SERVERLESS360 by Kovai.co

Whitepaper

Building integration solutions with Microsoft Azure Logic Apps Standard

Microsoft Azure

Steef-Jan Wiggers
Microsoft Azure MVP
Technical Integration Architect, HSO

Contents

1 About the Author

2 The Journey to Logic Apps Standard

3 Runtime change

4 The new version of Logic Apps

5 Running Logic Apps in a Container

6 Monitoring Logic Apps with Application Insights

7 Networking and Security

8 Considerations

9 Summary

Logic Apps are Microsoft's Integration Platform as a Service offering in Azure, allowing you to build cloud-native, hybrid, and now even on-premise workflows and integrations. With the upcoming Logic Apps release, the service will have a new designer, hosting option in containers, and the ability to run inside the App Service, bringing a completely new experience for its users. This paper will dive into the latest version of Logic Apps, which was in preview for the last couple of months and is now generally available.

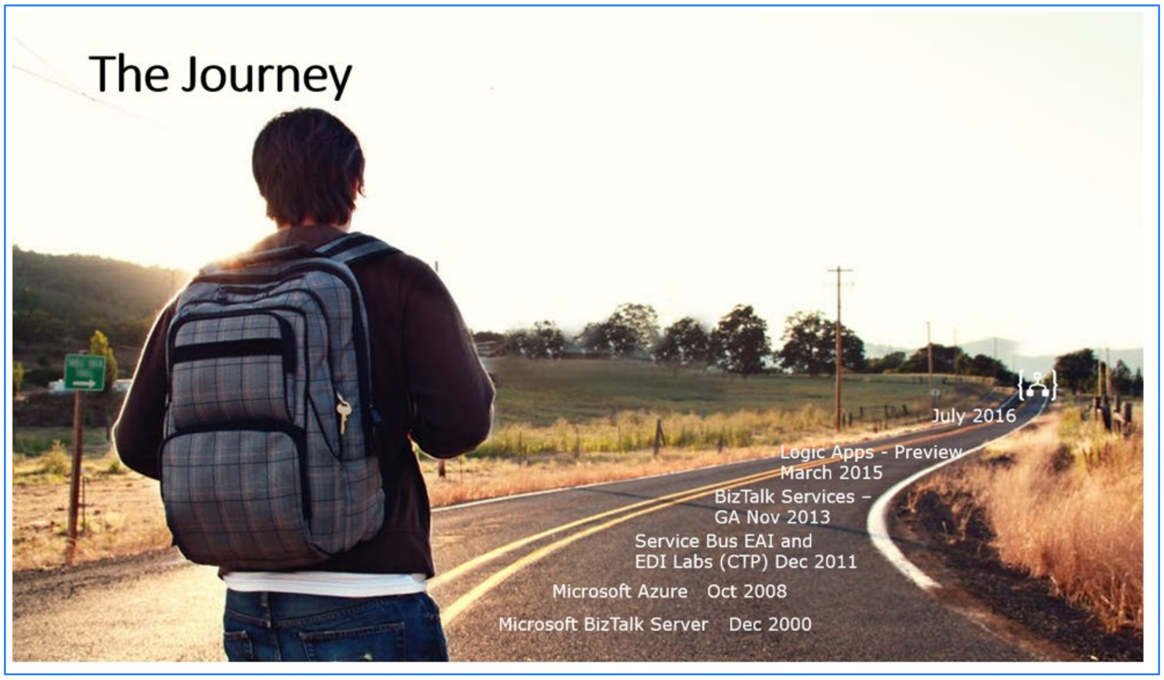
1 About the Author

Steef-Jan Wiggers works in the Netherlands as a Technical Integration Architect at HSO. His current technical expertise focuses on integration platform implementations, Azure DevOps, and Cloud Solution Architectures. Steef-Jan is a board member of the Dutch Azure User Group (WAZUG NL), a regular speaker at conferences and user groups, and he writes for InfoQ and Serverless Notes. Furthermore, Microsoft has recognized him as Microsoft Azure MVP for the past eleven years.

2

The Journey to Logic Apps Standard

Microsoft released BizTalk Server over 20 years ago, providing enterprises a message broker product to facilitate system integration. The product evolved to its recent 11th release. During that period, the company revealed Azure in 2008, which made general availability two years later. It marked the starting point to developing similar capabilities of BizTalk Server in the Cloud. The first is the service bus providing a similar pub/sub mechanism at the core of BizTalk. Next to the messaging capabilities, Microsoft also made attempts to have the orchestration function and myriad adapters available in a cloud offering.



The first attempt was known as EA CTP, which was succeeded by the second iteration of a service called 'BizTalk Services.' After its availability, the company redesigned the architecture of the service into Logic Apps – available as a preview in 2005 and made generally available in July 2006. Ever since the adoption grew steadily and with the continuous investment and improvements by the product team, the service became a leader in the Gartner Quadrant of integration Platform as a Service (iPaaS) services next to Dell Boomi, MuleSoft, and other providers. A fourth iteration recently made generally available (GA) allows you to run Logic Apps on the App Service runtime – just like Azure Functions. More hosting options will become available, including the containerization of Logic Apps.

With the new version of Logic Apps, developers can expect the following:

- Running Logic Apps stateless and stateful
- Ability to containerize Logic Apps
- New observability option with Application Insights
- A different pricing model for Logic Apps

The options to host and leverage Logic Apps increase with the new version. You ultimately will have the opportunity to run Logic Apps in a multi-tenant mode – that is, through consumption (an option available today) or run in a dedicated isolated manner through an Integrated Service Environment (ISE) or stand-alone (single-tenant), which is the version in preview. Single-tenant means, according to the Microsoft documentation:

"The workflows in the same logic app and a single-tenant share the same processing (compute), storage, network, and so on."

3

Runtime change

A significant difference between the multi-tenant Logic App and isolation option ISE on the one hand and the single-tenant Logic App version on the other is the runtime.

Logic Apps is a job scheduler with a JSON-based Domain Specific Language (DSL) describing a dependency graph of actions – an inverse-directed graph. What Microsoft Pro-Integration Team responsible for this service have done is that a user might feel like you're doing step A, step B, then step C. However, what they have done is an inverse dependency on the previous step. Therefore, if you look at the code view, you will see a **RunAfter** step. And it says **RunAfter** step A, right? Step B runs after step A – thus then when you create your Logic app, you have the sequence of actions.

```
"actions": {
        "HTTP": {
           "type": "Http",
           "inputs": {
              "method": "GET",
              "uri": "https://openexchangerates.org/api/latest.json?app_id=7f565dfaeec748de8ad9b1
c794bcd079"
           },
           "runAfter": {}
        },
        "Transform_JSON_To_JSON": {
           "type": "Liquid",
           "kind": "JsonToJson",
           "inputs": {
              "content": "@body('HTTP')",
              "map": {
                 "name": "currency.liquid"
              }
           },
           "runAfter": {
              "HTTP": [
                 "Succeeded"
              ]
           }
        },
"Response": {
            "type": "Response",
            "kind": "http",
            "inputs": {
               "statusCode": 200,
               "body": "@outputs('Transform_JSON_To_JSON')?['body']"
            },
            "runAfter": {
               "Transform_JSON_To_JSON": [
                  "Succeeded"
               ]
            }
         }
      }
```

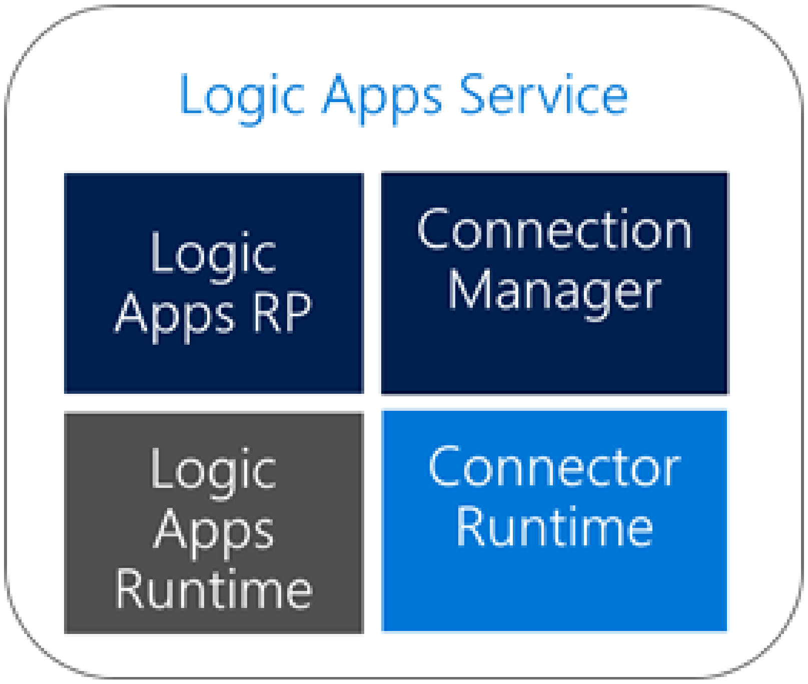
The JSON DSL allows you to have is – no dependencies and then have everything run in parallel. Moreover, every action is essentially a task that runs in the background – allowing to be a very highly parallelizable engine so that your workflows can run in parallel. To conclude, keep that in mind as you write your logic apps; it's a dependency graph, not a step one and a step two processes.

