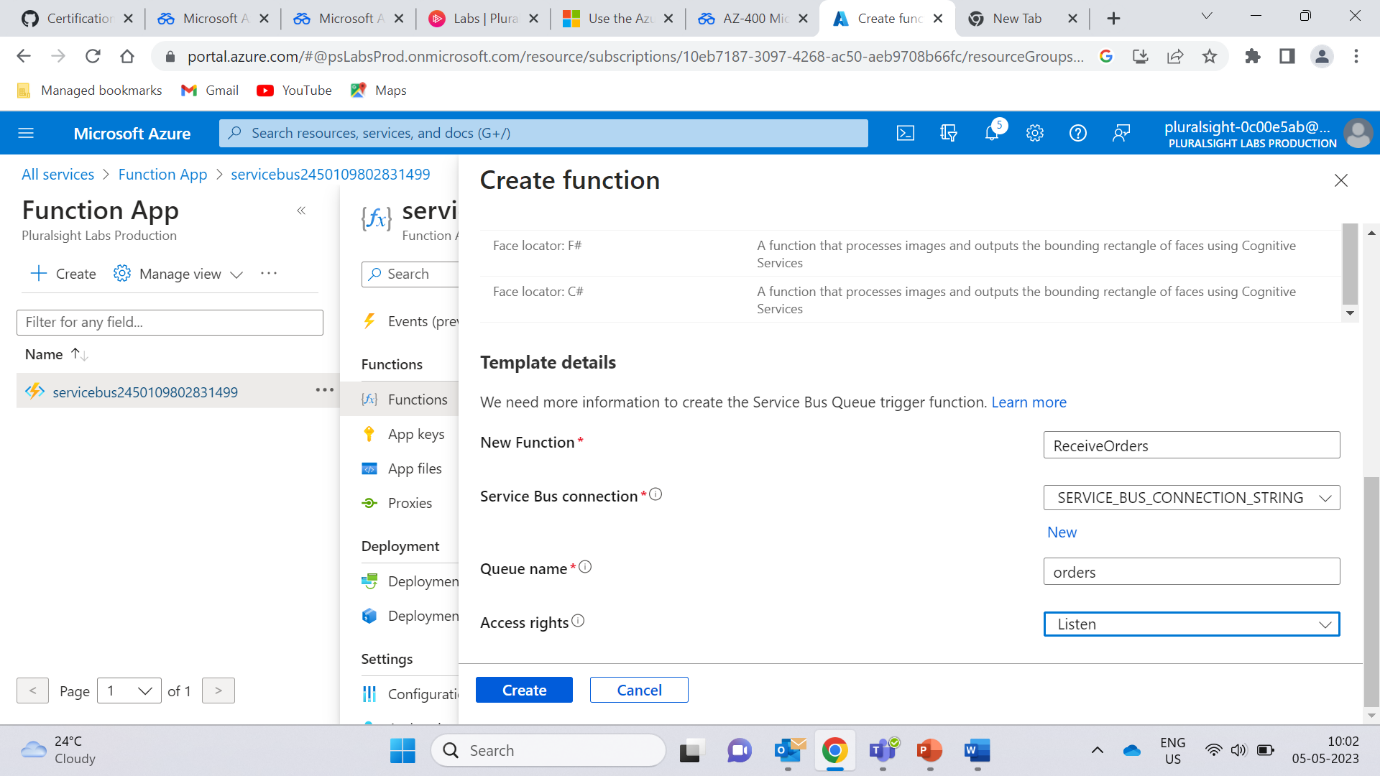
Now that random messages are being sent to the queue, you can configure a receiver for the queue.

1. Click on the **X** button in the upper-right corner to close the function.
2. Click the **+** **Create** button to create another function.
3. Search for and select **Service Bus Queue trigger: C#**.
4. Enter ReceiveOrders as the name under **New Function**.

The **Service Bus connection** setting should have automatically found your SERVICE\_BUS\_CONNECTION\_STRING variable.

1. Enter orders for the **Queue name**.
2. Set **Access rights** to **Listen**.
3. Click the blue **Create** button.

Once the function has been created, you'll be redirected to the function's details page.



1. Click the **Code + Test** button under **Developer** on the left sidebar.
2. Replace the contents of the run.csx file with the following code.

