Microsoft Cloud Workshop

App modernization / Serverless Apps

Hands-on lab

October 2019

Contents

App modernization / Serverless App hands-on lab

Abstract and learning objectives

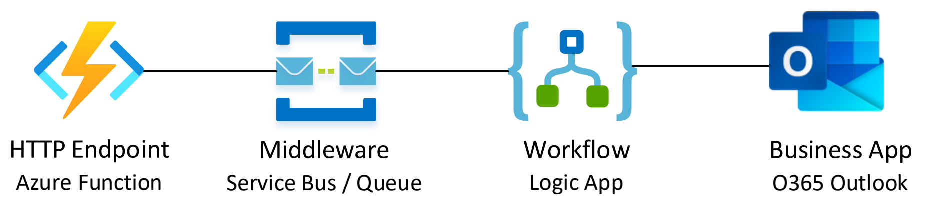
Explore the options for serverless architectures.

Learning Objectives:

- Implement a simple HTTP Endpoint using Azure Functions
- Utilize Azure Service Bus as messaging middleware
- Explore the possibilities of Azure Logic Apps to integrate with 3rd party applications

Solution architecture

After lawyers affirmed that Contoso Ltd. could legally process and store customer data in the cloud, Contoso created a strategy that capitalized on the capabilities of Microsoft Azure.



The existing solution is a Web Service accepting update notifications on new cases. Relying parties have been reporting issues with the performance of the service. Investigation exposed the synchronous integration pattern was not a perfect solution for the given amount of connected services.

The new solution should leverage serverless application architecture as much as possible. The API should accept the message and initiate the processing steps asynchronously. It must also be able to adjust to higher load of notifications.

API endpoint will be implemented as Azure Function. Messages requiring further processing will be put in a queue from where they will be picked by a Logic App, which in turn will handle the communication with 3rd party applications.

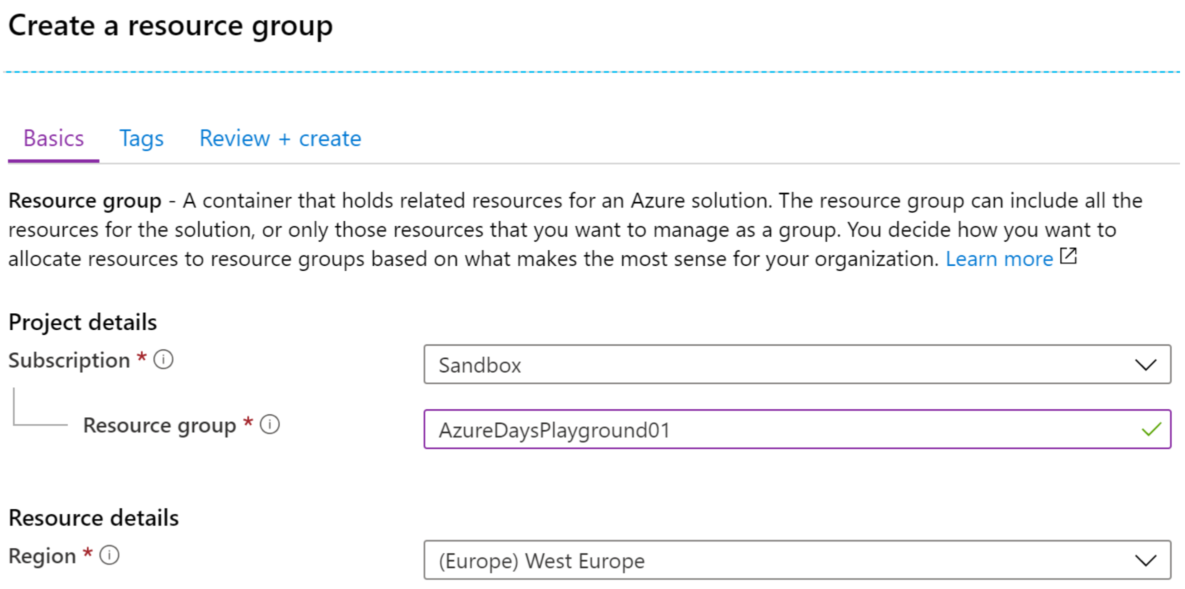
Requirements

To complete this lab, you will only need a browser (Microsoft Edge or Google Chrome recommended)

Before the hands-on lab

Duration: 5 minutes

Please make sure that you have access to the Azure Portal and either have permissions to create new Resource Groups or you have been provided a dedicated Resource Group where you can create Azure Assets.



Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Learn more

Project details

Subscription *	Sandbox
Resource group *	AzureDaysPlayground01

Resource details

Region *	(Europe) West Europe

Exercise 1: Provision and execute a Function App

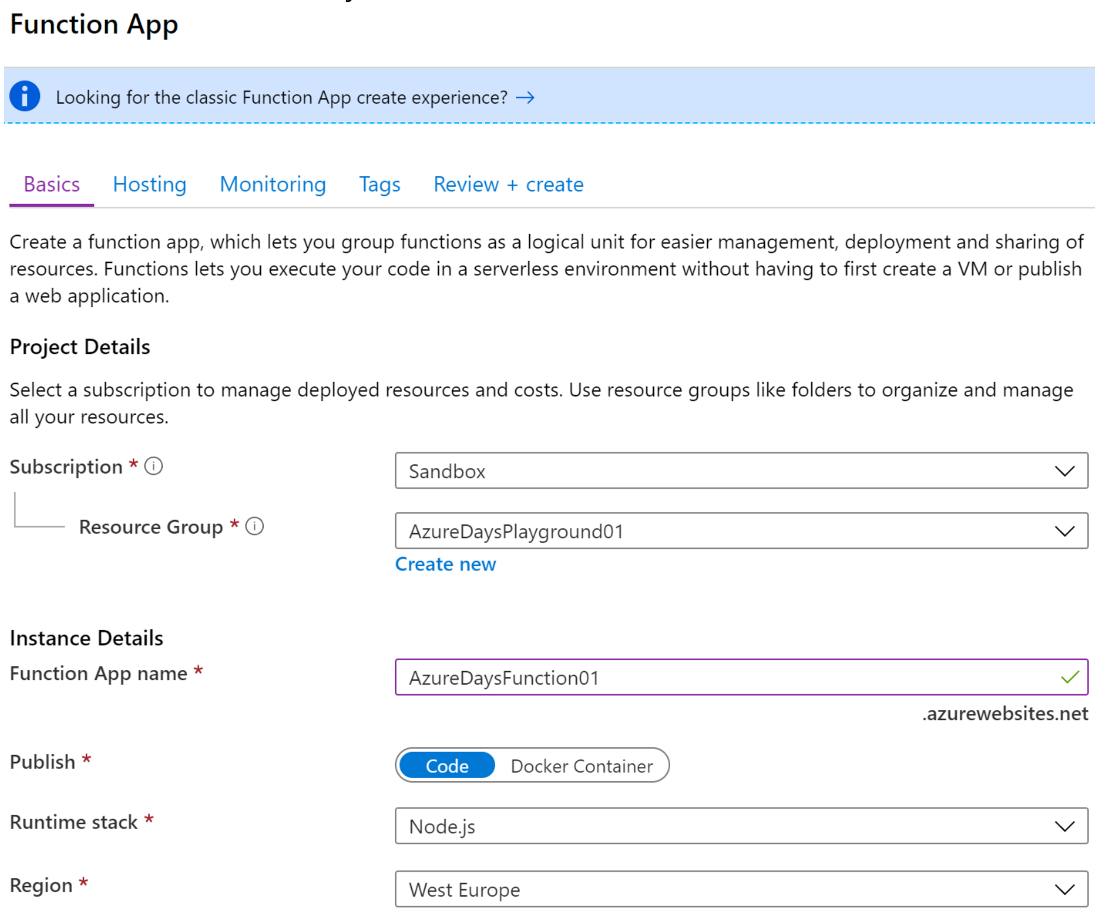
Duration: 20 minutes

The company has asked you to provision a simple API to process incoming events. As a Node.js developer, you prefer to keep using this stack for development.

Task 1: Create a new Function App

Tasks to complete

1. Locate the **Function App** template in the available Azure Resources and click on **Create**.
2. Provide basic information for the new Function App. Make sure you selected the Resource Group you plan to use for this exercise. Select Node.js as Runtime stack.