The DSL remains the same for any Logic App version; however, the runtimes executing are different. The multi-tenant Logic Apps service consists of four general components:

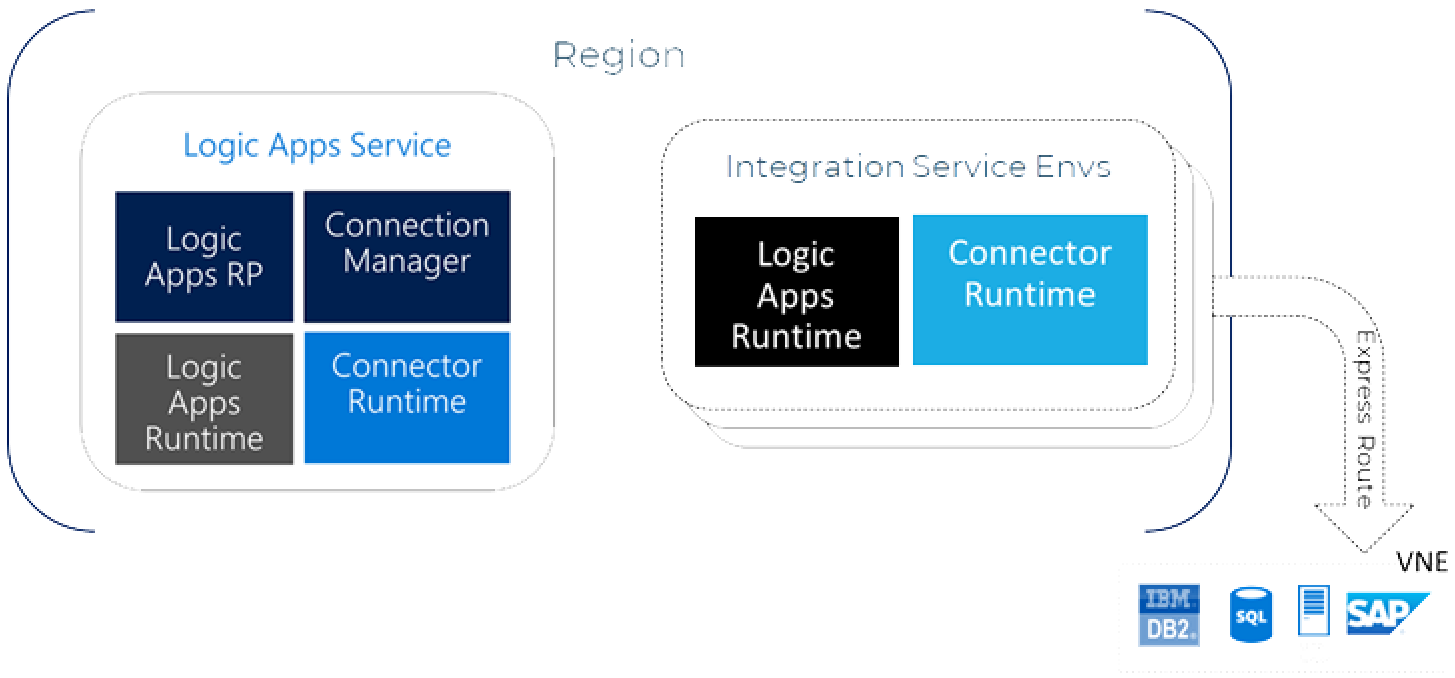
- A Logic Apps RP, a **Resource Provider** – essentially the Logic App frontend handling all the requests. This component reads the workflow definition, breaks it down into component tasks with dependencies, and then puts it into storage, which then the backend will process.
- The Logic Apps **Runtime** is a distributed compute that will go ahead and coordinate those tasks that have been broken down from your Logic App.

- Then we have a **Connection Manager** that goes forward and manages the connection configuration, token refreshment, and credentials that you have in your API connections.

- And then finally, the **Connector Runtime**, which hosts the API abstractions to all the APIs that you have and hosts– sometimes are codeful, sometimes they're codeless, and then manages that abstraction for you.



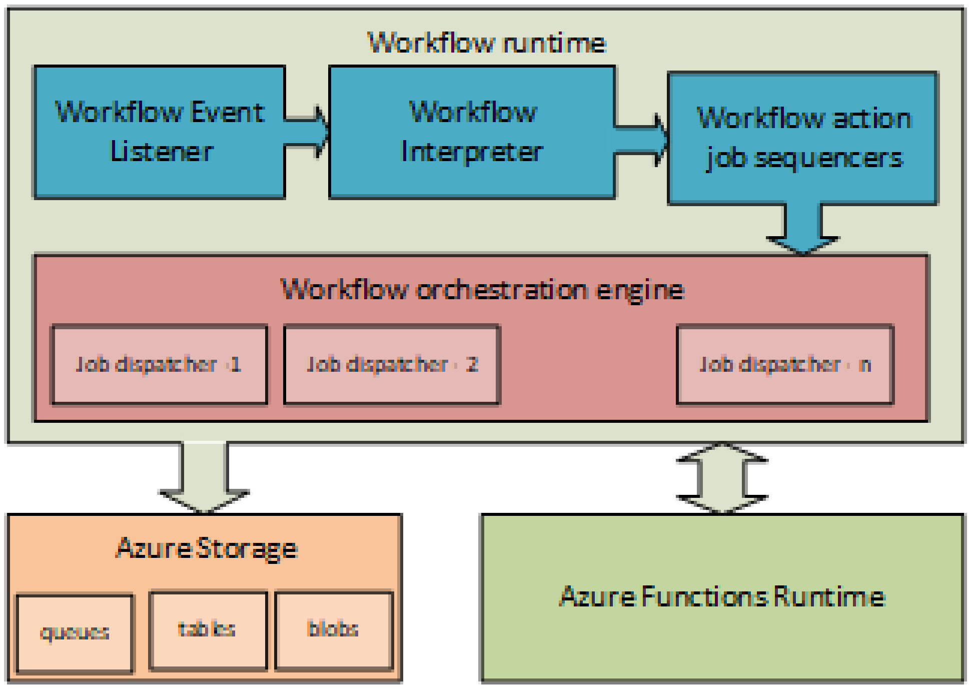
When you want to run Logic Apps in an isolated manner, you can choose to provision an ISE. In that case, you will have a dedicated Logic Apps- and Connector runtime. Furthermore, you can benefit from enterprise features such as VNET- and Express Route support.



There is a different runtime with the single-tenant version of Logic Apps, i.e., Logic Apps run in a Function App runtime. The Runtime deep dive blog post by Rohitha Hewawasam explains it as follows:

The Logic Apps runtime implements each action type in a workflow definition as a job run by the underlying Logic Apps job orchestration engine. For example, the HTTP action type is implemented as a job that evaluates the inputs to the HTTP action, sends the outbound request, handles the returned response, and records the result from the action.

Every workflow definition contains a sequence of actions mapped to a directed acyclic graph (DAG) of jobs with various complexity. When a workflow is triggered, the Logic Apps runtime looks up the workflow definition and generates the related jobs organized as a DAG, which Logic Apps calls a job-sequencer. Each workflow definition is compiled into a job sequencer that orchestrates the running of jobs for that workflow definition. The job orchestration engine is a horizontally scalable distributed service that can run DAGs with large numbers of jobs.



