Microsoft

Azure Integration Services (AIS)

Lab 1
Developing Enterprise Serverless Messaging Integration

DISCLSIMER

The information herein is for informational purposes only and represents the current view of Microsoft Corporation or any Microsoft Group affiliate as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

CONFIDENTIALITY

Contents

Lab 1 – Developing Enterprise Serverless Messaging Integration 5
Exercise 1- Set up Azure environment (5m) 6
Module 1 – Create a new APIM Instance 6
Exercise 2- Provision Required Infrastructure (40 min) 8
Module 1 – Create CosmosDB for data storage 9
Module 2 – Create Service Bus 10
Module 3 – Create Service Bus Topic and Subscription 11
Module 4 – Create Logic App Facade 12
Module 5 – Create Logic App for Business Integration Logic 13
Module 6 – Create Fisheries database in CosmosDB 13
Module 7 – Verify your infrastructure deployment 14
Exercise 3 – Configure Accesses (40min) 15
Module 1 – Understand required accesses 15
Module 2 – Configure Access from Logic App Façade to Service Bus 15
Module 3 – Configure Access from Logic App for Business to Service Bus 17
Module 4 – Configure Access from Logic App for Business to CosmosDB 17
Summary 19
Exercise 4 – Implement Integration Logic for Logic App Façade (50min) 20
Module 1 – Add HTTP Request Trigger 21
Module 2 – Add An Action for Extracting Business Data 23
Module 3 – Add Conditional Check 25
Module 4 – Add a 'Case' Action for Switch Statement 26
Module 5 – Send Message to Service Bus 27
Module 6 – Validate Logic App logic 30
Summary 31
Exercise 5 – Implement Integration Logic for Logic App For Business (60min) 32
Module 1 – Add a Trigger for Service Bus new message 32
Module 2 – Add an Action to Parse Message Content (JSON) 33
Module 3 – Validate Service Bus Message and Logic App Trigger 34
Module 4 – Add an Action for Parsing Message Properties 35
Module 5 – Add an Action for Composing Required Data for Save 36
Module 6 – Save The Vessel Data to CosmosDB 37
Module 7 – Validate Messaging Integration 38
Summary 39
Exercise 6 – Configure External Endpoint (15min + Optional) 40
Module 1 – Publish Logic App Façade endpoint in APIM 40
Module 2 – Validate End-to-end integration flow 42
Module 3 (Optional) – Enable and Use Developer Portal 43

Module 4 (Optional) – Configure API Policy(s) 45
Summary 46
Exercise 7 – (Optional) Implement Additional Requirement for a Full Audit (40 min) 47
Module 1 – Create New Audit Database in CosmosDB and Configure Access 47
Module 2 – Validate Audit Functionality 49
Summary 49

Lab 1 – Developing Enterprise Serverless Messaging Integration

Objective(s)	• To be able to provision required infrastructure resources in Azure • Configure required accesses • Implement integration logic • Persist the data for business use • Expand additional integration requirements by adding new components without impacting the existing functionality
Duration of Lab	• 4h
Prerequisite(s)	• Azure Subscription and Resource Group • **Contributor** Role in a selected Resource Group • Access to the Internet
Tool(s)	• Azure Portal
Exercises	1. Set up Azure environment (5m) 2. Provision Required Integration Infrastructure (40m) 3. Configure Accesses (40min) 4. Implement Integration Logic for Logic App Façade (50min) 5. Implement Integration Logic for Logic App for Business (60min) 6. Configure External Endpoint (15min + Optional) 7. Optional: Implement Additional Requirement for a Full Audit (40min)
Subscription	[Selected Subscription]
Resource Group	[Selected RG]
Navigation	Throughout this Lab, we will open and use several Browser tabs for easy access. Until the end of the Lab, keep your Browser tabs open.

Naming Convention for Labs

For completing various labs during the workshop, we will use this naming convention. It is slightly different from Microsoft online guidance (Define your naming convention - Cloud Adoption Framework | Microsoft Learn).

The naming convention below is designed to group your Azure resources together for easy access.

[you name/initials]-[short name for Azure service]-[service description]

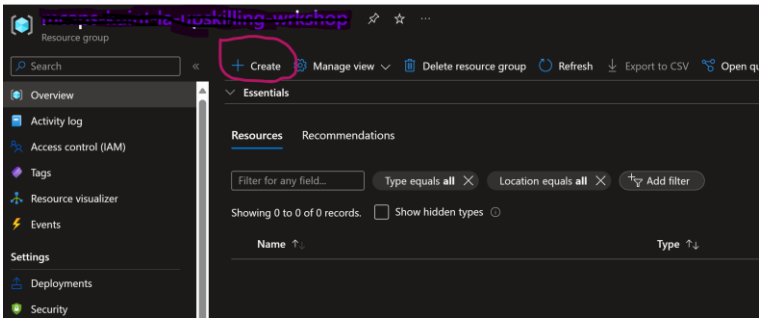
Exercise 1- Set up Azure environment (5m)

Topics	In this exercise, we will cover the following topics. • Create APIM
Duration	• 5 min
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]

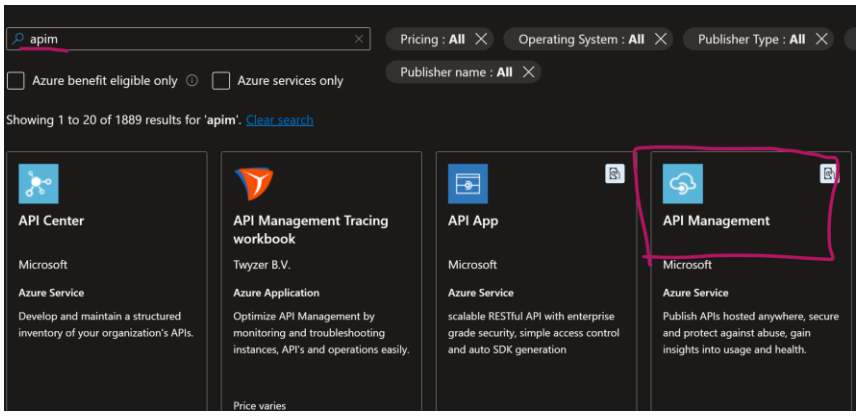
Module 1 – Create a new APIM Instance

1. In a browser, go to *Azure Portal* (https://portal.azure.com)
2. Select Resource groups menu

3. Within the *Overview* menu, select the **Create** button

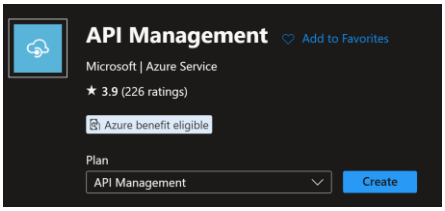


4. Type **APIM** in the search box and select **API Management resource**



5. Note: You can create APIM from various areas in Azure portal. Doing this way, pre-fills the Subscription and Resource Group fields for you automatically.

6. Click the **Create** button

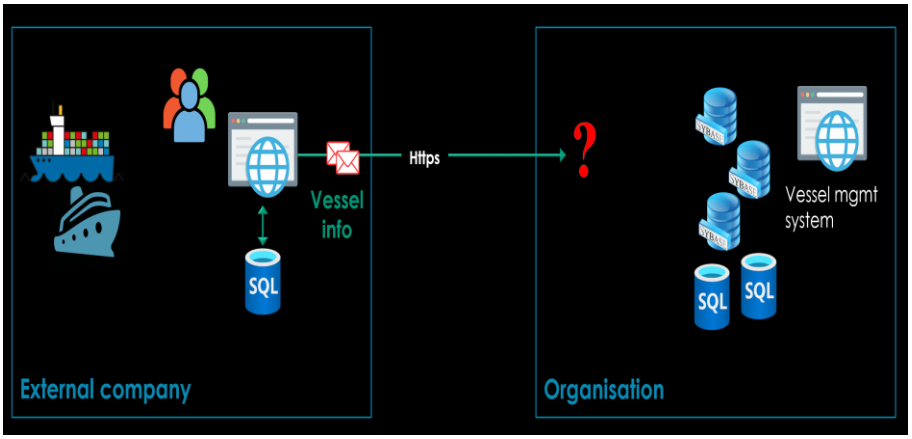


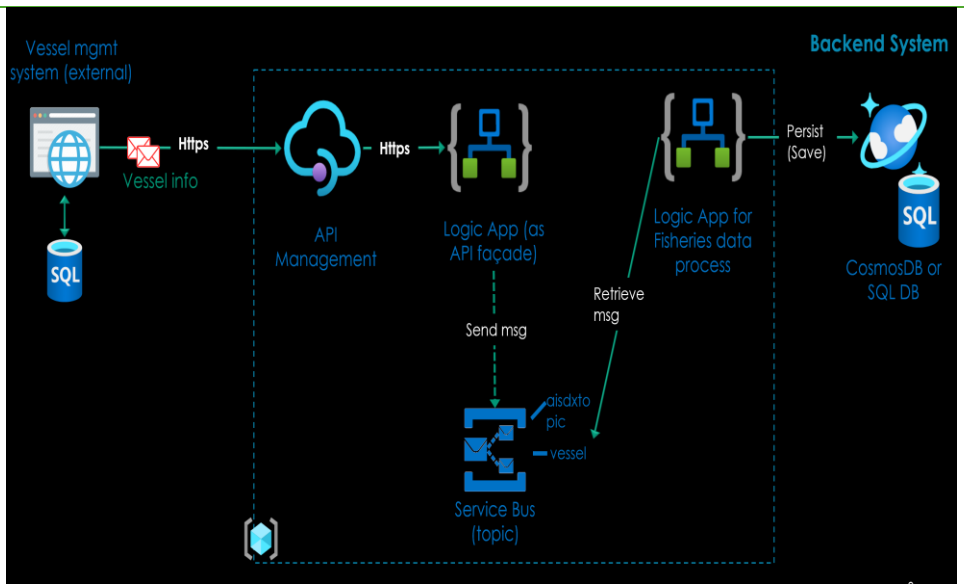
7. Enter the following details

Area	Value
Subscription	[Selected subscription]
Resource Group	[Selected RG]
Region	**Australia East** (or pre-selected based on selected Resource Group)
Resource name	**[your name/initials]-apim-ais-labs**
Organisation name	[any value is fine for the lab. Or enter your organisation's name]
Administrator email	[your email address] Note: This email is used to contact an administrator in the event of APIM service issue (e.g. unavailable)
Pricing tier	**Developer (no SLA)**

8. Click the **Next: Monitor + secure>** button
9. Click the **Next: Virtual network>** button
   Connectivity type = **None**
10. Click **the Next: Managed identity >** button
11. Click the **Next: Review + install >** button
12. At the Review + install tab, click the **Create** button at the bottom
13. It takes a few minutes for a new APIM instance to be created
14. Once the resource is created, go to the resource

Exercise 2- Provision Required Infrastructure (40 min)

Prerequisite(s)	• APIM Instance
Topics	In this exercise, we will cover the following topics. • Creating all the required resources for messaging integration in Azure portal
Duration	• 40 min
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]
API Management	**[your name/initials]-apim-ais-labs**
Scenario	The Fisheries group in the ministry collects and records the vessel information for all vessels that are coming into the country and depart the country. Vessel information is managed by an external party who uses their own system to capture and record all the vessel information. The external party is responsible for sending all the vessel information to the ministry regularly and correctly. The external party decides to adopt modern integration mechanism rather than relying on traditional batch sync or data dump (e.g. butch database job). The external party needs to send a *message-based* data to the ministry as soon as a new vessel information is entered in their internal system. The ministry needs to store a newly arrived vessel information in their Vessel database 
Architecture	Using Azure Integration Services (AIS) capabilities, the solution architecture looks like this:

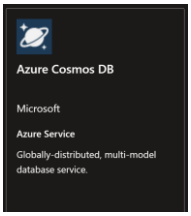
<table>
<tr>
<td></td>
<td>


- **API Management:** Exposes an integration endpoint to the external party
- **Logic Apps (as façade):** Accepts HTTP calls to receive vessel information (messages) from the external party and acts as façade (pass-through). It sends messages to ServiceBus topic/queue
- **ServiceBus:** Implements 1-to-many (pub/sub) messaging for integration for this Lab
- **Logic Apps (for Fisheries):**
</td>
</tr>
<tr>
<td>Out of scope</td>
<td>

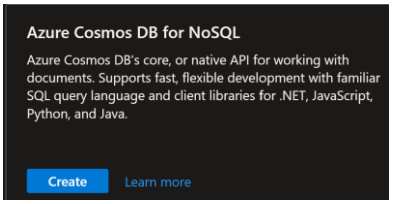
- For this lab, Virtual networking design and considerations are not included as well as Azure networking architecture (e.g. App GateWay, Network routing etc…)
</td>
</tr>
</table>

Module 1 – Create CosmosDB for data storage

1. You will create a new CosmosDB for persisting non-structured data (e.g. json) for integration
2. In a browser in Azure portal, type ***"CosmosDB"*** in the search textbox, whether it's within your selected Resource Group screen or the top search textbox in the Portal
3. Select **Azure Cosmos DB** and click the **Create** button



4. You will use Azure Cosmos DB for NoSQL API for the workshop. Click the **Create** button



5. Enter the following details

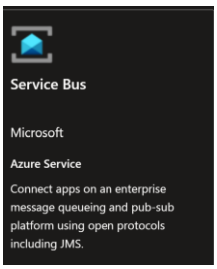
Area	Value
Workload Type	**Development /Testing**
Subscription	[Selected subscription]

Area	Value
Resource Group	[Selected RG]
Region	**Australia East** (or pre-selected based on selected Resource Group)
Resource name	**[your name/initials]-apim-ais-labs**
Account Name	**[your shortname]-cosmos-ais-labs**
Availability Zones	**Disable**
Location	**Australia East** Note: if Australia East is not available, select **Australia Southeast**
Capacity mode	**Serverless**
Global distribution Tab	
Geo-Redundancy	**Disable**
Multi-region Writes	**Disable**
Networking Tab	
	(no change)
Backup Policy Tab	
	(no change)
Backup storage redundancy	**Locally-redundant backup storage**
Security	
	(no change)
Tags Tab	
	(no change)

6. Review the details and click the **Create** button

Module 2 – Create Service Bus

1. You will create a Service Bus for asynchronous messaging integration
2. Enter **"Service Bus"** in the search text box, whether it's within your selected Resource Group or in the search textbox at the top
3. Select Service Bus and click the Create button



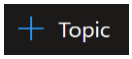
4. Enter the following

Area	Value
Subscription	[Selected subscription]
Resource Group	[Selected RG]

Area	Value
Namespace name	**[your short name]-sb-ais-labs**
Location	**Australia East**
Pricing Tier	**Standard**
Enable Geo-replication	(untick)

5. Click **Next** buttons for the subsequent tabs and progress to the *Review + create* Tab
6. Click the **Create** button

Module 3 – Create Service Bus Topic and Subscription

1. Service Bus supports many-to-many (pub/sub) messaging to broadcast messages received to the consumers/end users
2. Once Service Bus resource is provisioned, go to the resource
3. Select the **Topics** menu under the *Entities* section on the left menu
4. Select **+ Tocpic** button 

to create a new messaging topic
5. Enter the following details

Area	Value
Name	**Vessels**
Max topic size	1 GB

6. Click the **Create** button
7. A new topic, Vessels, is created

Name	Status	Scheduled messages	Max size	Subscription count	Enable partitioning
vessels	Active	0	1024 MB	0	false

8. This allows anyone who's interested in the "Vessels" can subscribe to this topic
9. You need to create a topic subscription to be able to receive messages for the vessel information

A *Topic* in Service Bus is like a broadcast channel. When a message is sent to a topic:

- It is **not consumed directly**
- Instead, it is **delivered to each subscription** under that topic
- Each subscription acts like an **independent queue**

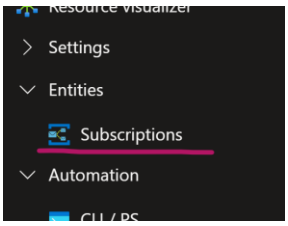
Each system or service needs to process the same message without interfering with others.

e.g.

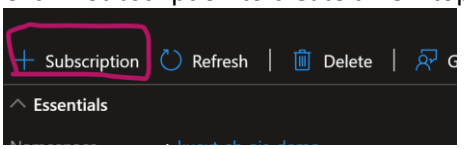
- Subscription A: sends notifications to port authorities
- Subscription B: updates the logistics dashboard
- Subscription C: triggers customs clearance workflows

Each of these services gets its own copy of the message.

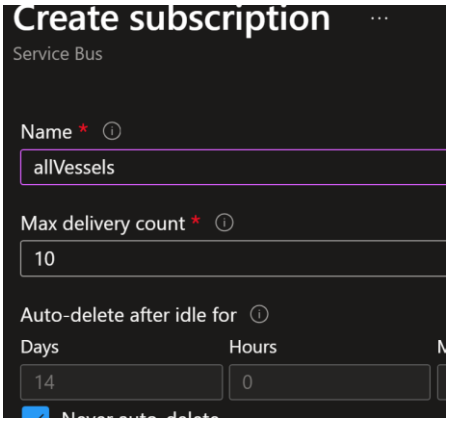
10. Select the newly created topic, *vessels*
11. Select the **Subscriptions** menu under the *Entities* section on the left



12. Click + Subscription to create a new topic subscription



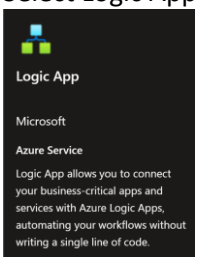
13. Enter ***"allVessels"*** for the *Name* field and leave the rest as they are



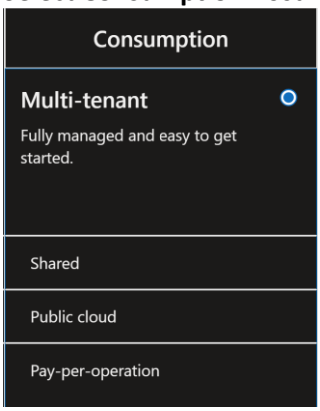
14. Click the **Create** button
15.

Module 4 – Create Logic App Facade

1. You will create a Logic Apps for receiving messages as façade
2. Type ***"Logic App"*** in the search textbox, whether it's in your selected Resource Group or at the top of the portal
3. Select Logic App and click the Create button



4. Select **Consumption** hosting plan and click the **Select** button



5. Enter the following details

Area	Value
Subscription	[Selected subscription]
Resource Group	[Selected RG]
Logic App name	**[your short name]-la-ais-labs-facade**
Region	**Australia East**
Enable log analytics	**No**

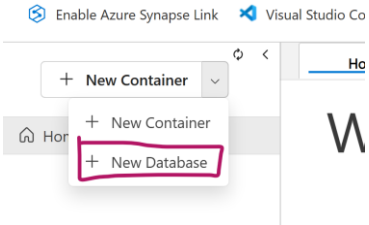
6. Click the **Review + create** button and click the **Create** button
7.

Module 5 – Create Logic App for Business Integration Logic

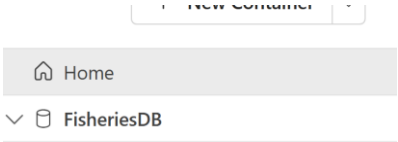
1. Create another Logic App. This time, Logic App will monitor Service Bus activities to receive messages and implement the business requirements to handle received vessel details for their use
2. Create another Logic App called **[your short name]-la-ais-labs-fisheries**

Module 6 – Create Fisheries database in CosmosDB

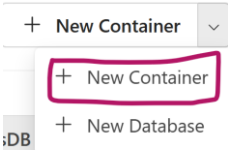
1. By now, your Cosmos DB has been provisioned and is ready for use
2. Select your *Cosmos DB resource*
3. Select the **Data Explorer** menu on the left menus
4. Click the dropdown and select + New Database



5. Enter the *Database id* as **FisheriesDB** and click the **OK** button
6. Your new database called FisheriesDB is created



7. At the moment, you have no table(s) in the database. You will create one for storing vessel information
8. Select the database in the *Data Explorer*
9. Select the *dropdown list* at the top and select **+ New Container**



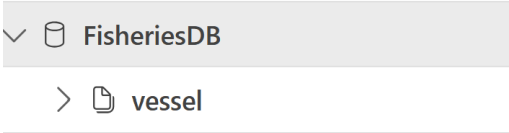
Note: In CosmosDB, a *database table* is referred as a ***Container***

10. Enter the following details

Area	Value
Database di	Use existing – (select **FisheriesDB**)

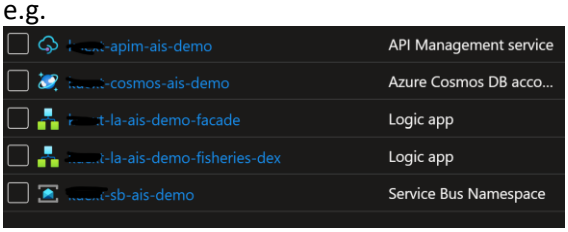
Area	Value
Container id	**vessel** Note: a table name is not plural
Partition key	**/vessel**

11. Click the **OK** button
12. Now your Container (database table) is created



Module 7 – Verify your infrastructure deployment

1. Go to your Resource Group
2. You now have 5 resources provisioned
   e.g.

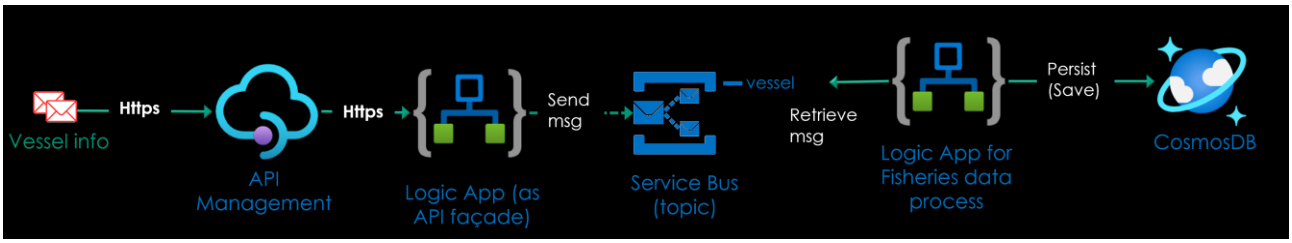


Exercise 3 – Configure Accesses (40min)

Topics	In this exercise, we will cover the following topics. • Managed identity and relevant RBAC (role-based access controls) • Assign Cosmos Built-in SQL Role to the managed identity • Use Cloud Shell to run Azure CLI commands in Azure Portal
Duration	• 40 min
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Understand required accesses

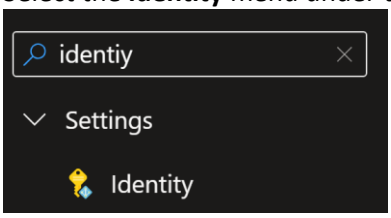
1. The overall integration flow looks like this
    a. Messages sent to the external endpoint (HTTPS) via API Management where Logic App endpoint is configured for API
    b. Logic App – Façade receives messages over HTTPS
    c. Logic App – Façade sends messages over to Service Bus for many-to-many messaging integration
    d. Logic App for Business is configured to monitor Service Bus activities on a specific topic, and retrieves when Service Bus receives a new message
    e. Logic App for Business saves the required vessel data to CosmosDB



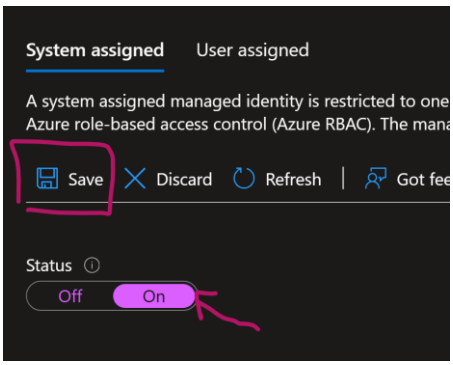
2. Now you need to configure required access for those resources above