Function App

Looking for the classic Function App create experience? →

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

Project Details

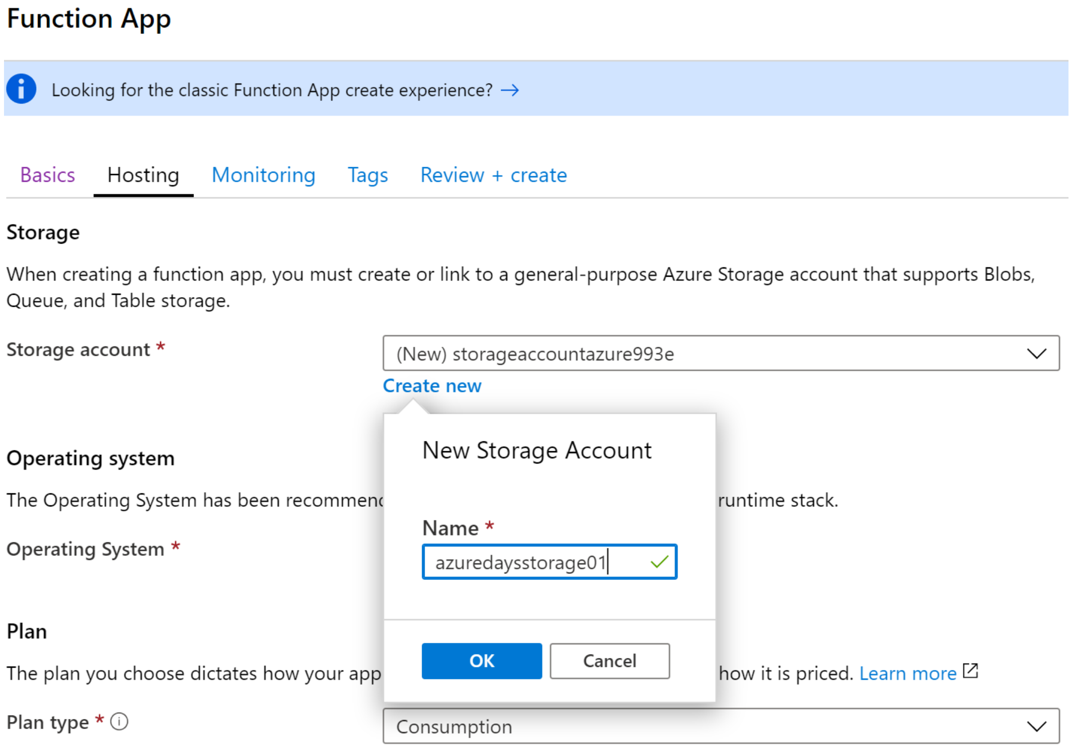
Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Sandbox
Resource Group *	AzureDaysPlayground01

Instance Details

Function App name *	AzureDaysFunction01.azurewebsites.net
Publish *	[✓ Code] Docker Container
Runtime stack *	Node.js
Region *	West Europe

3. In the hosting tab you can adjust its the name of the storage account that will be created automatically.



Function App

Looking for the classic Function App create experience? →

Basics | Hosting | Monitoring | Tags | Review + create

Storage

When creating a function app, you must create or link to a general-purpose Azure Storage account that supports Blobs, Queue, and Table storage.

Storage account * (New) storageaccountazure993e

Create new

New Storage Account

Name * azuredaysstorage01

OK Cancel

Operating system

The Operating System has been recommend runtime stack.

Operating System *

Plan

The plan you choose dictates how your app how it is priced. Learn more

Plan type * Consumption

4. Leave the remaining settings unchanged, review the settings and start creation of the Resource. Wait until the Function App is successfully provisioned.