The runtime for the single-tenant Logic Apps allows running workflows stateful and stateless. Hewawasam explains:

> *For stateful workflows, the orchestration engine schedules the jobs in the sequencers by using Azure Storage queue messages. Behind the scenes, multiple dispatcher worker instances (job dispatchers), which can run on multiple compute nodes, monitor the Logic Apps job queues. By default, the orchestration engine is set up with a single message queue and a single partition on the configured storage account. However, the orchestration engine can also be configured with multiple job queues in multiple storage partitions. For stateless workflows, the orchestration engine keeps its states entirely in memory.*

4

The new version of Logic Apps

With the new version or fourth iteration of its integration Platform as a Service (iPaaS), Microsoft brings new features for its users, such as running Logic App on the app service runtime and including a new designer. Next to the new hosting option, it also offers:

- Better developer experience
- Streamlined DevOps
- Improved Performance
- Enhanced Security
- Optimized Monitoring
- Extensibility

We will start with the developer experience first before continuing with DevOps and non-functional aspects of performance, security, and monitoring aspects. And lastly, we will discuss some considerations with the new version of Logic Apps.

4.1

Better Developer Experience

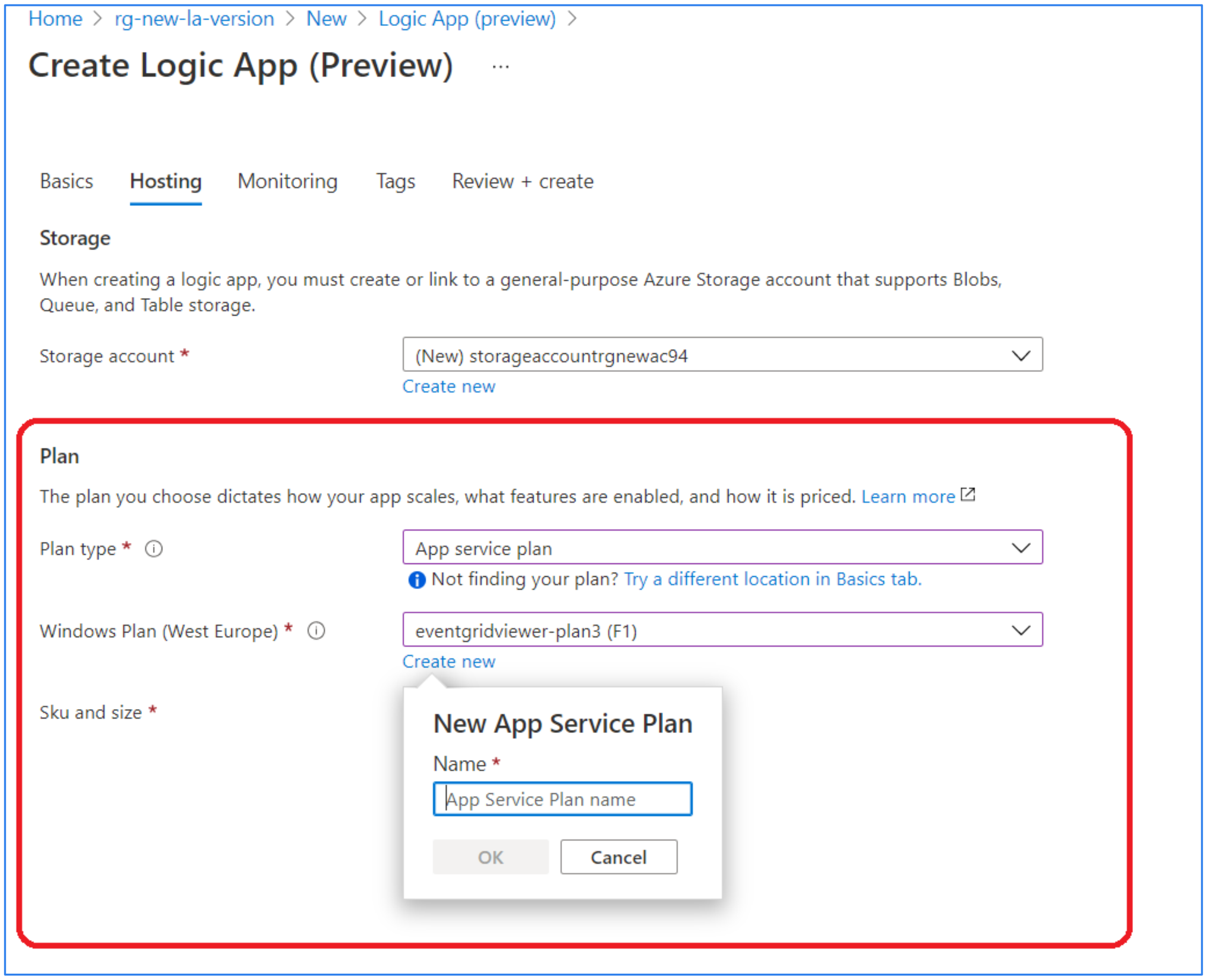
4.1.1

Create a Logic App Instance

When you go to the Azure Portal, you can access the Logic Apps in preview through the Marketplace. By choosing to create a Logic App in preview, you will see the **Basics** tab first. Within this tab, you can select the subscription and resource group for your Logic App, and next to name and region, whether you like to choose to publish the Logic App as a workflow or docker container. The publishing part is new.

The following tab details **hosting**, and here you configure (select) your storage account and app service plan. The hosting section is similar to the details you set when creating a Function App. Since the preview (new version) of Logic Apps is the stand-alone (single-tenant) option (now GA). You can select either the premium or App Service plan.

Premium App Service plan automatically scales based on demand using pre-warmed workers, which run applications with no delay after being idle, runs on more powerful instances, and connects to virtual networks according to the Microsoft hosting options documentation. The App Service Plan can be a Service Plan you created yourself earlier and selected in the tab or a new one.


Home > rg-new-la-version > New > Logic App (preview) >

Create Logic App (Preview) ...

Storage

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

Storage account * (New) storageaccountrgnewac94

Create new

Plan

The plan you choose dictates how your app scales, what features are enabled, and how it is priced. Learn more

Plan type * App service plan

Not finding your plan? Try a different location in Basics tab.

Windows Plan (West Europe) * eventgridviewer-plan3 (F1)