Module 2 – Configure Access from Logic App Façade to Service Bus

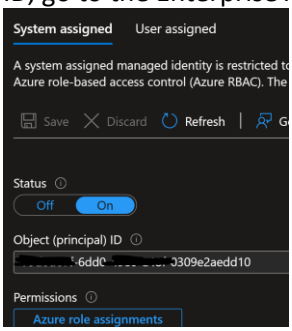
1. To access, there are several access methods available for Logic App to connect to Service Bus
    a. Shared Access Signature (SAS) Authentication using a Shared Access Key
    b. Microsoft EntraID Integrated using OAuth 2.0 token
    c. Client Certificate Authentication using mutual Certs
    d. Managed Identity (system-assigned or user-assigned)
2. A recommended methos is using **Managed Identity**
3. Go to your *Logic App – Façade*
4. Select the **Identity** menu under the *Settings*, or type identity in the search textbox in the side menu



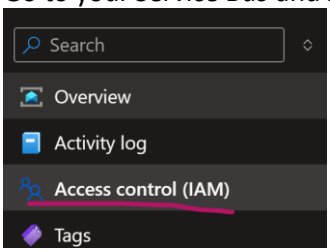
5. Enable *System-assigned identity* and click the **Save** button



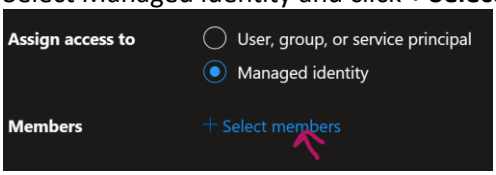
6. A new managed identity for Logic App Façade is created for you. If you want to check this in Entra ID, go to the Enterprise Applications menu and search the identity using the object ID



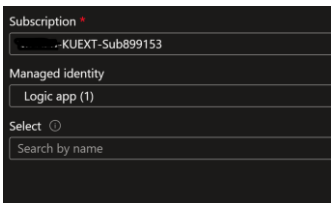
7. You need to give this Logic App managed identity the permission to connect to Service Bus
8. Go to your *Service Bus* and select the **Access control (IAM)** menu on the left



9. You can assign users or applications with a specific role(s) at the Service Bus namespace level or at each specific topic or queue level. Ensure to follow ***the Principle of Least Privilege***
10. For this lab, assign the Logic App managed identity with the *Reader* role at the namespace level
11. The Logic App Façade's job is to connect to Service Bus and send messages. Service Bus has the Job function roles, such as *Azure Service Bus Data Owner, Azure Service Bus Data Receiver, Azure Service Bus Data Sender*. Azure Service Bus Sender role is appropriate for the Logic App Façade.
12. However, the problem of assigning the data-level permissions is that you cannot search Service Bus(s) available in Logic Apps actions (i.e. displaying Service Bus namespaces, topics/queues), as you are not given the parent (namespace) level access.
    Thus, you assign the Logic App Façade with the *Reader* role at the Service Bus namespace level
13. For simplicity and lab purpose today, you assign the *Contributor* role to the Logic App Facade
14. Select **+ Add** menu and select **Add role assignment**
15. Select **Privileged administrator roles** tab and select **Reader** role
16. Click the **Next** button
17. Select Managed identity and click **+ Select members** button



18. Select **Logic app** (#number of Logic Apps) in the Managed identity dropdown
19. Select your Logic App façade resource



20. Click the **Select** button
21. Click the **Review + assign** button which will display the confirmation screen for the assigned role(s)
22. Click the **Review + assign** button again
23. Repeat this process to add *Logic App façade* to **Azure Service Bus Data Sender** role
24. Verify that the managed identity for your Logic App is added to the Reader role and Azure Service Bus Data Sender role by viewing the **Role assignments**

Module 3 – Configure Access from Logic App for Business to Service Bus

1. When a message is received at Service Bus, Logic App for Business needs to connect to Service Bus and retrieve the message from Service Bus topic
2. Create a **System-assigned managed identity** for *Logic App for Business (Fisheries)*, like you did for the previous exercise
3. Assign the **Reader** role and **Azure Service Bus Data Receiver** role to that managed identity for Logic App for Business in Service Bus
4. Verify that the managed identity for the Logic App for Business is added to those 2 roles

Module 4 – Configure Access from Logic App for Business to CosmosDB

1. When the Logic App for Business (Fisheries) completes processing the vessel data, it needs to persist in a database. In this lab, the data is saved to CosmosDB
2. To connect to CosmosDB from Logic App, there are several authentication methods available
   a. Managed Identity (System-assigned or User-Assigned)
   b. Using Connection String (SAS or Primary Key)
   c. EntraID App Registration using OAuth 2.0 token
3. Managed Identity authentication is recommended
4. CosmosDB separates the **Control plane** (i.e. Azure RBAC) and **Data Plane** (i.e. CosmosDB Built-I Data Contributor/Reader) access. Therefore, you need to assign the *SQL Role (Data Plane)* for reading/writing/querying documents in a database or executing stored procedures (Use data plane role-based access control - Azure Cosmos DB for NoSQL | Microsoft Learn)

Access Type	Purpose	Example	Applies to
Control Plane	Manage Cosmos DB resources (e.g., create DBs, containers)	Cosmos DB Account Contributor, Reader	Azure Resource Manager (ARM)
Data Plane	Read/write/query data inside containers	Cosmos DB Built-in Data Contributor, Reader	Cosmos DB SQL API

5. This is the summary of CosmosDB Built-in Data Plane Roles

Role Name	Role ID	Permissions
Cosmos DB Built-in Data Reader	00000000-0000-0000-0000-000000000001	Read-only access to data
Cosmos DB Built-in Data Contributor	00000000-0000-0000-0000-000000000002	Read and write access to data

6. Open **Cloud Shell**  at the top right in Azure portal. If you cannot access to Cloud Shell in Azure portal, use Azure CLI from your local machine, or VS Code Terminal
7. Run the following commands

```
cosmosAccountName="my-cosmosdb-account"
resourceGroup="my-resource-group"
logicAppIdentityObjectId="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"  # Replace with Logic
App's managed identity object ID
roleDefinitionId="00000000-0000-0000-0000-000000000002"  # Data Contributor
scope="/"

# Create the role assignment
az cosmosdb sql role assignment create \
  --account-name $cosmosAccountName \
  --resource-group $resourceGroup \
  --role-definition-id $roleDefinitionId \
  --scope $scope \
  --principal-id $logicAppIdentityObjectId
```

When declaring variables below, you can enter each line at a time in Cloud Shell, then hit the Enter button, and continue until all the variables.
Then run the last command ***az cosmosdb sql rol assignment create***.

Copy the following commands to Notepad and replace values with your resources.
Ensure selecting Logic App for Business (Fisheries).

Azure CLI commands	# Variables
	cosmosAccountName="<your-cosmosdb-account>"
	resourceGroup="<your-resource-group>"
	logicAppIdentityObjectId="<xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx>" # Replace with Logic App's managed identity object ID
	roleDefinitionId="00000000-0000-0000-0000-000000000002" # Data Contributor
	scope="/"
	
	# Create the role assignment
	az cosmosdb sql role assignment create \
	  --account-name $cosmosAccountName \
	  --resource-group $resourceGroup \
	  --role-definition-id $roleDefinitionId \
	  --scope $scope \
	--principal-id $logicAppIdentityObjectId

8. Verify that your SQL role assignment is successful. Run the following commands

```
az cosmosdb sql role assignment list \
  --account-name <your-cosmosdb-account> \
  --resource-group <your-resource-group> \
  --query "[?principalId=='<logic-app-object-id>']" \
  --output table
```

Azure CLI commands	az cosmosdb sql role assignment list \

	--account-name $cosmosAccountName \ --resource-group $resourceGroup \ --output table

Summary

ACHIEVEMENTS

After you have completed the Lab, you are now able to:

- ✓ Create System-assigned managed identity
- ✓ Assign RBAC to the managed identity
- ✓ Assign SQL role to the managed identity in CosmosDB

Exercise 4 – Implement Integration Logic for Logic App Façade (50min)

Topics	In this exercise, we will cover the following topics. • Add HTTP Request Trigger • Add an Action for Extracting Business Data • Add Conditional Check • Add a 'Case' Action for Switch Statement • Send Message to Service Bus • Validate Logic App logic
Duration	• 50 min
Tool(s)	• Azure portal

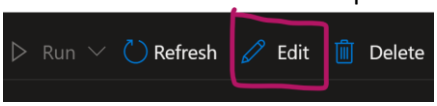
Requirements	You want to establish enterprise integration messaging capability in Azure that allows any type of data/message to be sent to the organisation.
	
	To accommodate that requirement, you will design the message payload to be flexible. You accept JSON formatted messages and enforce some required properties.
	• Category: Message category
	• Requestor: The name of person, organisation or entity requesting to connect to your Integration Platform in Azure
	• Payload: the actual business relate data is included
	
	A sample payload looks like this.
	Payload (message)
	`{`
	`"Category":"Fisheries",`
	`"Requestor":"NZ Fisheries Sub Company1",`
	`"Payload": {`
	`}`
	`}`
	
	The Fisheries team requires a particular set of the vessel information that will be included in the *Payload* property.
	e.g.
	`"Vessel": "Blabbergust",`
	`"VesselRegistrationNo": "NZ124-900",`
	`"OriginalPort": "Christchurch",`
	`"DestinationPort": "Wellington",`
	`"ETA": "2022-10-01T18:25:43.511Z",`
	`"DepartedDate": "2022-09-20T18:25:43.511Z",`
	`"Category": "Cargo ship",`
	`"Purpose": "Unloading cargo"`
	
	A sample JSON payload looks like this.

	{
	"Category":"Fisheries",
	"Requestor":"NZ Fisheries Sub Company1",
	"Payload": {
	"Vessel": "Blabbergust Hey",
	"VesselRegistrationNo": "NZ900-900",
	"OriginalPort": "Auckland",
	"DestinationPort": "Christchurch",
	"ETA": "2025-03-20T18:25:43.511Z",
	"DepartedDate": "2024-12-29T18:25:43.511Z",
	"Category": "Fishing Boat Heaven",
	"Purpose": "Unloading cargo"
	}
	}
	
	The processes of Logic App Façade include:
	• Receive a message(s) in JSON format over HTTPS
	• Extract the business data within the message payload
	• Check the category of the message
	• If the category is for the Fisheries team, then send the message to Service Bue topic called *"vessel"*

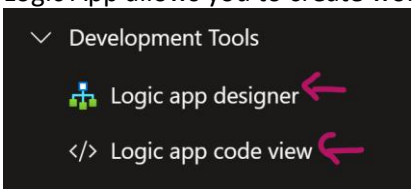
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Add HTTP Request Trigger

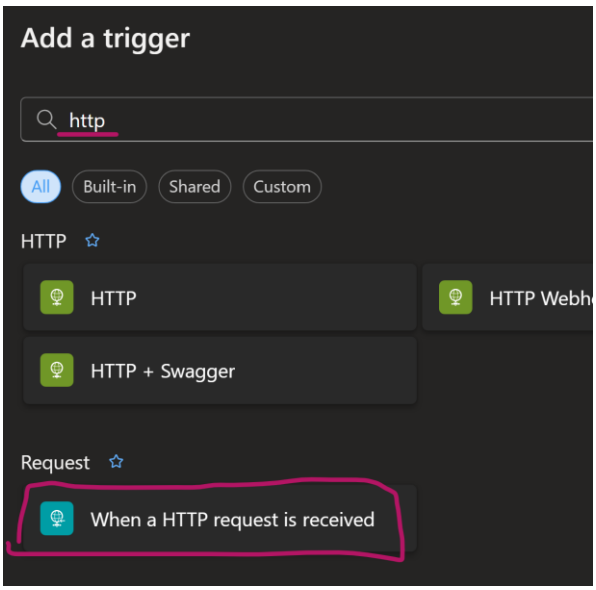
1. Select your façade Logic App, **[your short name]-la-ais-labs-façade**
2. Click the **Edit** button at the top