Exit criteria

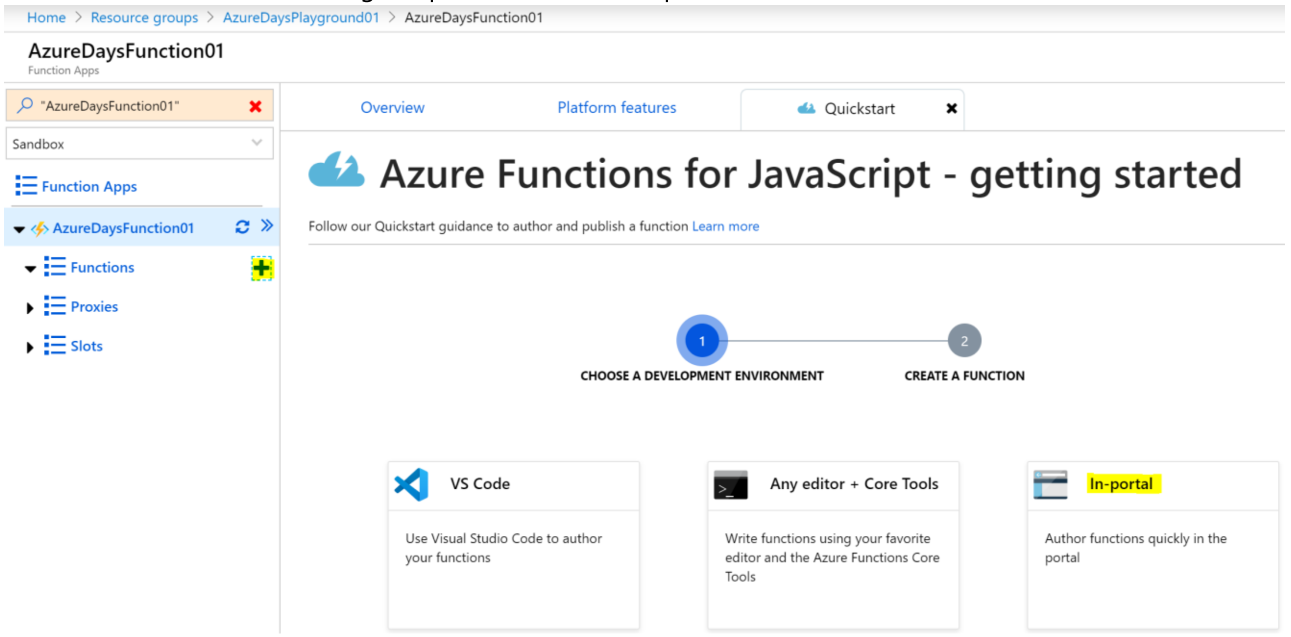
- Function App is successfully provisioned in the dedicated Resource Group

Task 2: Create and a new Function

Tasks to complete

1. Locate and open the new Function App in the dedicated Resource Group.

2. From here you can create a new Function by clicking on the "+" Button next to Functions menu entry. In this exercise, we will be using the portal as our development environment.



Azure Functions for JavaScript - getting started

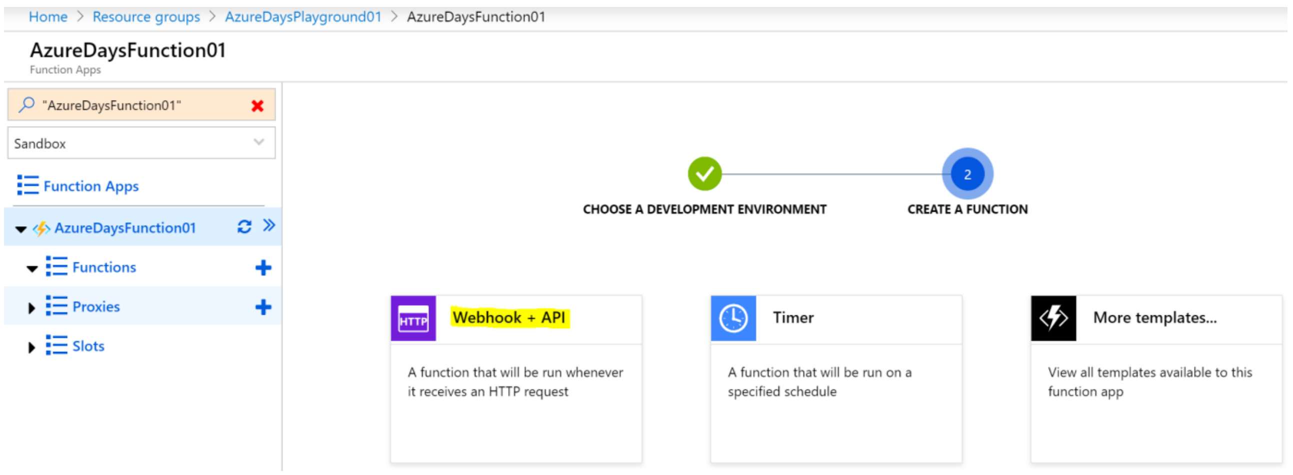
Follow our Quickstart guidance to author and publish a function Learn more

VS Code — Use Visual Studio Code to author your functions

Any editor + Core Tools — Write functions using your favorite editor and the Azure Functions Core Tools

In-portal — Author functions quickly in the portal

3. We will be using an HTTP(s) endpoint as a trigger for our function. Feel free to explore other options for triggering Functions.