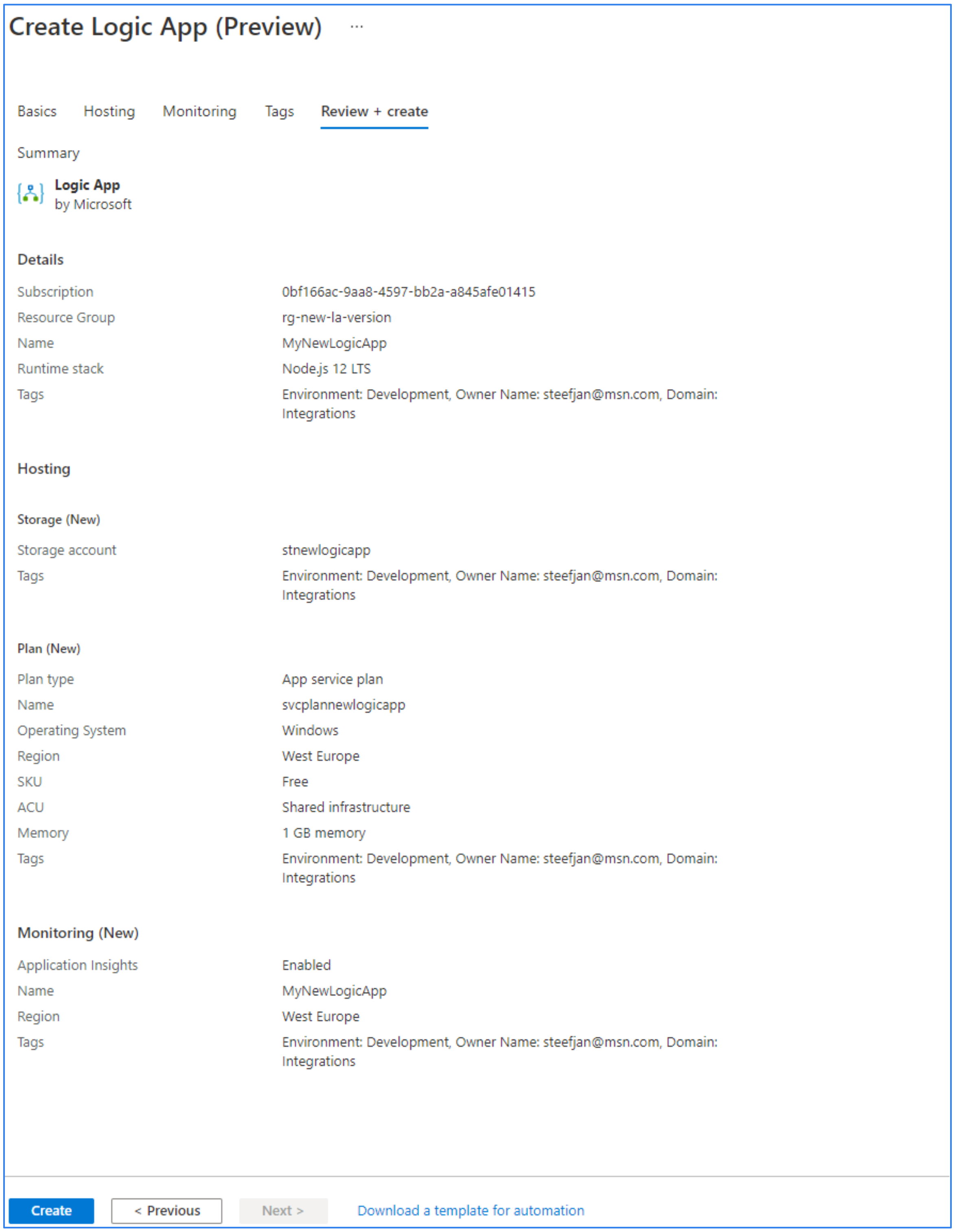
Create new

Sku and size *

New App Service Plan

Name * App Service Plan name

The following tab deals with **monitoring**, and similar to a Function App, you can choose to enable Application Insights and select an existing instance or create a new one. And finally, you can apply tags on the Logic App and review the configuration before create.



Create Logic App (Preview) ...

Basics | Hosting | Monitoring | Tags | **Review + create**

Summary

Logic App
by Microsoft

Details

Subscription	0bf166ac-9aa8-4597-bb2a-a845afe01415
Resource Group	rg-new-la-version
Name	MyNewLogicApp
Runtime stack	Node.js 12 LTS
Tags	Environment: Development, Owner Name: steefjan@msn.com, Domain: Integrations

Hosting

Storage (New)

Storage account	stnewlogicapp
Tags	Environment: Development, Owner Name: steefjan@msn.com, Domain: Integrations

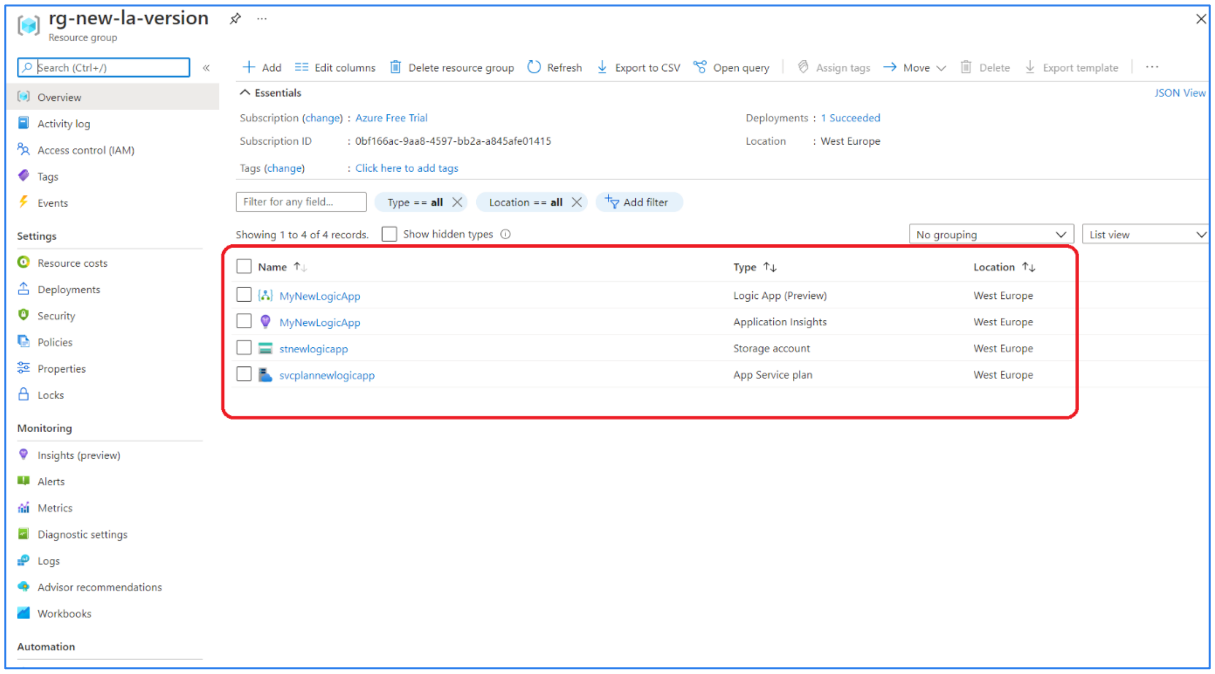
Plan (New)

Plan type	App service plan
Name	svcplannewlogicapp
Operating System	Windows
Region	West Europe
SKU	Free
ACU	Shared infrastructure
Memory	1 GB memory
Tags	Environment: Development, Owner Name: steefjan@msn.com, Domain: Integrations

Monitoring (New)

Application Insights	Enabled
Name	MyNewLogicApp
Region	West Europe
Tags	Environment: Development, Owner Name: steefjan@msn.com, Domain: Integrations

When you click to create the Logic App, its dependencies listed in the screenshot above will be provisioned. The Storage account supports the logging and monitoring capabilities of the App Service Runtime, while the plan deals with hosting.


rg-new-la-version
Resource group

Subscription (change) : Azure Free Trial
Subscription ID : 0bf166ac-9aa8-4597-bb2a-a845afe01415
Tags (change) : Click here to add tags
Deployments : 1 Succeeded
Location : West Europe

Showing 1 to 4 of 4 records.

Name	Type	Location
MyNewLogicApp	Logic App (Preview)	West Europe
MyNewLogicApp	Application Insights	West Europe
stnewlogicapp	Storage account	West Europe
svcplannewlogicapp	App Service plan	West Europe

4.1.2

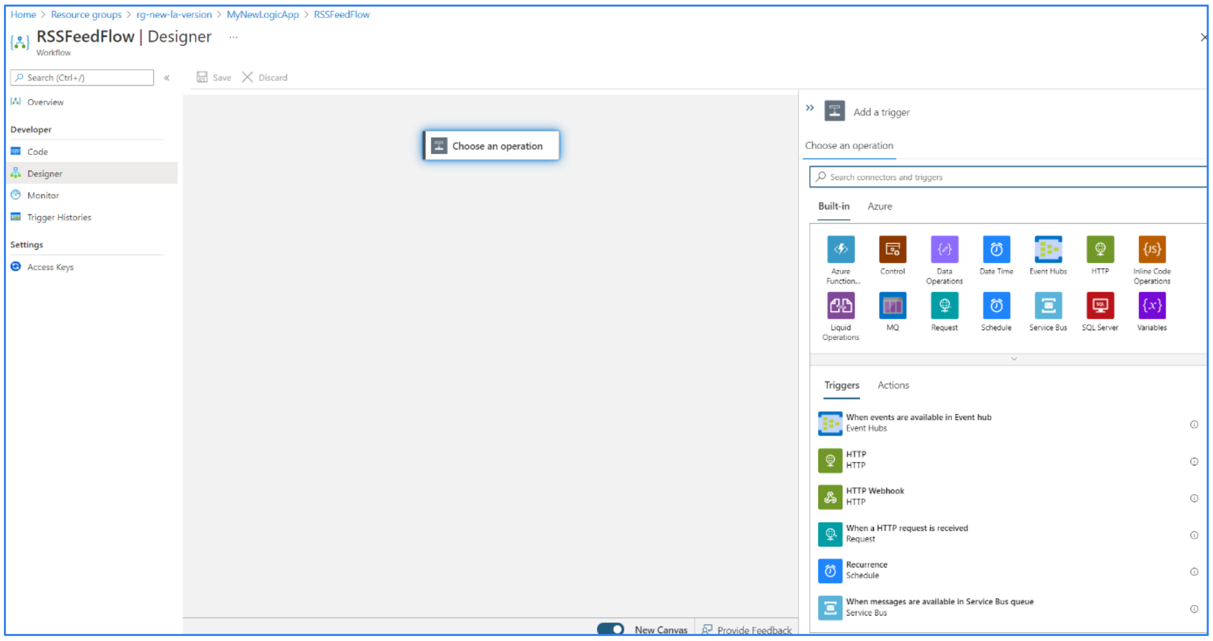
Create a workflow

By selecting your Logic App after the provisioning is complete, you can see in the overview and all the available tabs (left-hand side) that your Logic App instance is pretty familiar with a Function App. You will see settings, app service plans, deployment tools, and other features you would also find in the Function App. What you will notice also is workflows, and by selecting it, you can add one.