#r "Newtonsoft.Json"

using System;

using Newtonsoft.Json;

using Newtonsoft.Json.Linq;

public static void Run(string myQueueItem, TraceWriter log)

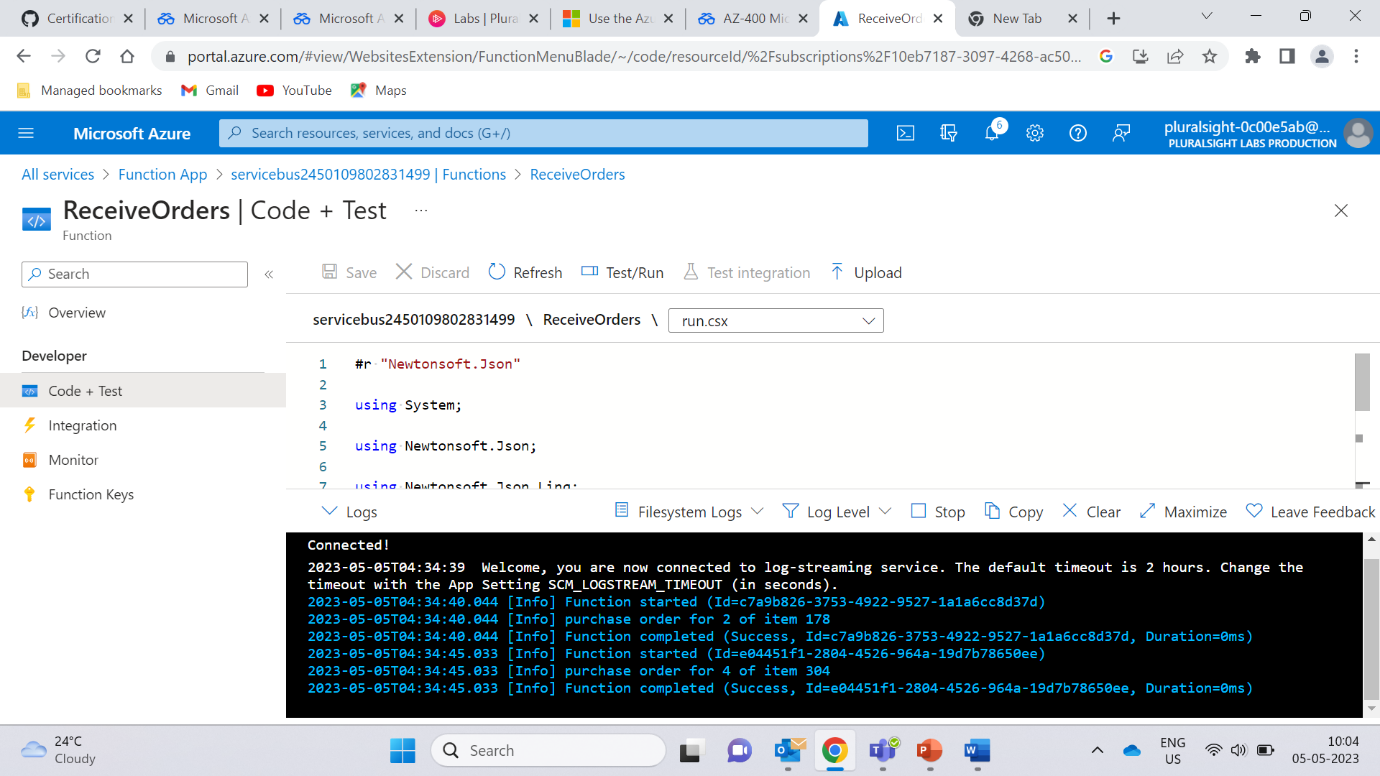
{

var message = JObject.Parse(myQueueItem);

log.Info($"{message["type"]} order for {message["quantity"]} of item {message["item"]}");

}

1. Click the **Save** button.
2. Expand the **Logs** tray at the bottom of the screen.



Scrolling to the bottom of the logs, you should see that this function is being triggered every 5 seconds by the random messages which are being sent to the queue by the other function.

You should now have two Azure Functions which are using an Azure Service Bus queue to communicate.