Webhook + API — A function that will be run whenever it receives an HTTP request

Timer — A function that will be run on a specified schedule

More templates... — View all templates available to this function app

4. The newly created function exposes some basic functionality. Explore the source code of the app (index.js) and identify how the HTTP input is being processed.
5. Execute the Function using the portal testing facilities, analyze the output.
6. Retrieve the Function URL and call it directly in your browser.
7. Inspect the call history in the Monitor section of the function.

Exit criteria

- Webhook / API Function is successfully created
- Function can be successfully executed with different values of parameter in the portal and by calling a URL in browser
- Run history is recorded and visible in the Monitoring section

Exercise 2: Deploy and utilize messaging middleware

Duration: 20 minutes

Some messages received via the newly created API will need to be processed asynchronously. You need to provide a way to enable deferred processing of such messages.

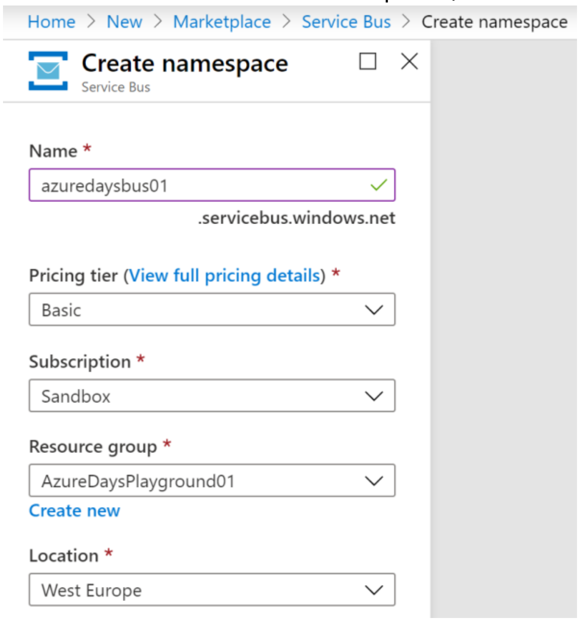
Help references

Azure Functions Service Bus Bindings	https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-service-bus

Task 1: Create a message queue

Tasks to complete

1. Create a new Service Bus Namespace (Basic Pricing Tear is sufficient for the purpose of the given exercise)



Name * azuredaysbus01 .servicebus.windows.net

Pricing tier (View full pricing details) * Basic

Subscription * Sandbox

Resource group * AzureDaysPlayground01

Create new

Location * West Europe

2. In the newly created namespace, create a new Queue named **outqueue** with standard settings.

Exit criteria

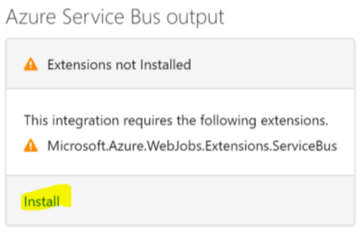
- Service Bus Namespace and Queue are provisioned and visible in the dedicated Resource Group.

Task 2: Integrate Service Bus Queue with Function

Tasks to complete

1. Return to the Function App in Azure Portal, select the Http Trigger Function and click on **Integrate**. Explore the 3 different binding types: Trigger, Input and Output.

2. Add a new Output of type **Azure Service Bus**. Follow the suggestion to install the required extension – it may take a few minutes to complete.

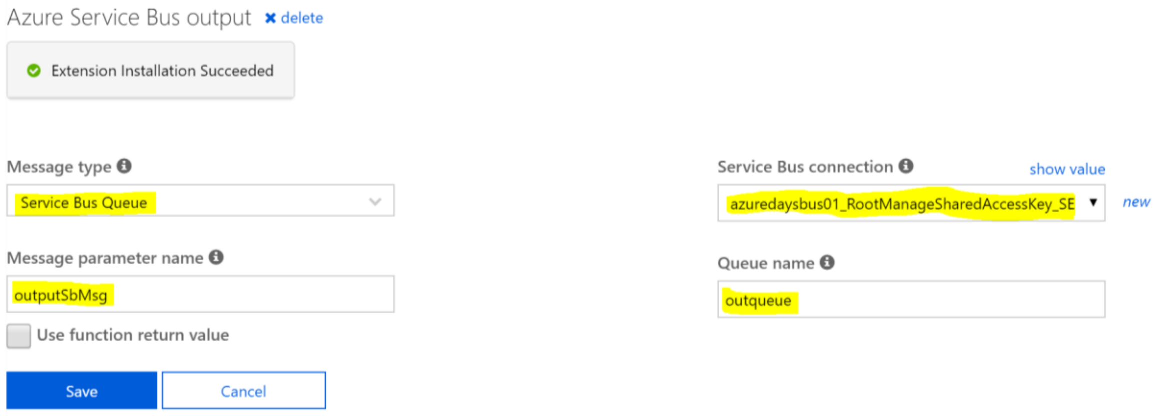


Azure Service Bus output

Extensions not Installed

This integration requires the following extensions.
Microsoft.Azure.WebJobs.Extensions.ServiceBus