Clicking adds (+), you will see a new pane appear on the right-hand side with the following details:

- Workflow name
- State type: **Stateful** or **stateless**.

When you provide a name and choose, for instance, stateful, a new workflow template will be created. I am explicitly stating the template because you will see a developer tab when you select the workflow, where you can choose designer, providing you a canvas to drop the trigger and actions. Furthermore, the code view is the same as the code behind in the current version of Logic Apps.

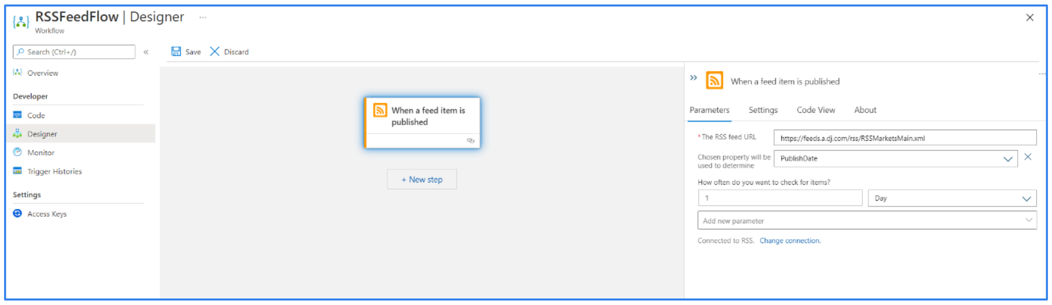


4.1.3

Build a workflow

You can create a workflow yourself when reading the earlier text of this paper to provision a Logic App instance. By adding a workflow, providing a name, and choosing the state type, you will end up with an empty template as described in the previous paragraph.

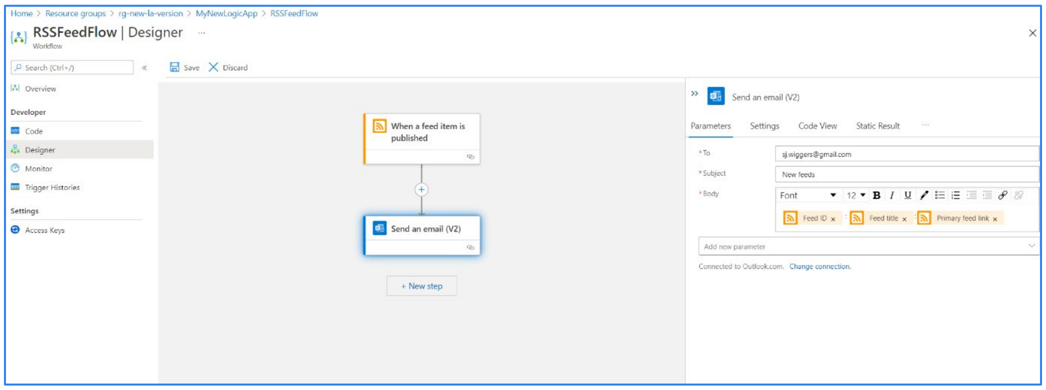
For example, as a trigger, you can double click the RSS connector (you can find it under Azure – not built-in!), and it will appear on top (over the choose an operation). You will see on the right-hand side a pane with the specifics for the RSS Feed connector. You can rename it and specify the RSS Feed URL – you can select https://feeds.a.dj.com/rss/RSSMarketsMain.xml - or another RSS Feed URL you prefer. Subsequently, you can choose whether you like when there's an update or publication date (options for the feed) and the interval.



When a feed item is published

* The RSS feed URL: https://feeds.a.dj.com/rss/RSSMarketsMain.xml
Chosen property will be used to determine: PublishDate
How often do you want to check for items? 1 Day
Connected to RSS.

Note that you can also look at the settings of the trigger action, code, and about. Next, you can click the next step to add an action. For instance, under Azure, you can look for the Outlook.com connector and choose the send mail (V2) operation. A new pane will appear, asking you to login into your Outlook.com account. Once you usefully logged in, a connection is created (similar to the current Logic App version). You can again, like with the trigger, specify specific settings – for the email you can set to, title, content, and so on:



Send an email (V2)

* To: sj.wiggers@gmail.com
* Subject: New feeds
* Body: Feed ID : Feed title : Primary feed link
Connected to Outlook.com.

Finally, you can save the workflow, and based upon the configuration of the feed, you will see invocations of the workflow in the trigger history. Furthermore, based upon the email address set to receive the feeds, it will show one or more emails in the belonging account. Alternatively, you can choose a compose action below the change feed if you do not want to send emails.

The steps above provide a way to create a Logic App workflow in Azure and get familiar with the new designer and developer tools' behaviour. Note that authoring a workflow can also be done using IDE Visual Code. In the following paragraphs, we will also touch upon that.

4.1.4

Running Logic Apps Stateless and Stateful

As mentioned earlier, workflows can be stateless or stateful when you want to create one. According to the Microsoft documentation:

> *Stateless means when you **don't need to save, review, or reference data** from previous events in external storage for later review. These workflows save the inputs and outputs for each action and their states only in memory, rather than transferring it to external storage.*

For that reason nothing has to be stored in the storage account – which is one of the dependencies of the Logic App, as you have seen in the previous paragraph. The runs are significantly shorter than stateful workflows and typically run no longer than five minutes (the same threshold as for Functions running in a consumption plan), will have faster performance (of course, since it runs in memory and no state is being written), and have reduced running costs. Another aspect of stateless also mentioned in the documentation is that when a run is interrupted, the caller of the workflow will need to resubmit that run manually – which means stateless runs are synchronous.

In the previous section, we discussed building a workflow leveraging an RSS feed that can run stateless or stateful. Choosing between these types depends on your use case and requirements. The trade-off depends on whether the Logic App (process) is something that does not save or reference information about previous operations (retrieve or persist state) or does need some specific information (state retrieved from, for instance, a database).

Until now, we showed and discussed using the Azure Portal for building a Logic App. You can also follow the Microsoft documentation (how-to guide) for building Logic apps (Standard) in the Portal. However, we recommend creating a Logic App (Standard) using VS Code, which we will discuss in the upcoming paragraph.

4.1.5

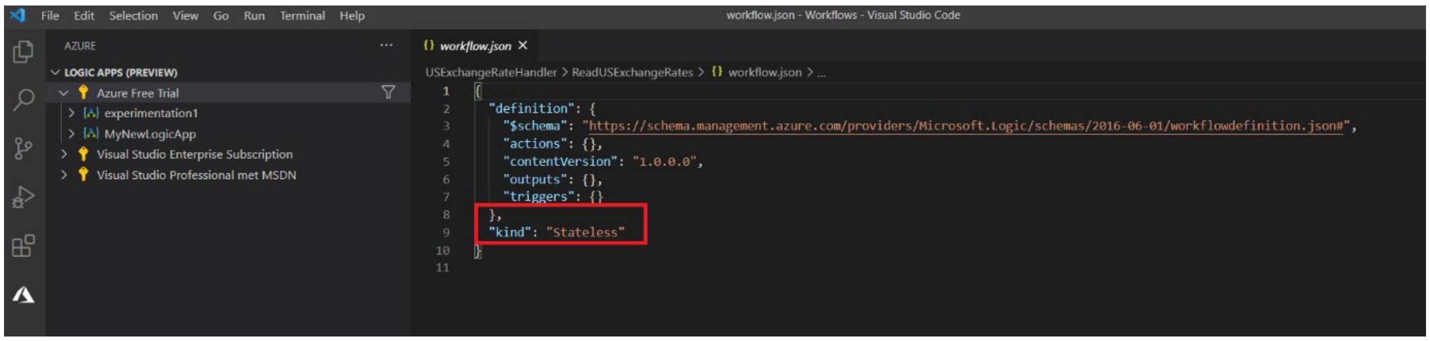
Building a Workflow using Visual Code

The new Logic App brings support for VS Code, enabling you to author a workflow with an IDE like VS Code. There are some perquisites to do so. According to the Microsoft document 'Create stateful and stateless workflows in Visual Studio Code with the Azure Logic Apps (Preview) extension' you need:

- An Azure Storage Emulator version 5.10 (make sure to start the emulator before you begin)
- VS Code version 1.30.1 or higher
- The Azure Account extension - providing a single common Azure sign-in and subscription filtering experience for all other Azure extensions in Visual Studio Code.
- C# for Visual Studio Code extension
- Azure Functions Core Tools 3.0.3245 or later, and
- Azure Logic Apps (Preview) extension for Visual Studio Code - am extension providing the capability for you to create logic apps - where you can build stateful and stateless workflows that locally run in Visual Studio Code and then deploy those logic apps directly to Azure or Docker containers.