3. Logic App allows you to create workflow logic in *GUI (graphical user interface)* or in *code* (JSON)

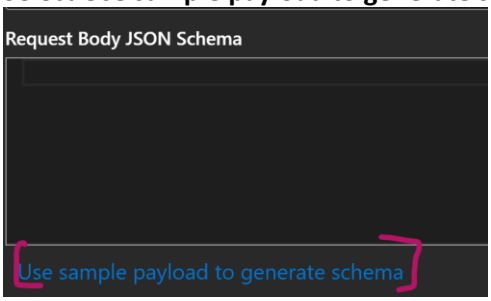


4. Logic App always container one *Trigger* and one or more than one *Actions*
5. Façade's job is to receive messages for vessel information and pass them over to Service Bus
6. To receive messages over HTTPS, you simply need to create a HTTP Request trigger
7. Click **Add a trigger** button on the editor. The side panel will appear
8. Type ***http*** in the search textbox and find the trigger for **When a HTTP request is received**

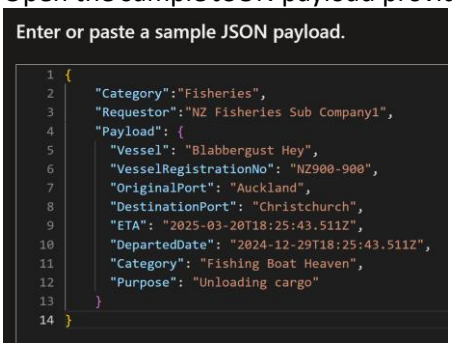


9. You can add the **Description** field. E.g. ***"Receiving integration messages over HTTPS/HTTP"***
10. At this stage, any message via HTTP request can be accepted. You don't want any type of messages to be sent to you.
11. Let's put some JSON schema validation
12. Change the **Method** dropdown to ***POST*** only

13. Select **Use sample payload to generate schema** link



14. Open the sample JSON payload provide, copy the contents and paste it onto the editor



15. Remove anything under the Payload property. The JSON payload looks like this

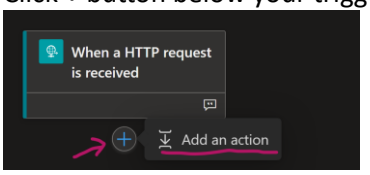
16. Click **Done** button. The **Request Body JSON Schema** is automatically generated for you

17. Click **Save** button to save your progress

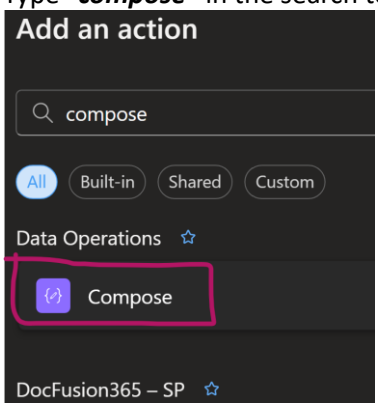
> When you're working on your Logic App in the Azure portal, remember to save your changes often.
>
> The portal doesn't automatically save what you're doing - so if you navigate away or close the browser, you could lose your progress.
>
> Just click ***"Save"*** regularly to make sure your updates are kept

Module 2 – Add An Action for Extracting Business Data

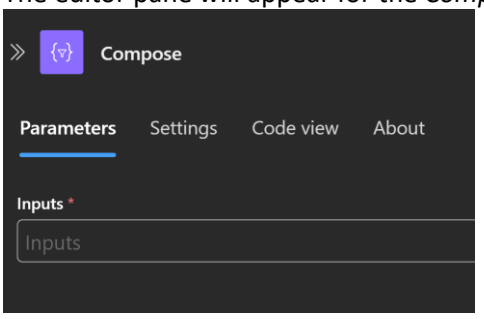
1. Click **+** button below your trigger and select **Add an action** button



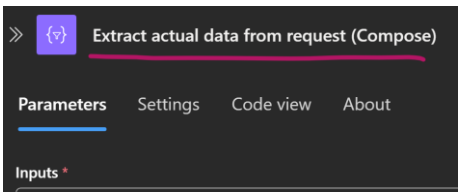
2. Type ***"compose"*** in the search textbox and select the **Compose** action from the *Data Operations*



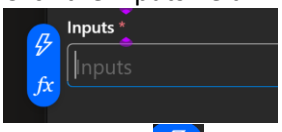
3. The editor pane will appear for the *Compose* action



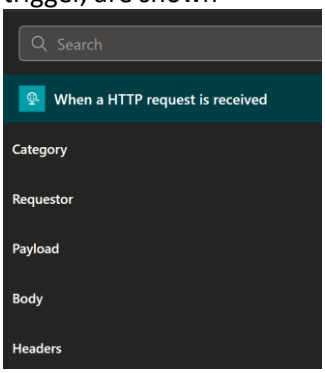
4. Let's rename the step to make it readable and supportable
5. Click the word *"Compose"*. You will be able to change the name of the action
6. Change the value to ***"Extract actual data from request (Compose)"***



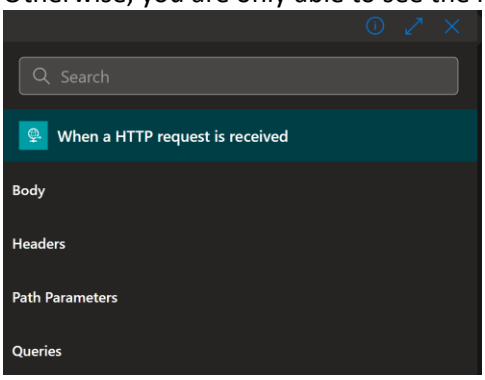
7. Click the Inputs field. The floating menu will appear



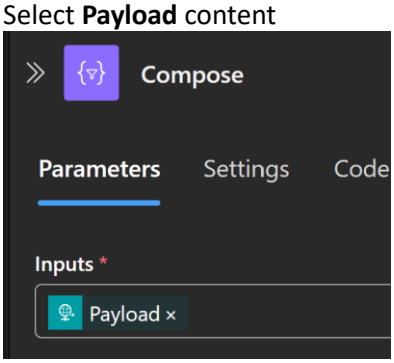
8. The fist icon allows you to enter the *dynamic content (data)* from previous step(s). And the second icon allows you to insert Logic App *expression* (built-in functions)
9. You want to use the received message (data) and send that to Service Bus
10. Click the **Dynamic Content** icon. Dynamic contents from the previous step, HTTP request trigger, are shown



11. Because you have added the JSON schema validation, you can access to a specific property in Logic App – *Category, Requestor*, and *Payload*.
Otherwise, you are only able to see the HTTP request *Body* like below



12. Select **Payload** content



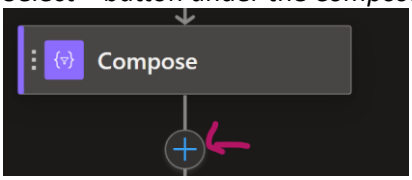
13. Let's look at the code and see how the Logic App logic is actually implemented
14. Click **{} Code view** button at the top menu, or **Code view tab** within your selected action
15. Find the *"actions"* section and *"Extract actual data from request (Compose)"* action. The *"inputs"* value is set to ***"@triggerBody()?['Payload']"***

```
"Extract_actual_data_from_request_(Compose)": {
    "type": "Compose",
    "inputs": "@triggerBody()?['Payload']",
    "runAfter": {}
},
```

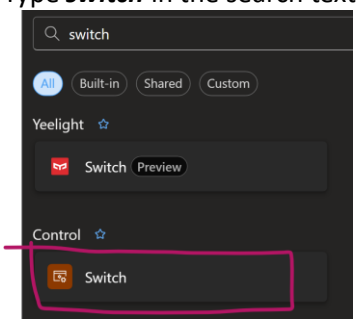
16. Let's break this down, so it's easy to understand
    a. **@triggerBody():** This gets the entire body of the incoming HTTP request - basically, the full JSON message
    b. **?** (question mark): This is a safety check. It tells Logic Apps:
       "Only try to get the next part if the body actually exists."
       This helps avoid errors if the message is missing or empty
    c. **['Payload']:** This is how you access a specific property inside the JSON.
       In this case, you're saying:
       "Give me the value of the Payload property from the body."
17. Save the progress

Module 3 – Add Conditional Check

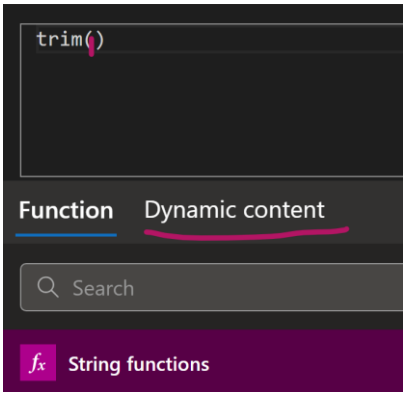
1. The next step is to check if given Category property = "Fisheries" or not
2. Select **+** button under the *Compose* step and select **Add an Action**



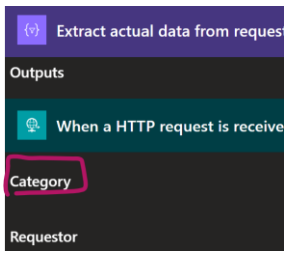
3. Type ***switch*** in the search textbox and select the **Switch** action under the *Control*



4. You can use *"If"* conditional statement, instead. But for this exercise, you use *"switch"* conditional statement
5. Rename the action to ***"Check message category (Switch)"***
6. Click the **On** field and select the **Expression** icon
7. Enter **trim(** in the editor. The closing bracket ")" is automatically added
8. While the mouse cursor is still inside the bracket, select the Dynamic content tab



9. Select Category content from the "When a HTTP request is received" step



10. The editor contains the expression like this

```
trim(triggerBody()?['Category'])
```

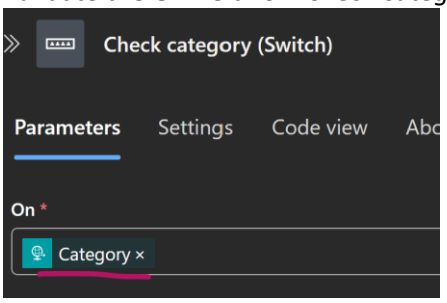
Working with *'Switch'* action in Consumption Logic Apps have some quarks.