3. Verify the settings of the binding, ensure that the **Message type** is set to "**Service Bus Queue**".



Azure Service Bus output

Extension Installation Succeeded

Message type: Service Bus Queue

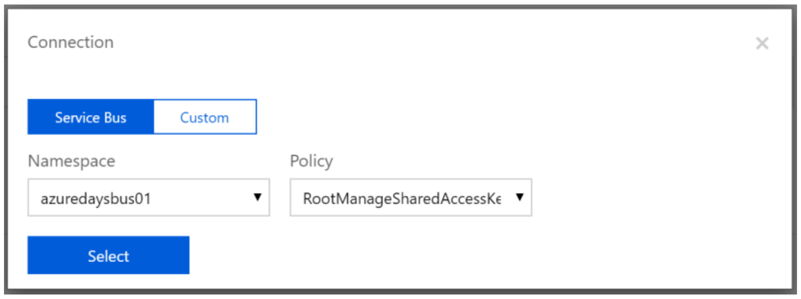
Service Bus connection: azuredaysbus01_RootManageSharedAccessKey_SE

Message parameter name: outputSbMsg

Queue name: outqueue

Use function return value

For Service bus connection, click on "new" and select the service bus namespace you provisioned in the previous task.

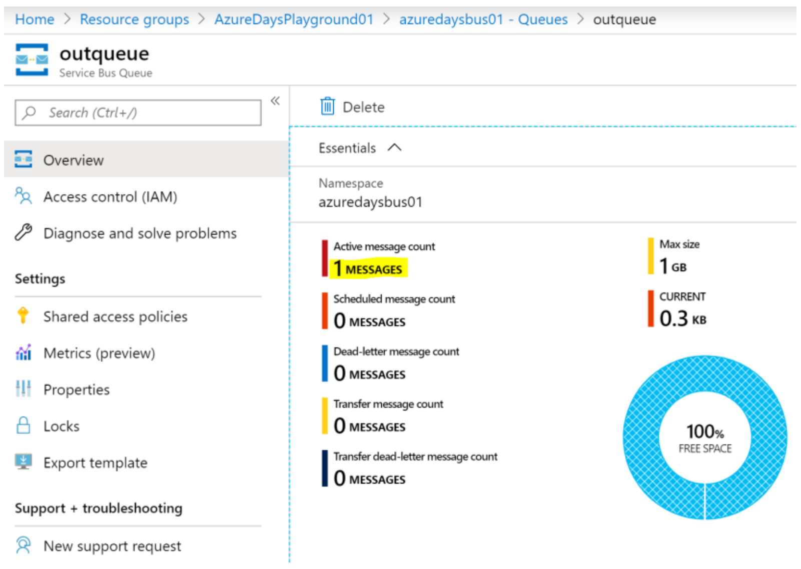


Connection

Namespace: azuredaysbus01

Policy: RootManageSharedAccessKe

4. Save the binding settings.
5. The new binding is exposed via `context.bindings.outputSbMsg`. Assigning a text value to the property will result in sending a message to the configured queue after the function is executed. Add a line such as
`context.bindings.outputSbMsg = (req.query.name || req.body.name) + " was here";`
to the function body. Please refer to the official documentation for details.
6. Execute the function either via portal test or by calling the function-related URL in the browser.
7. Switch to the service bus queue in the portal. A dashboard displayed in the queue overview should indicate that the queue contains active messages.



Namespace
azuredaysbus01

Active message count
1 MESSAGES

Scheduled message count
0 MESSAGES

Dead-letter message count
0 MESSAGES

Transfer message count
0 MESSAGES

Transfer dead-letter message count
0 MESSAGES

Max size
1 GB

CURRENT
0.3 KB

Exit criteria

- Service Bus binding is configured for the new function, dependencies are successfully installed.
- Function execution sends messages to the queue.