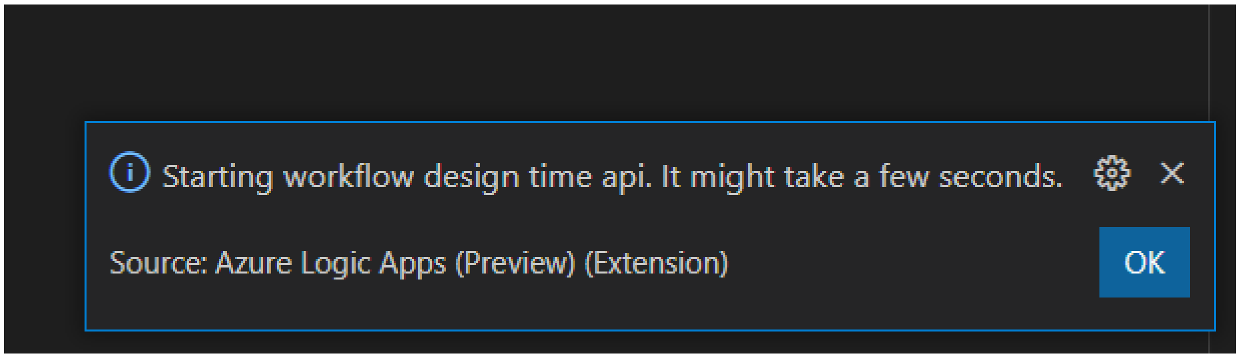
With all the prerequisites in place, you can start creating your workflow. The previously mentioned Microsoft document details how to make one step-by-step. Note that by forgetting one of the prerequisites, you will encounter warnings in VS Code.

When choosing stateless, the Explorer pane will show a generated *workflow.json* with kind: stateless.


File Edit Selection View Go Run Terminal Help
workflow.json - Workflows - Visual Studio Code
AZURE
LOGIC APPS (PREVIEW)
Azure Free Trial
experimentation1
MyNewLogicApp
Visual Studio Enterprise Subscription
Visual Studio Professional met MSDN
workflow.json
USExchangeRateHandler > ReadUSExchangeRates > workflow.json > ...

```
{
  "definition": {
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
    "actions": {},
    "contentVersion": "1.0.0.0",
    "outputs": {},
    "triggers": {}
  },
  "kind": "Stateless"
}
```

Furthermore, when opening the designer, it might take a few seconds. Otherwise, there's an issue; for instance, not all prerequisites are installed correctly.



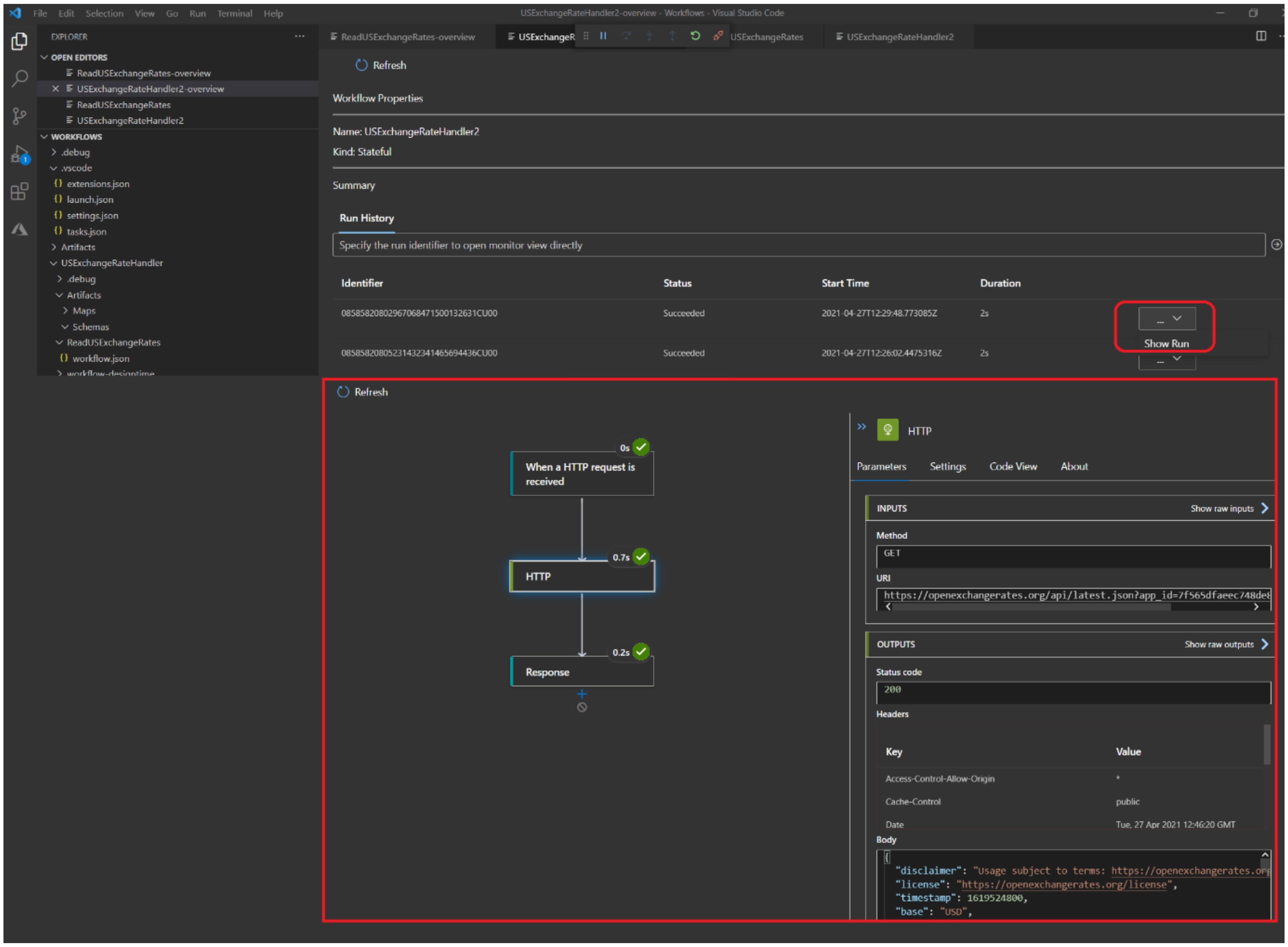
Starting workflow design time api. It might take a few seconds.

Source: Azure Logic Apps (Preview) (Extension)

From within VS Code, you can create a project and add one or more workflows – either stateful or stateless. Note that when choosing stateless, you have less trigger action available than when choosing stateful.

Stateless	**Stateful**
When Events are available in Event Hubs	When Events are available in Event Hubs
NA	HTTP
NA	HTTP Webhook
When an HTTP request is received	When an HTTP request is received
	Recurrence (schedule)
When messages are available in Service Bus Queue	When messages are available in Service Bus Queue
When messages are available in Service Bus Topic	When messages are available in Service Bus Topic

Furthermore, when running the Logic App locally, you cannot examine the run details of a stateless workflow. However, with a stateful workflow, you can investigate the run in Visual Code (when the workflow is running!).