Common pitfalls:

- 'Case' values must be static strings -> no Dynamic content or expressions
- 'Case' values must be typed manually – pasting from Dynamic content can introduce hidden tokens
- 'Switch' expression must be simple

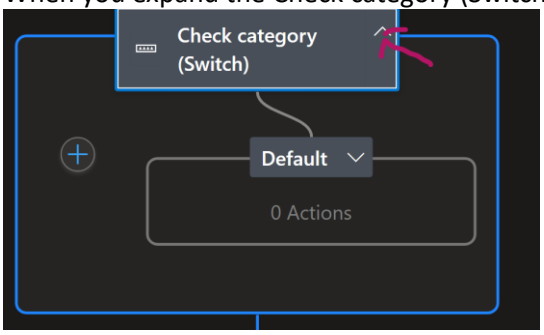
To doubly ensure no spaces are added for the 'Switch' check, the expression, **trim()**, is used here

11. Validate the **On** field for *"Check category (Switch)"* step is updated with the correct value

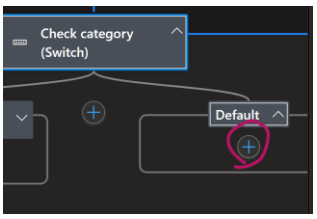


Module 4 – Add a 'Case' Action for Switch Statement

1. When you expand the Check category (Switch) step, the ***default case*** is already added automatically



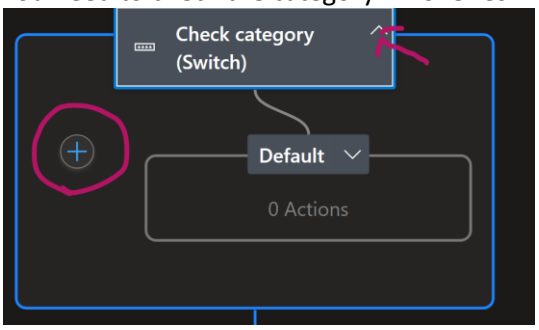
2. For now, just send an email to yourself
3. Insert an action for the Default case



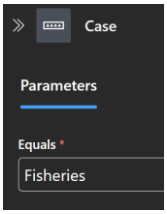
4. Type **outlook** in the search textbox
5. You have 2 types of Outlook connectors available – *Outlook.com* and *Office 365 Outlook*. Outlook.com is used for your personal email account and Office 365 Outlook is used for work/school email account, respectively.
   For this exercise, make sure using your company email server, Office 365 Outlook
6. Select **Send an email (V2)** action within *Office 365 Outlook* connector
7. Enter the following details

Area	Value
To	**[your email address]**
Subject	AIS demo – new message for integration
Body	[Enter some message body. Try using the dynamic content etc…]

8. Sign in using your company's email address
9. You need to check the category = Fisheries. Add a new case by clicking **+** button within Switch step

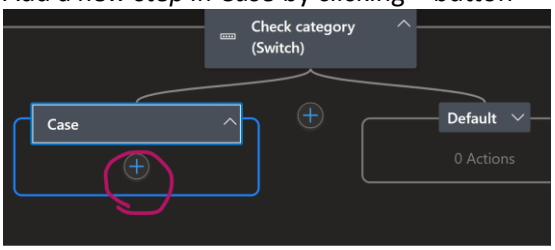


10. Rename the action to ***"Fisheries Case"***
11. Enter the value ***"Fisheries"*** for the **Equals** field

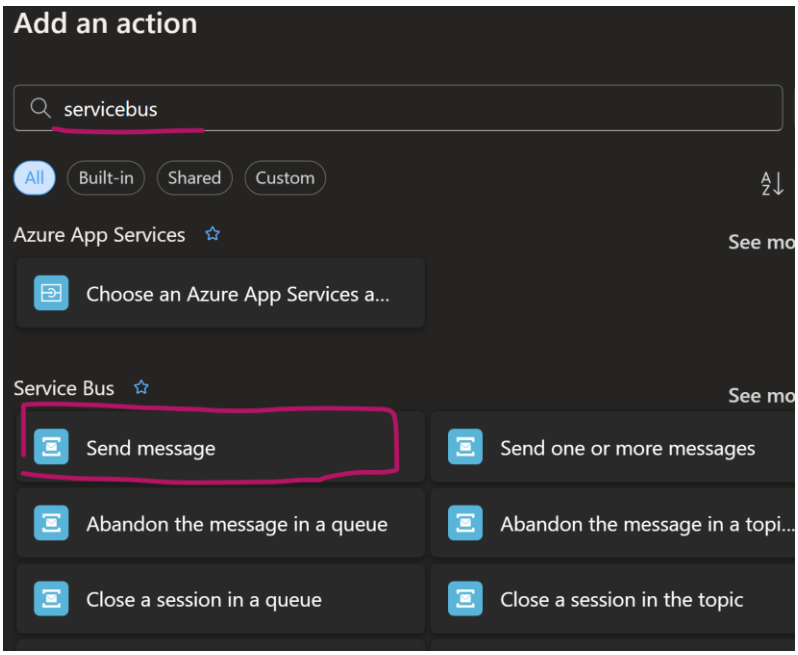


Module 5 – Send Message to Service Bus

1. Add a *new step in Case* by clicking **+** button



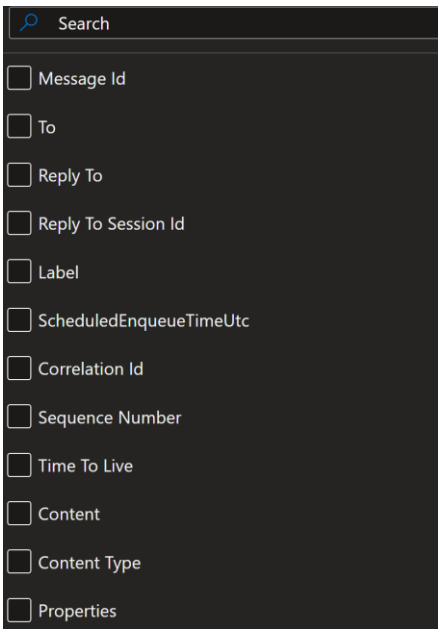
2. Type ***"servicebus"*** in the search textbox. Various actions for Service Bus will be displayed
3. Select **Send message** action



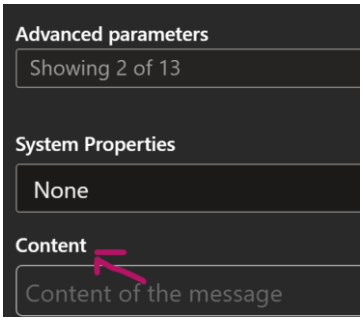
4. Because you are creating a Service Bus action first time, you need to configure the connection resource first
5. Enter the following details

Area	Value
Connection Name	**[your short name]-ais-lafacade-to-sb-conn**
Authentication Type	**Logic Apps Managed Identity**
Namespace Endpoint	**sb://[your SB namespace].servicebus.windows.net/** Note: You can get the *ServiceBus hostname* in the *Overview* menu for your ServiceBus. Prepend ***"sb://"*** syntax in front

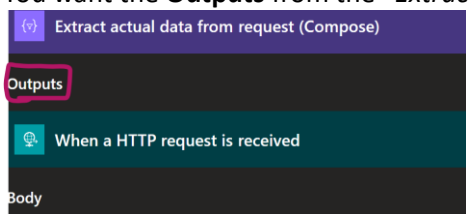
6. Click **Create New** button
7. You are back to the editor pane for the *Send message* action
8. Rename the action to *"Send message to Service Bus"*
9. Select ***"vessels(topic)"*** from the **Queue/Topic name** dropdown
10. Select **Advanced parameters** dropdown. Here, various message properties can be included as part of Service Bus message



11. Select **Content**
12. Move your mouse to the editor, outside of the parameter window. The Content property is added now



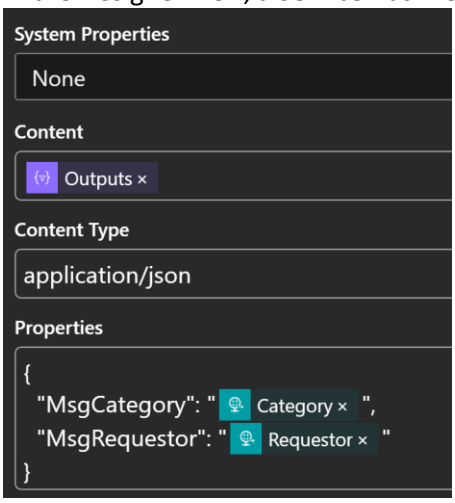
13. A message content should be the actual business data which we extracted in the previous step
14. Click the **Content** field and select the **Dynamic Content icon**
15. You want the **Outputs** from the *"Extract actual data from request (Compose)"* action



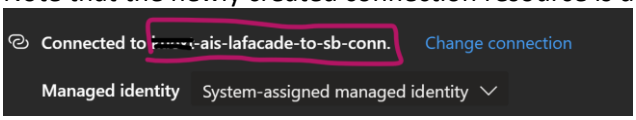
16. Select **Outputs**
17. You need to set the message *content type*. Select **Content Type** property from the *Advanced parameters* dropdown
18. Enter ***"application/json"*** in the Content Type property
19. Service Bus supports creating *customer properties* for a message. Let's add some custom properties to retain the original properties for *Category* and *Requestor*
20. Select **Properties** from the *Advanced parameters* dropdown
21. *"Properties"* field is an object containing key-value pairs. An object is represented with enclosed *curly brackets* ***"{}"*** in JSON.
22. Enter the following JSON to add 2 keys

```
{
  "MsgCategory": "<category-property>",
  "MsgRequestor": "<requestor-property>"
}
```

23. Replace <category-property> with the actual Category property, <requestor-property> with the actual Requestor property using the Dynamic content
24. In the Designer view, a Service Bus message contains properties like this

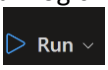


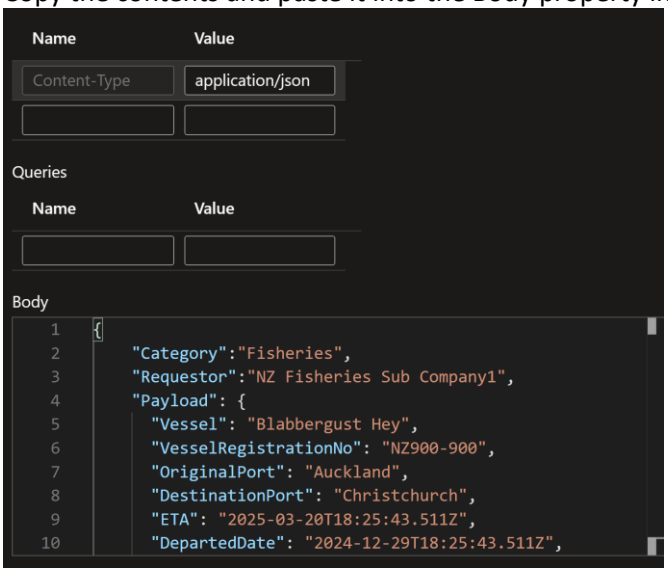
25. Note that the newly created connection resource is displayed at the bottom of the editor pane



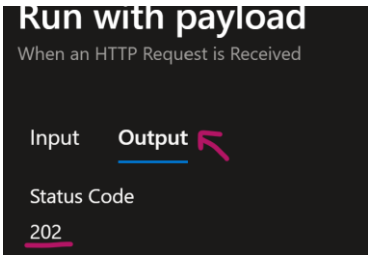
26. If you need to change or create another connection, click **Change connection** link/button
27. Save the progress

Module 6 – Validate Logic App logic

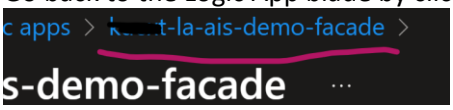
1. Let's validate if your Logic App façade works as expected
2. Select **Run** button  at the top and select **Run with payload**
3. Open one of samples provided (e.g. ais-messaging-sample1.json) in Notepad
4. Copy the contents and paste it into the Body property in the editor pane



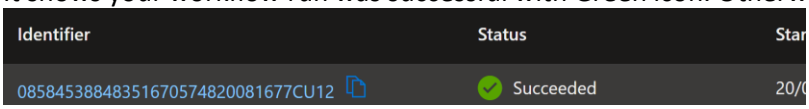
5. Click **Run** button
6. Check if your **Output** returns 202 (success)



7. Go back to the Logic App blade by clicking the hyperlink at the top of the screen



8. It shows your workflow run was successful with Green icon. Otherwise, it shows with Red icon



9. Click the successful run URL. You can see the successful execution paths and also check inputs and outputs of each action

Summary

ACHIEVEMENTS

After you have completed the Lab, you are now able to:

- ✓ Add HTTP Request trigger in Logic App
- ✓ Add actions in Logic App
- ✓ Rename an action
- ✓ Use built-in and managed Logic App connectors
- ✓ Create connections to connect to another resources (e.g. Service Bus) from Logic App
- ✓ Use conditional check within the workflow
- ✓ Send emails from Logic App
- ✓ Validate the workflow logic by running with sample payload
- ✓ Check run history and execution details

Exercise 5 – Implement Integration Logic for Logic App For Business (60min)