Exercise 3: Process messages using a Logic App

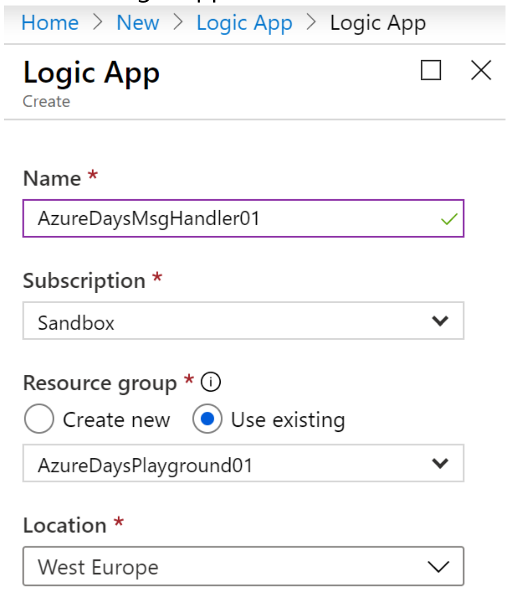
Duration: 20 minutes

Azure Logic Apps' wide range of supported connectors make it a perfect candidate for scenarios where integration with various Line of Business and Productivity applications is required. In this exercise we will trigger a Logic App to send an Email directly to O365 Outlook Mailbox.

Task 1: Create a message handler Logic App

Tasks to complete

1. Create a Logic App in the Azure Portal



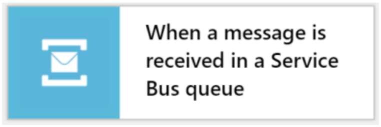
Name * AzureDaysMsgHandler01

Subscription * Sandbox

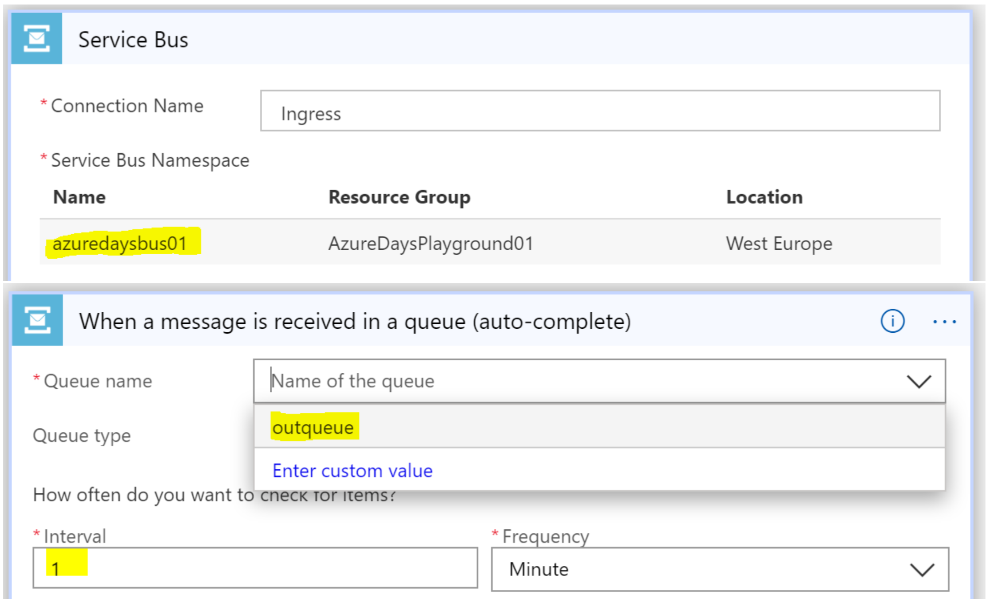
Resource group * Create new [✓ Use existing] AzureDaysPlayground01

Location * West Europe

2. Select a Service Bus queue as a trigger



3. In the new Logic App, configure the service bus connection by selecting the Service Bus namespace and the Queue name. Also set the polling interval to 1 Minute.