Workflow Properties

Name: USExchangeRateHandler2
Kind: Stateful

Summary

Run History

Specify the run identifier to open monitor view directly

Identifier	Status	Start Time	Duration
08585820802967068471500132631CU00	Succeeded	2021-04-27T12:29:48.773085Z	2s
08585820805231432341465694436CU00	Succeeded	2021-04-27T12:26:02.4475316Z	2s

HTTP

INPUTS

Method: GET

URI: https://openexchangerates.org/api/latest.json?app_id=7f565dfaeec748de

OUTPUTS

Status code: 200

Headers

Key	Value
Access-Control-Allow-Origin	*
Cache-Control	public
Date	Tue, 27 Apr 2021 12:46:20 GMT

Body

```
{
  "disclaimer": "Usage subject to terms: https://openexchangerates.org
  "license": "https://openexchangerates.org/license",
  "timestamp": 1619524800,
  "base": "USD",
```

Note that you need to attach your runtime to either a storage account or a local emulator or use Azurite.

An extensive walkthrough is available on the Microsoft documentation: Create an integration workflow using single-tenant Azure Logic Apps and Visual Studio Code. Furthermore, we also recommend from a developer perspective to look at:

- Edit host and app settings for logic apps in single-tenant Azure Logic Apps, and
- Create parameters for values that change in workflows across environments for single-tenant Azure Logic Apps

After successfully building and debugging your workflow(s) – the next step is to deploy them to Azure. There is a direct option to deploy them to Azure using VS Code described in the upcoming paragraph.

4.2 Deployment Options

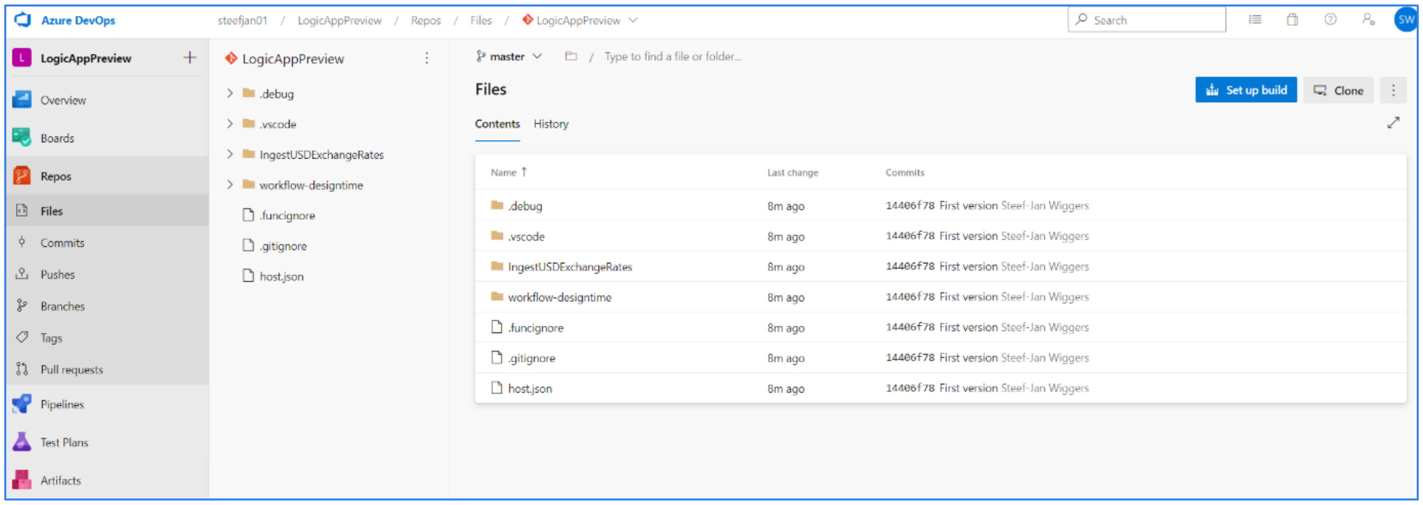
4.2.1 Deploy Workflows with VS Code to Azure

You can directly publish your project to Azure, which deploys your Logic App using the new Logic App (Preview) resource type. You go to Visual Studio Code Activity Bar, select the Azure icon. Next, you on the Azure: Logic Apps (Preview) pane toolbar, select Deploy to Logic App. Subsequently, you will go through a series of steps described in the Microsoft documentation section 'Publish to a new Logic App (Preview) resource.' By following these steps, your Logic Apps and flows will be published in Azure.

4.2.2 Deployment options for Workflows in Logic Apps Standard

Besides directly publishing Logic Apps to Azure, you can also push it to a repo in DevOps first. Next, you can decide whether you want to push the workflows using Azure Pipelines or connecting the repo via the **deployment center** to the Logic App environment in Azure. We will briefly discuss both approaches.

In a DevOps project, navigate to Repos, create a Repo, and clone it in VS Code. Note that you need to have the git extension installed for your VS Code. Next, you can start adding a new Logic App preview project. Build your Logic App locally, run and debug it. If you are satisfied, you can commit and push the project to the repo.



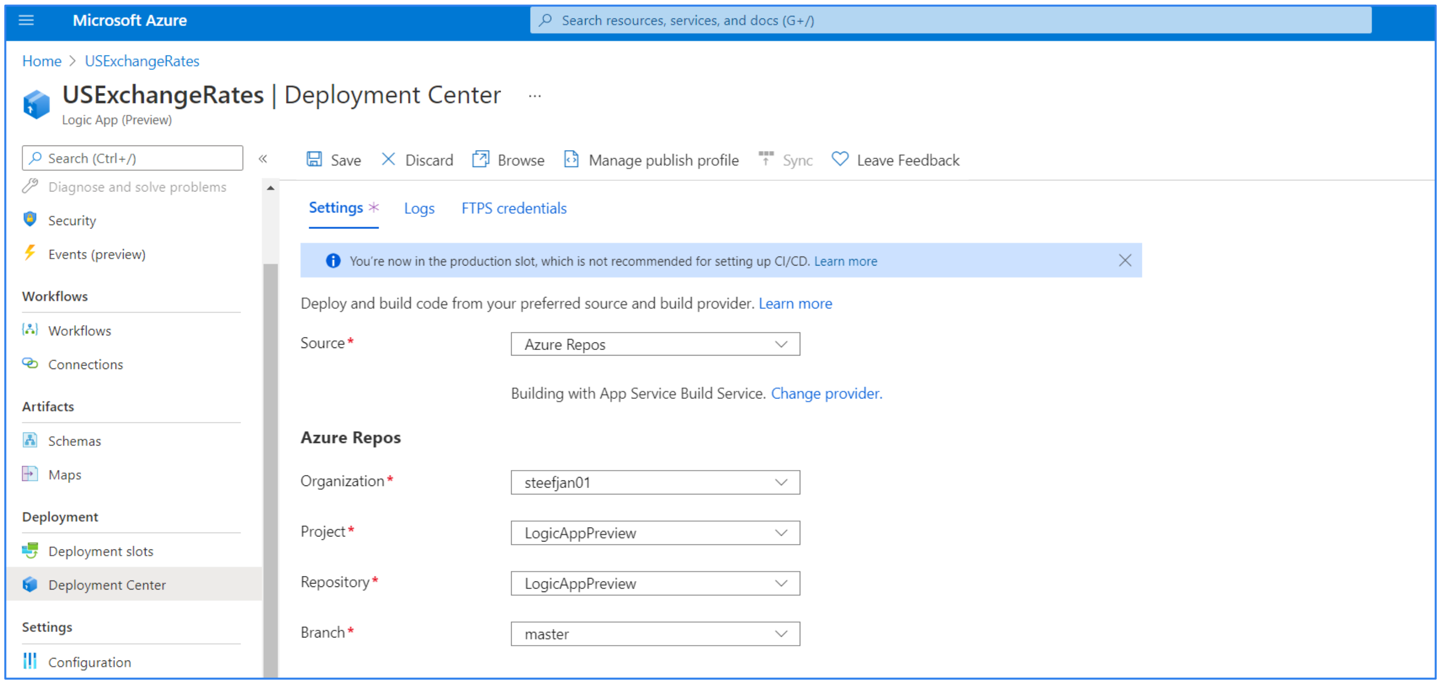
Name ↑	Last change	Commits
.debug	8m ago	14406f78 First version Steef-Jan Wiggers
.vscode	8m ago	14406f78 First version Steef-Jan Wiggers
IngestUSDExchangeRates	8m ago	14406f78 First version Steef-Jan Wiggers
workflow-designtime	8m ago	14406f78 First version Steef-Jan Wiggers
.funcignore	8m ago	14406f78 First version Steef-Jan Wiggers
.gitignore	8m ago	14406f78 First version Steef-Jan Wiggers
host.json	8m ago	14406f78 First version Steef-Jan Wiggers

After having your Logic App in a DevOps repo, the next step is deploying it to its host, i.e., Logic App Standard. You can use, as mentioned earlier, Deployment Center or Azure Pipelines, or you can even use Bicep.