Topics	In this exercise, we will cover the following topics. • Add a Trigger for Service Bus New Message • Add an Action to Parse Message Content (JSON) • Validate Service Bus Message and Logic App Trigger • Add an Action for Parsing Message Properties • Add an Action for Composing Required Data for Save • Save the Vessel Data to CosmosDB • Validate Messaging Integration
Duration	• 60 min
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]
Scenario	Now that you have extracted the business-related data (vessel information) from messages. The Fisheries team wants to persist that data in their database. In this exercise, you will store the vessel information in their database in Cosmos DB. The processes of Logic App for Business include: • Triggers the Logic App when a message arrives in a specific subscription of a topic • Parse Fisheries Data (JSON) to align with the business requirement • Parse message properties, if required • Compose the message, so that it can be saved in CosmosDB • Save the message in the business database in CosmosDB

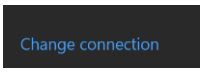
Module 1 – Add a Trigger for Service Bus new message

1. In Azure portal, select Logic App for Business (Fisheries)
2. Click the **Edit** button at the top centre
3. Select **Add a trigger** 

4. Type **"servicebus"** in the search textbox and find the action for **"When a message is received in a topic subscription (auto-complete)"**
5. Enter the following details

Area	Value
Topic Name	**Vessels**
Topic subscription Name	**allVessels**
How often do you want to check fo items? (expand the section)	**3 Second**

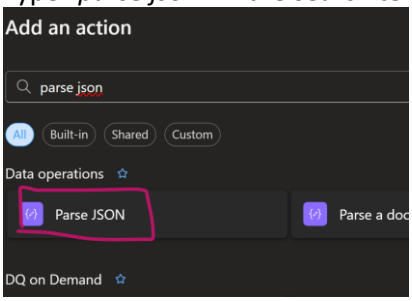
6. Scroll down and check what connection is used for connecting to Service Bus

7. If your Logic App is reusing the existing connection (e.g. connection created for Logic App Façade, or someone else's connection within the same Resource Group), you need to create your own and configure that
8. Click **Change connection** 

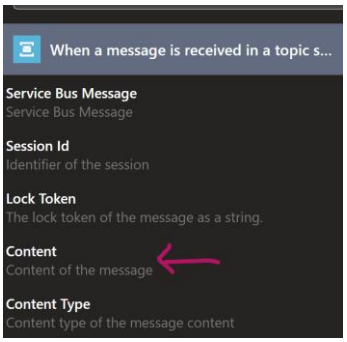
9. Click **Add new** button
10. Configure a new connection like you did previously
11. Save your progress

Module 2 – Add an Action to Parse Message Content (JSON)

12. **Add an action** by clicking the **+** icon
13. Type *"parse json"* in the search textbox and select the **Parse JSON** action



14. The purpose of parsing JSON data is to decrypt the message content and apply the schema validation. So, each property within the content will be accessible for later steps or actions
15. Click the *Content* field and select the **Dynamic content** icon 
16. Select **Content**



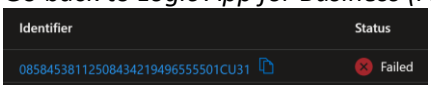
17. Copy the following JSON schema and paste it to the **Shema** field

```
{
 "properties": {
  "Category": {
   "type": "string"
  },
  "DepartedDate": {
   "type": "string"
  },
  "DestinationPort": {
   "type": "string"
  },
  "ETA": {
   "type": "string"
```

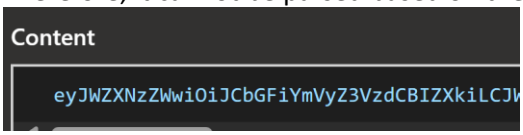
```
        },
        "OriginalPort": {
          "type": "string"
        },
        "Purpose": {
          "type": "string"
        },
        "Vessel": {
          "type": "string"
        },
        "VesselRegistrationNo": {
          "type": "string"
        }
      },
      "type": "object"
    }
```

Module 3 – Validate Service Bus Message and Logic App Trigger

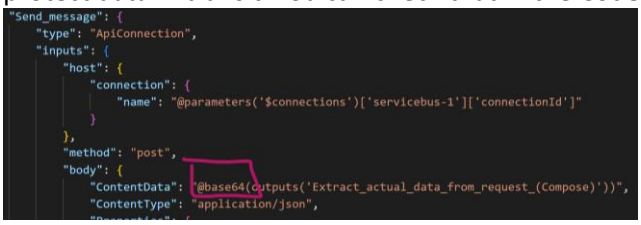
1. Let's validate the logic here
2. Open another browser tab, go back to *Logic App Façade* in Azure portal. Run the Logic App workflow with payload, just like you did in the previous exercise
3. Go back to *Logic App for Business (Fisheries)*. The history run shows the workflow run and it failed



4. Click the link for the workflow run and find where the Red icon is shown
5. Select the failed step and troubleshoot
6. The problem here is that the message **Content** sent by Service Bus is encoded/encrypted. Therefore, it cannot be parsed based on the specified JSON schema

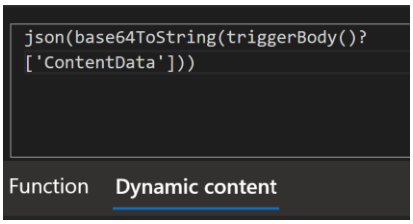


7. When sending a message from Logic App, the message is encrypted with Base64 automatically to protect data in transit. You can check that in the Code view for Logic App façade

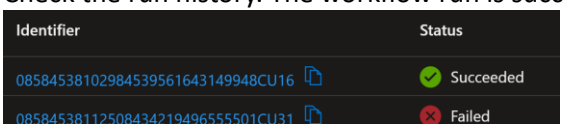


8. Therefore, when you receive a message from Service Bus, you need to decode/decrypt the message
9. Go back to the Editor for Logic App for Business (Fisheries)
10. Select **Parse JSON** step and remove *Content* data in the **Content** field
11. Select Expression icon 
12. You need to decode the message first to simple string type, then convert that string into JSON format

13. Enter ***json()***, with the mouse cursor still set within the brackets enter ***base64ToString()***, and then select *Dynamic content tab* and select **Content** object/data
14. It looks like this

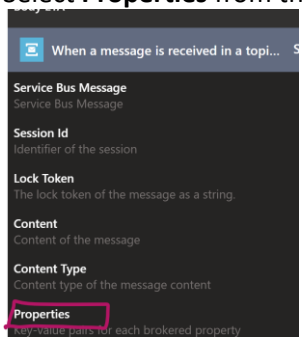


15. Click **Add** button
16. Let's validate it again by sending a message from Logic App Façade
17. Check the run history. The workflow run is successful this time



Module 4 – Add an Action for Parsing Message Properties

1. In the *Edit* mode for Logic App for Business (Fisheries), add another action below *Parse JSON* step
2. We want to expose the message properties, so those can be accessed at later steps within the workflow. Find **Parse JSON** action and select it
3. Rename it to ***"Parse Msg properties (JSON)"***
4. Select **Properties** from the previous step for the *Content* field



5. Insert the following JSON schema

```
{
  "properties": {
    "MsgCategory": {
      "type": "string"
    },
    "MsgRequestor": {
      "type": "string"
    }
  },
  "type": "object"
}
```

Module 5 – Add an Action for Composing Required Data for Save

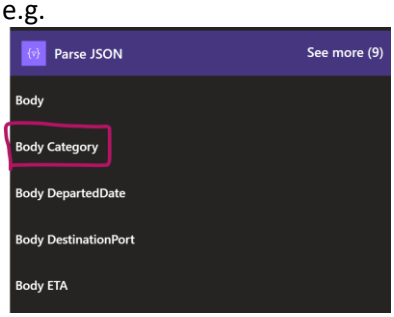
1. Add another **Action** under the *Parse Msg Properties* step
2. Find **Compose** action under the Data Operations connector and select it
3. Rename it to ***"Prepare Bus Data (Compose)"***
4. The business requirement specifies additional property called ***"messageReceivedOn"*** to be saved in addition to the properties available from message integration. Every document (data) in Cosmos DB container (database) must have an ***"id"*** property

> QUICK TIPS
>
> Every document in a Cosmos DB container must have an **id (string)** property. The *id* must be **unique** within the same partition key value. It is not auto-generated value.
>
> If you don't explicitly set it, Cosmos DB will **not** create one for you

5. The business data (JSON) required here looks like this

```
{
  "Category": "",
  "DepartedDate": "",
  "DestinationPort": "",
  "ETA": "",
  "OriginalPort": "",
  "OriginalRequestor": "",
  "Purpose": "",
  "Vessel": "",
  "VesselRegistrationNo": "",
  "messageReceivedOn": "",
  "id": ""
}
```

6. Copy the above JSON and paste it in the **Inputs** field
7. Now you need to generate the data one by one. Place the mouse cursor between the double quotations for a selected property value
8. Select Dynamic content icon and find relevant content
   e.g.



9. The *inputs* field looks like this

```
Inputs *
{
  "Category": " Body Category × ",
  "DepartedDate": " Body DepartedDate × ",
  "DestinationPort": " Body DestinationPort × ",
  "ETA": " Body ETA × ",
  "OriginalPort": " Body OriginalPort × ",
  "OriginalRequestor": " Body MsgRequestor × ",
  "Purpose": " Body Purpose × ",
  "Vessel": " Body Vessel × ",
  "VesselRegistrationNo": " Body VesselRegistratio... × ",
  "messageReceivedOn": "",
  "id": ""
}
```

10. For the *messgeReceivedOn* property, add an **Expression** using **formatDateTime()**

```
formatDateTime(utcNow(), 'yyyy-mm-dd')
Function   Dynamic content
```

11. For the *id* property, we use built-in function using **guid()** to auto-generate GUID value
12. Add an **Expression** for the id property

```
guid()
Function   Dynamic
```

13. The final dataset looks like this

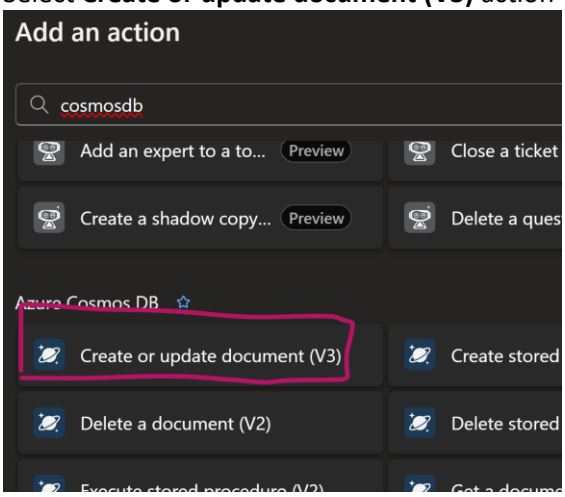
```
Inputs *
{
  "Category": " Body Category × ",
  "DepartedDate": " Body DepartedDate × ",
  "DestinationPort": " Body DestinationPort × ",
  "ETA": " Body ETA × ",
  "OriginalPort": " Body OriginalPort × ",
  "OriginalRequestor": " Body MsgRequestor × ",
  "Purpose": " Body Purpose × ",
  "Vessel": " Body Vessel × ",
  "VesselRegistrationNo": " Body VesselRegistratio... × ",
  "messageReceivedOn": " fx formatDateTime(...) × ",
  "id": " fx guid() × "
}
```

14. Save your progress

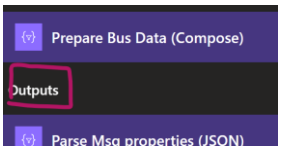
Module 6 – Save The Vessel Data to CosmosDB

1. Add a new **Action** below the *Prepare Bus Data (Compose)* step
2. Find CosmosDB connector and available actions

3. Select **Create or update document (V3)** action



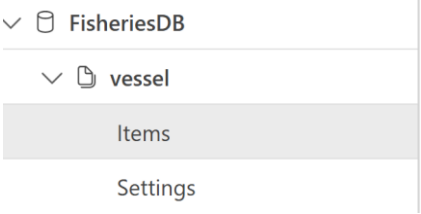
4. Enter the following details

Area	Value
Azure Cosmos DB Account Name	**[name of your CosmosDB]**
Database ID	**FisheriesDB**
Collection ID	**Vessel**
Document	**Outputs** (Dynamic content from the Prepare Bus Data step) 