Service Bus

*Connection Name: Ingress

*Service Bus Namespace

Name	Resource Group	Location
azuredaysbus01	AzureDaysPlayground01	West Europe

When a message is received in a queue (auto-complete)

*Queue name: Name of the queue

outqueue

Enter custom value

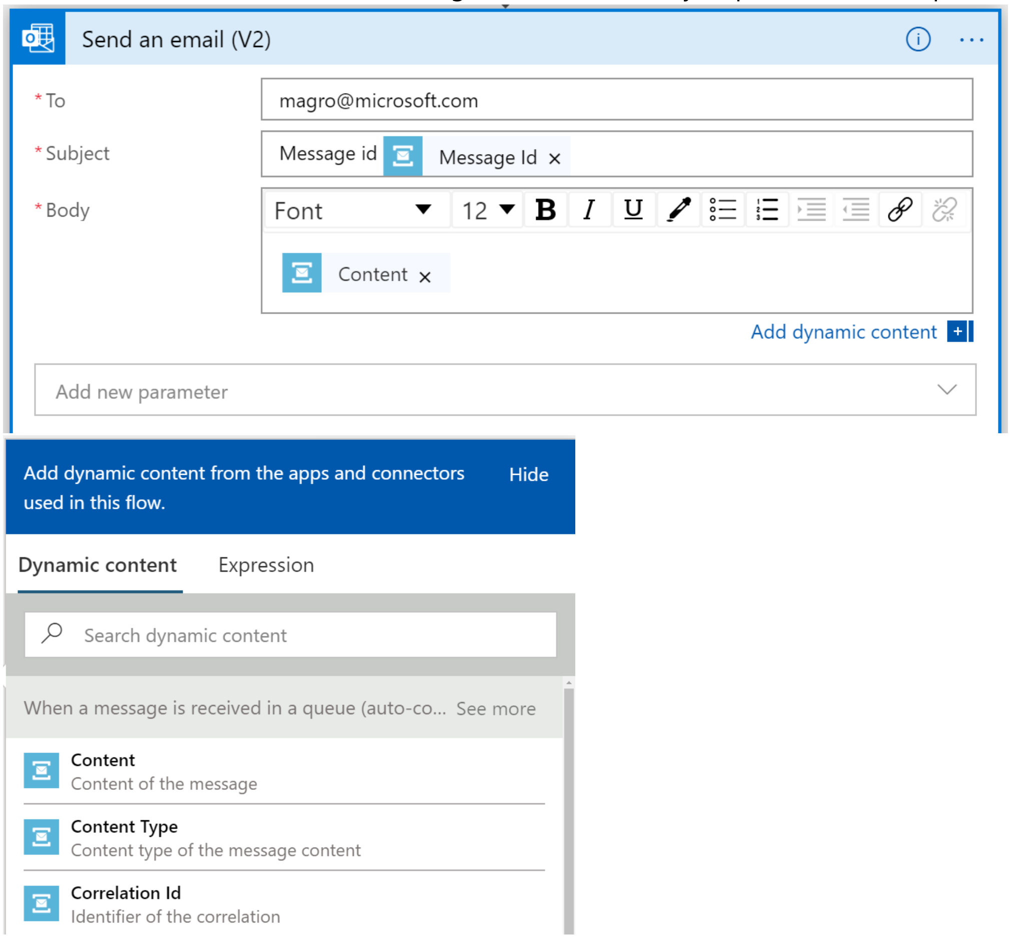
Queue type

How often do you want to check for items?

*Interval: 1

*Frequency: Minute

4. Add a new action of type Send an Email (V2). Configure a personal connection to O365 Outlook. Explore how the output blocks of the previous steps is made available in the subsequent steps of the workflow via dynamic content: add the content of the message to the email body, experiment with expressions, there's a lot to discover.



Send an email (V2)

*To: magro@microsoft.com

*Subject: Message id Message Id

*Body: Content

Font
12
Add dynamic content
Add new parameter
Add dynamic content from the apps and connectors used in this flow.
Hide
Dynamic content
Expression
Search dynamic content
When a message is received in a queue (auto-co... See more

Content
Content of the message

Content Type
Content type of the message content

Correlation Id
Identifier of the correlation

5. After saving the workflow it automatically starts listening to the trigger events and runs when messages arrive in the queue.

Exit criteria

- Message handler Logic App listening to the Service Bus queue is created and deployed.

Task 2: Test the application

Your first application spanning a lightweight HTTP Handler, a messaging middleware and a message processor is now complete and ready to be tested.

Tasks to complete

1. Submit a message to the Function URL.
2. Open the Monitor tab of your Function and inspect the record.
3. Wait until the Logic App is executed
4. Explore the run history of the Logic App, identify the most recent run. Inspect the inputs and outputs of the actions.

Exit criteria

- Your application is performing as expected
- You can identify the building blocks and understand how they work in combination

Exercise 3 (optional): Build your own app

Duration: 30 minutes

Feel free to suggest your own scenario and try to implement it. Explore additional opportunities for integration.