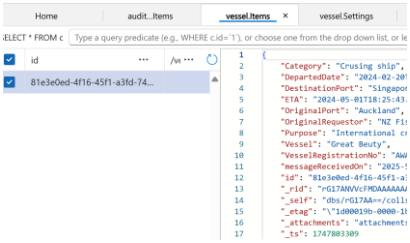
5. Check the connection to CosmosDB, whether Logic App reuses the existing connection (e.g. Logic App Façade or someone else's connection). If so, change the connection by adding a new connection for your Logic App
6. Additionally, add an action to notify you for logging purpose (e.g. sending an email for a new message)

Module 7 – Validate Messaging Integration

1. Go back to Logic App Façade and run the workflow with sample payload
2. Validate whether the end-to-end integration processes and flows work as expected
3. Check on the run history for Logic App Façade
4. Check on the run history for Logic App for Business (Fisheries)
5. Check whether a new vessel data is saved in the Vessel database in CosmosDB
6. In CosmosDB resource, select Data Explorer menu on the left and select your database



7. The new message is correctly saved in the database in CosmosDB



Summary

ACHIEVEMENTS

After you have completed the Lab, you are now able to:

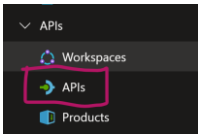
- ✓ Add a Trigger for Service Bus new message
- ✓ Parse Service Bus messages (JSON)
- ✓ Understand how a message is encoded and decoded
- ✓ Compose JSON dataset
- ✓ Create and change connections to other Azure services using Managed Identity
- ✓ Save a new message (data) to CosmosDB

Exercise 6 – Configure External Endpoint (15min + Optional)

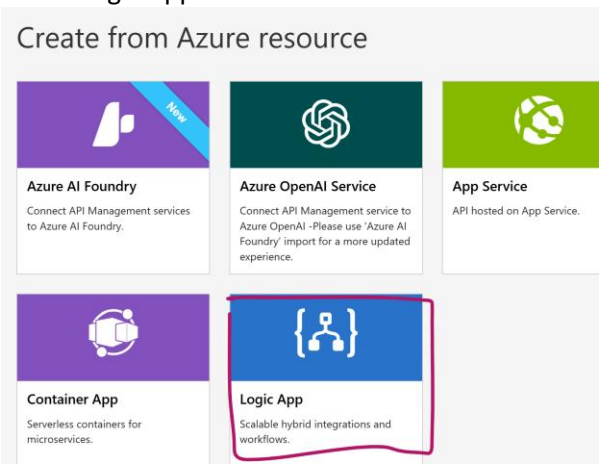
Topics	In this exercise, we will cover the following topics. • Publish Logic App Façade Endpoint in APIM • Validate End-to-end Integration Flow • (Optional) Enable and Use Developer Portal • (Optional) Configure API Policy(s)
Duration	• 15 min + Optional
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]
Scenario	At the moment, we have been testing the integration processes using Logic App Façade directly. We can configure the Logic App Façade endpoint to be available ane exposed to the external party. However, we need to consider building de-coupled integration platform and capability and don't plant to do point-to-point integration. We will introduce API Management in front of Logic App Façade. Having APIM give an abstraction layer to hide the details of backend services and underlying infrastructure for those services. At the same time APIM gives more control for API behaviours, security, load balancing and centrally managed API environment.

Module 1 – Publish Logic App Façade endpoint in APIM

1. In Azure port, select your APIM that is already provisioned
2. Select APIs under the AIPs section on the left menu



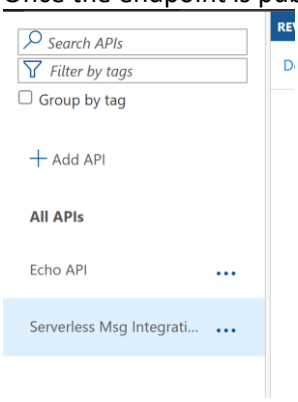
3. Scroll down the main pane and find **Create from Azure resource** section
4. Select Logic App icon



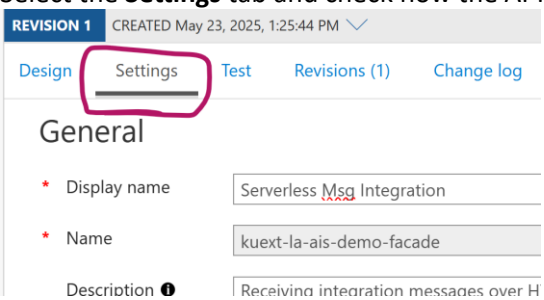
5. Click Browse button and find your *Logic App Façade*
6. Change the *mode* to **Full**
7. Enter the following details

Area	Value
Display name	**[your short name] Serverless Msg Integration**
Name	**[your Logic App Name]**
Description	**Receiving integration messages over HTTPS**
API URL suffix	**Vesseldx**
Tags	(Leave it as blank)
Products	**Unlimited**
Gateways	**Managed**

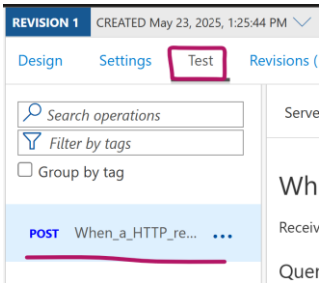
8. Click **Create** button
9. Once the endpoint is published in APIM, select that on the left menu



10. Select the **Settings** tab and check how the API is configured



11. At the moment, the endpoint for Logic App Façade is configured with
    a. Logic App URL is automatically generated for you
    b. Pass-through (no authentication check)
    c. URL is enforced with HTTPS
    d. API requires Subscription
    e. App Insights is not enabled for application-level monitoring
12. Let's validate your end-to-end integration flow
13. Select **Test** tab. This is where you can quickly submit API request(s) to test in Azure portal



14. We currently have one API operation defined (Logic App trigger)
15. Scroll down the editor on the right and find **Request body** section
16. Open one of the sample JSON files provided for the workshop in Notepad, copy the contents, and replace the existing Body content with that

Request body

Raw Binary

```
1  {
2      "Category":"Fisheries",
3      "Requestor":"NZ Fisheries Sub Company1",
4      "Payload": {
5        "Vessel": "Blabbergust Hey",
6        "VesselRegistrationNo": "NZ900-900",
```

17. Click **Send** button
18. Scroll down the editor. You will see successful HTTP response with the status = 202

HTTP response

Message Trace

```
HTTP/1.1 202 Accepted
cache-control: no-cache
content-length: 0
date: Fri, 23 May 2025 01:38:02 GMT
```

Module 2 – Validate End-to-end integration flow

1. Go back to *Logic App Façade* and check the run history
2. Go back to *Logic App for Business (Fisheries)* and check the run history
3. Go back to your *CosmosDB* and check that the new data is saved correctly (i.e. Data Explorer feature)
4. If you cannot view the items in your CosmosDB due to the error of *RBAC permission*, you need to assign yourself with CosmosDB built-in role to access to the data plane

```
1:47 PM  Request blocked by Auth[redacted]-cosmos-ais-demo : Request is blocked because principal [redacted] does not have required RBAC permissions to perform action
[Microsoft.DocumentDB/databaseAccounts/readMetadata] on resource [/]. Learn more: https://aka.ms/cosmos-native-rbac.
ActivityId:[redacted]036-bf4c-0e5f33d3b43a, Microsoft.Azure.Documents.Common/2.14.0
```

5. Run the following Azure CLI command in Cloud Shell in Azure portal and check if you're assigned with Data Plane permission (Write or Read)

```
az cosmosdb sql role assignment list --account-name <your-cosmosdb-name> --resource-group <your-rg-name>
```

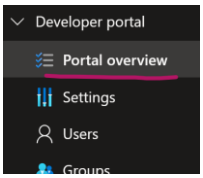
6. If your **PrincipalID** is not listed, add yourself to the built-in Write Role

```
az cosmosdb sql role assignment create \
--account-name <your-cosmosdb-name> \
--resource-group <your-rg-name> \
--scope "/" \
--principal-id <your principalID> \
--role-definition-id "00000000-0000-0000-0000-000000000002"
```

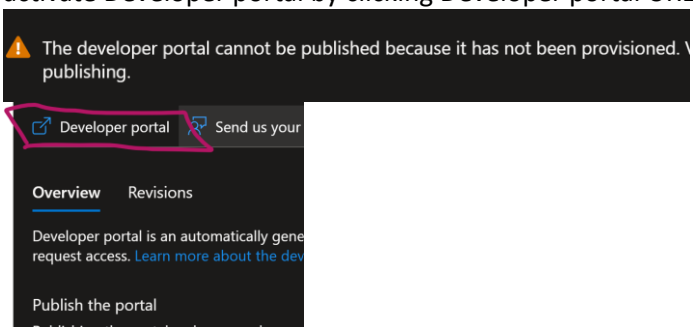
7. Go back to your Cosmos DB, refresh the browser, and select *Data Explorer* menu
8. You can see the new vessel information is saved correctly

Module 3 (Optional) – Enable and Use Developer Portal

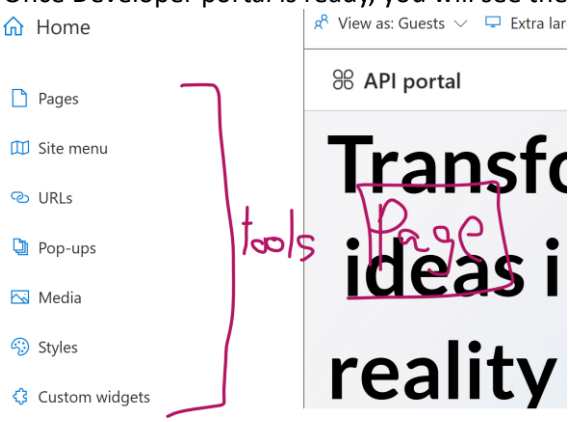
1. How do you let and allow the end-users or consumers to find your right API(s)? We will introduce **Developer Portal** for that purpose
2. Select **Portal overview** under the *Developer portal* section on the left menu



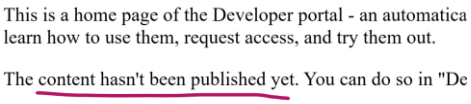
3. At this point, Developer Portal has not been provisioned for you yet. You need to initiate and activate Developer portal by clicking Developer portal URL at the top



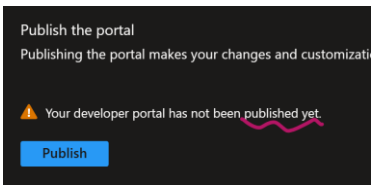
4. This process kicks off provisioning of Developer portal Wait for a minute or two
5. Once Developer portal is ready, you will see the editor site and screens for Developer portal



6. Copy the URL
7. Open a new *InPrivate browser window*, paste the URL address and see what happens
8. You only see some text messages on the screen. This is because your *Developer portal site* has **not been published yet**



9. Go back to *APIM* resource and refresh the browser to get the current state of Developer portal

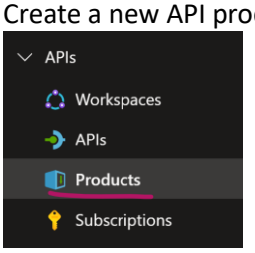


10. Click **Publish** button and **Enable CORS** button
11. Go back to InPrivate window and refresh the browser
12. This is how the end-user or consumers of APIs will see Developer Portal published by you/your organisation

13. Select **APIs** menu at the top right. You will see newly created API endpoint for Logic App Façade (Serverless Msg Integration)
14. Select the API. You will be able to check the API specifications and details. You can also test this API from Developer Portal by clicking **Try this operation** button
15. The Logic App Façade API endpoint required an API Subscription for access. You need to generate a subscription key for your API. Subscription keys can be scoped at different levels depending on how you want to control access

Scope Type	Description	Use Case
All APIs	Grants access to **all APIs** in the APIM instance.	Internal tools or trusted services needing full access.
Single API	Grants access to **one specific API** and all its operations.	Fine-grained access control for individual APIs.
Product	Grants access to **a group of APIs** bundled into a product.	Managing access tiers, quotas, and terms of use. Use this scope for most external developer scenarios

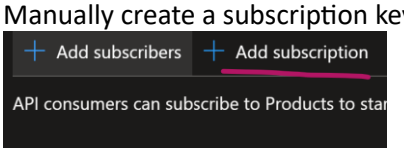
16. Create a new API product by selecting **Products** under the *APIs section* on the left menu



17. Click **+ Add** Button
18. Enter the following details

Area	Value
Display Name	**Fisheries Data Exchange**
Id	**fisheries**
Description	This product enables trusted external partners to securely submit vessel-related data to the Fisheries group. It provides access to APIs designed for reporting vessel activity, location, and compliance information. The data collected supports operational monitoring, regulatory enforcement, and sustainable fisheries management.
Legal terms	Access is restricted to authorized partners. All submitted data must comply with national fisheries regulations and data privacy standards. Misuse or unauthorized access may result in termination of access
Require subscription	(checked)
APIs	(Add Logic App Façade API endpoint)

19. Click **Create** button
20. Select the newly created Product
21. Select **Subscriptions** under the *Product* section on the left menu
22. Manually create a subscription key by selecting Add subscription button at the top



23. Enter the following details. At this stage, we will not associate a specific user(s) to this subscription

Area	Value
Name	**General-Vessel**
Display Name	**General Subscription for Vessel Data**

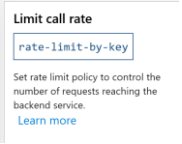
24. Click **Create** button
25. Click **…** button at the far right corner of the newly created product, and select **Show/hide keys**
26. Copy the Primary key
27. Go back to InPrivate Window for Developer portal
28. Use the newly created subscription key and send test API data using Developer Portal feature
29. Validate integration is completed and the new data is successfully saved in CosmosDB

Module 4 (Optional) – Configure API Policy(s)

1. So far, your API is configured for messaging integration, Developer portal available for external party(s) and you are ready to go
2. However, we want to make sure that your API covers NFRs, such as capacity, usage restrictions
3. We want to protect the Logic App API from excessive usage while allowing for a reasonable amount of traffic
4. You will create 2x API policies – ***Rate Limits and Quota policies***

Policy	Description
Rate Limits	These are used to protect against **short and intense bursts** of traffic. They limit the number of API calls that can be made in a short period, such as per minute or per second. This helps prevent **sudden spikes** in traffic that could overwhelm your backend services
Quotas	These are used to control the **total number** of API calls or the **amount of data** transferred over a longer period, such as per day or per month. Quotas are useful for managing overall usage and can be set differently for various subscription tiers, allowing for monetization strategies.

5. Select Logic App Façade API. You can directly implement API policies, or use built-in templates
6. In the *Design* tab, select **Inbound processing** and select **+ Add policy** button
7. Select Limit call rate template and configure the policy with the following details



Area	Value
Number of calls	**1000**
Renewal period	**60**
Counter key	**API subscription**
Increment condition	**Any request**

8. Click **Save** button
9. Repeat adding another policy for **Quotas**
10. Select **<>** button or scroll down to find **Other policies**

11. Let's try **Copilot**  to help you write the Quotas policy
12. Enter your prompt, e.g. *"help me create usage quotas policy here in addition to the existing policy(s)"*
13. Copy and paste the suggestion to the editor and save the change
14. Try out other prompts in Copilot (e.g. explain what 2 policies are)

Summary

ACHIEVEMENTS

After you have completed the Lab, you are now able to:

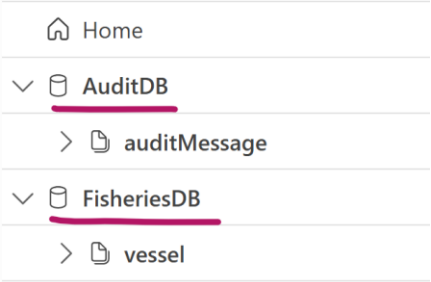
- ✓ Publish Logic App endpoint for the external party(s)
- ✓ Create API Product and configure
- ✓ Publish Developer Portal
- ✓ Use Developer portal to find Logic App Façade API
- ✓ Create API Product(s) and Subscription(s) to manage access
- ✓ Create and configure API policies using template and manually with the help from Azure Copilot

Exercise 7 – (Optional) Implement Additional Requirement for a Full Audit (40 min)

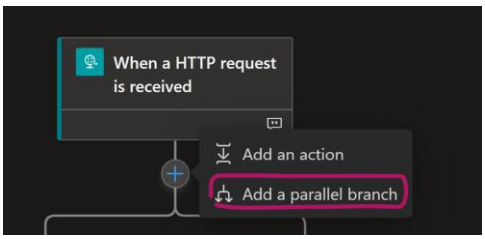
Topics	In this exercise, we will cover the following topics. • Create a new database in CosmosDB and configure access • Validate Audit Functionality
Duration	• 40min
Tool(s)	• Azure portal
Subscription	[selected subscription]
Resource Group	[selected RG]
Scenario	You have established enterprise message integration in Azure successfully. Now other business teams/groups are starting to see value in using your integration platform. The security team requests that all messaging coming to the organisation must be audited (without changing or updating) as it is (in original format). They want to establish their own audit database to store any and all messages for the organisation. Your job is to implement their requirement without creating a huge impact to the existing functionality and requirements

Module 1 – Create New Audit Database in CosmosDB and Configure Access

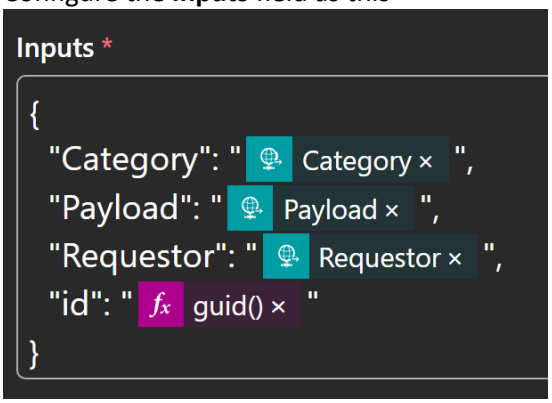
1. Since messages can be in any format containing all sort of datasets, you don't want to be fixed with the database schema. Therefore, you choose CosmosDB to store audit data/messages.
   We will implement some logic/workflow in Logic App Façade and save the original data in CosmosDB
2. Create a new *CosmosDB database* called **"AuditDB"**, a new *container (document table)* called **"auditMessage"**. Follow the instructions like you previous did for *Exercise 2 Module 1*
3. In your CosmosDB, you have 2x databases



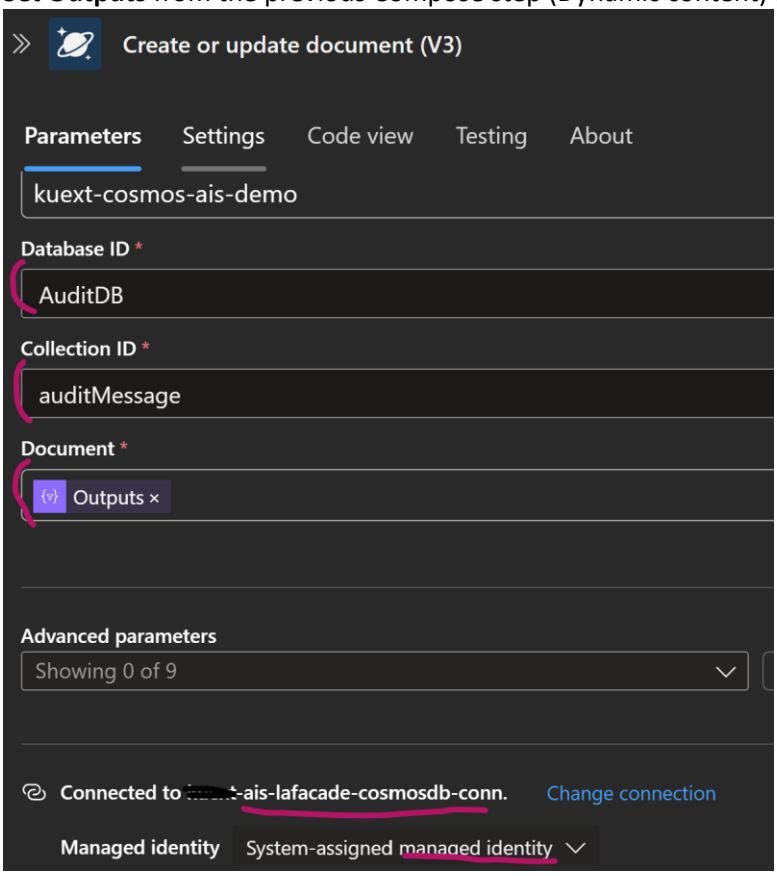
4. You have already configured the managed identity access for Logic App Façade in CosmosDB. For this exercise, we will reuse the existing access and permissions
5. Select your *Logic App Façade* and open the *Design Editor*
6. **Add a parallel branch** just below the trigger, *When a HTTP request is received*



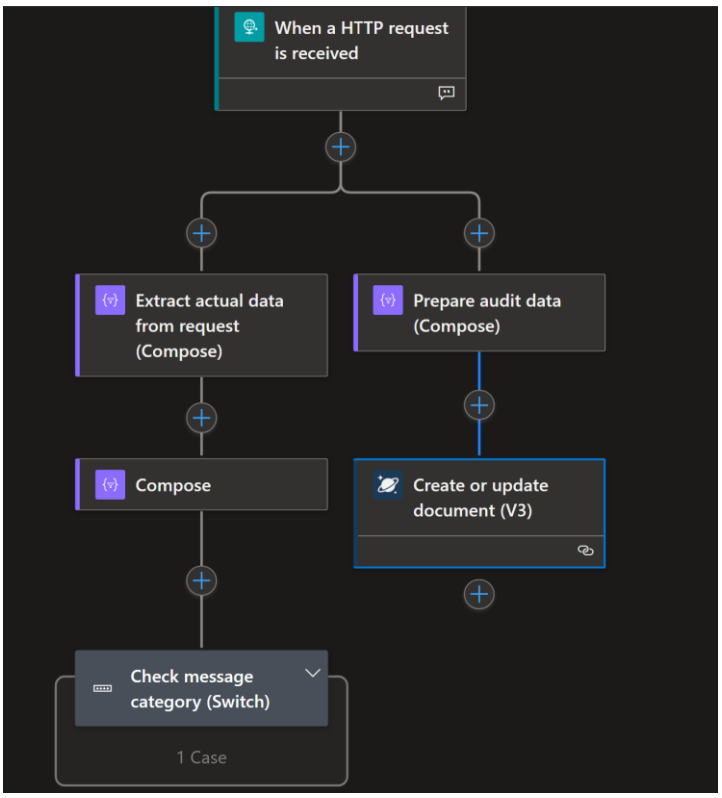
7. Add **Compose** action and rename it to ***"Prepare audit data (Compose)"***
8. Configure the **Inputs** field as this



9. Add **Create or update document (V3)** action from CosmosDB connector
10. Create a **new connection** using *managed identity* to *AuditDB* in CosmosDB
11. Set **Outputs** from the previous Compose step (Dynamic content) for the *Document* field



12. The overall steps for Logic App Façade looks like this



13. Save the change

Module 2 – Validate Audit Functionality

1. Validate whether the audit functionality works as expected. Send test data from APIM
2. Validate the run history for Logic App Façade
3. Validate if the data is saved in AuditDB

Summary

ACHIEVEMENTS

After you have completed the Lab, you are now able to:

- ✓ Expand the existing functionality and add further requirements without creating huge impact
- ✓ Validate all that you have learnt today
- ✓ Implement parallel actions/steps