4.2.3

Logic App Deployment Center Approach

Deployment center is the centralized way to get all of your code into an App Service or, in the case of Logic App Standard - workflows. Once you pushed your workflow to the repo in Azure DevOps, you can navigate to your Logic App and choose the Deployment Center in the Deployment part on the left-hand pane. Now you can choose to connect to a repo – with Azure DevOps as one of the options.



Microsoft Azure

Home > USExchangeRates

USExchangeRates | Deployment Center

Logic App (Preview)

You're now in the production slot, which is not recommended for setting up CI/CD. Learn more

Deploy and build code from your preferred source and build provider. Learn more

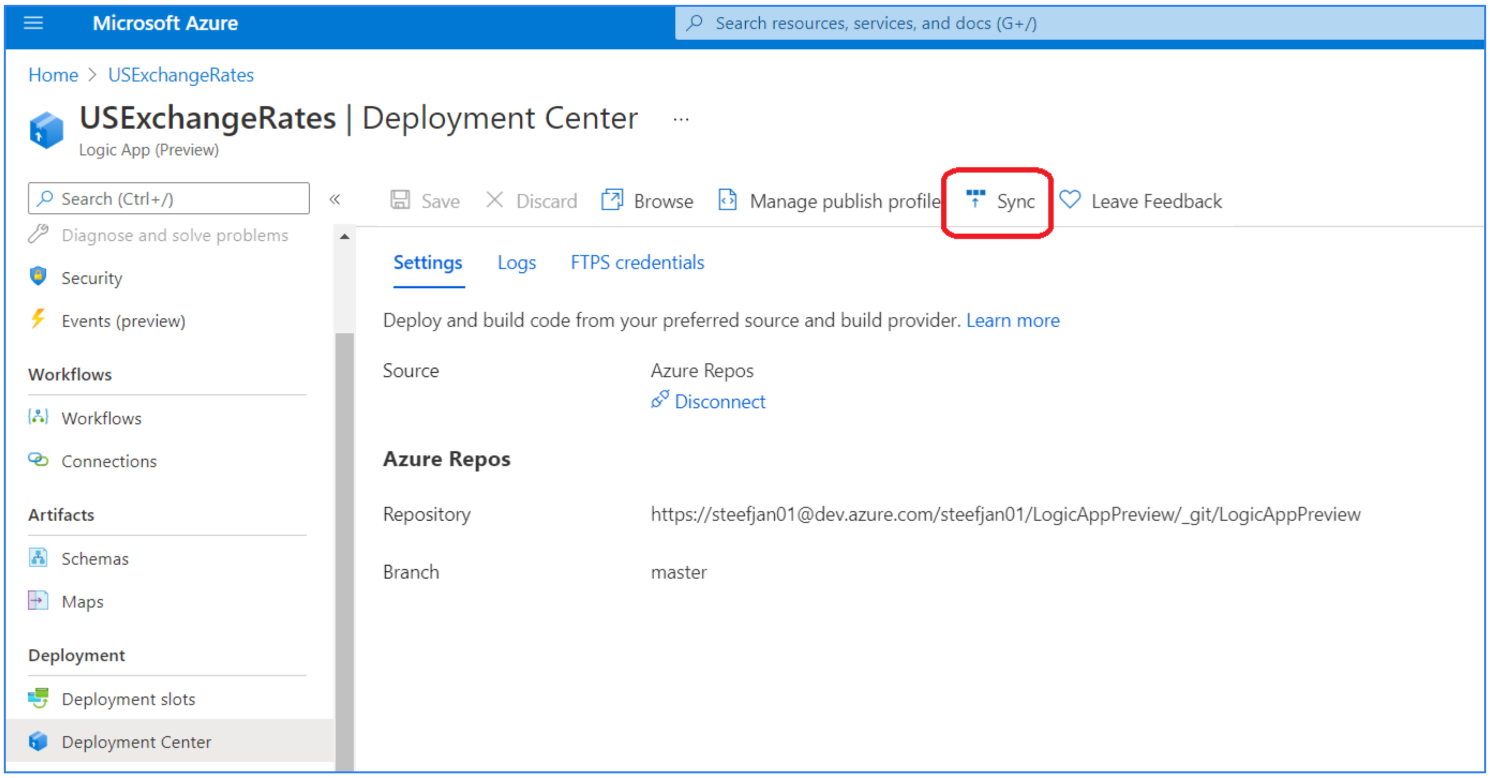
Source* Azure Repos

Building with App Service Build Service. Change provider.

Azure Repos

Organization*	steefjan01
Project*	LogicAppPreview
Repository*	LogicAppPreview
Branch*	master

Next, save and then choose sync.



Microsoft Azure

Home > USExchangeRates

USExchangeRates | Deployment Center

Logic App (Preview)

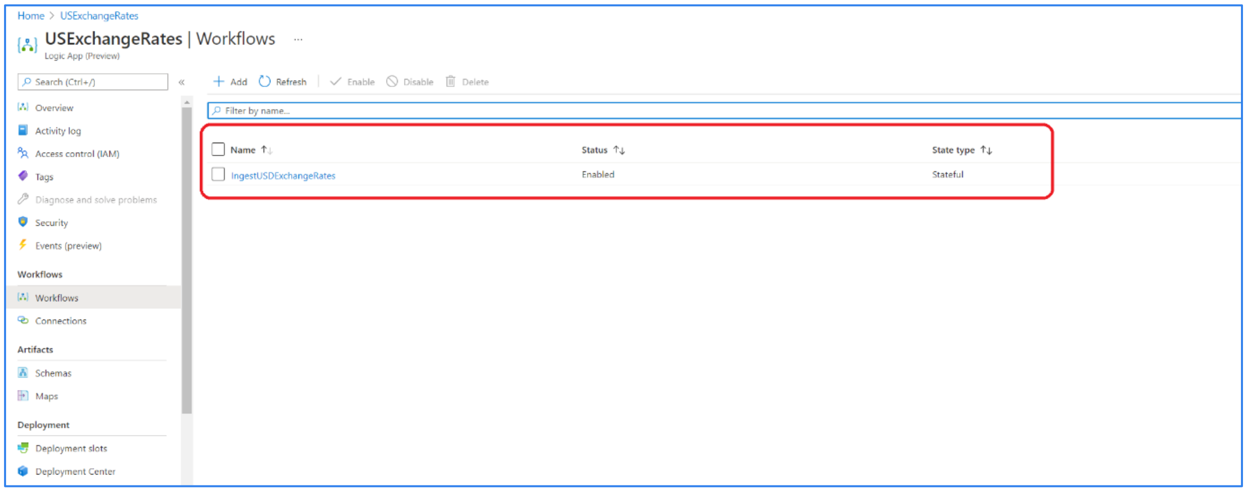
Deploy and build code from your preferred source and build provider. Learn more

Source Azure Repos
Disconnect

Azure Repos

Repository	https://steefjan01@dev.azure.com/steefjan01/LogicAppPreview/_git/LogicAppPreview
Branch	master

You will see in the Logic App one or more workflows appear, depending on the number you build locally in your project. We created one workflow in our project, pushed in an Azure Repo, and synced it with our Logic App.



Name	Status	State type
IngestUSDExchangeRates	Enabled	Stateful

With the Deployment Center, you have multiple options of choosing a repo (location) and have your code (workflow) deployed into your Logic App (Standard – the Logic App is more a container similar to a Function App or App Service). With the Sync feature, you can pull changes from a repo/branch into your Logic App. You need to set up the development center for multiple environments – yet you can, based upon your branching strategy, pull from the correct branch intended for the environment. That's the big plus and doesn't require a pipeline. The governance, however, for deploying to various environments is delegated to your repo, i.e., branching strategy. The other option you have for deploying to multiple environments is using Azure Pipelines, which we will discuss that approach briefly in the next paragraph.

4.2.4

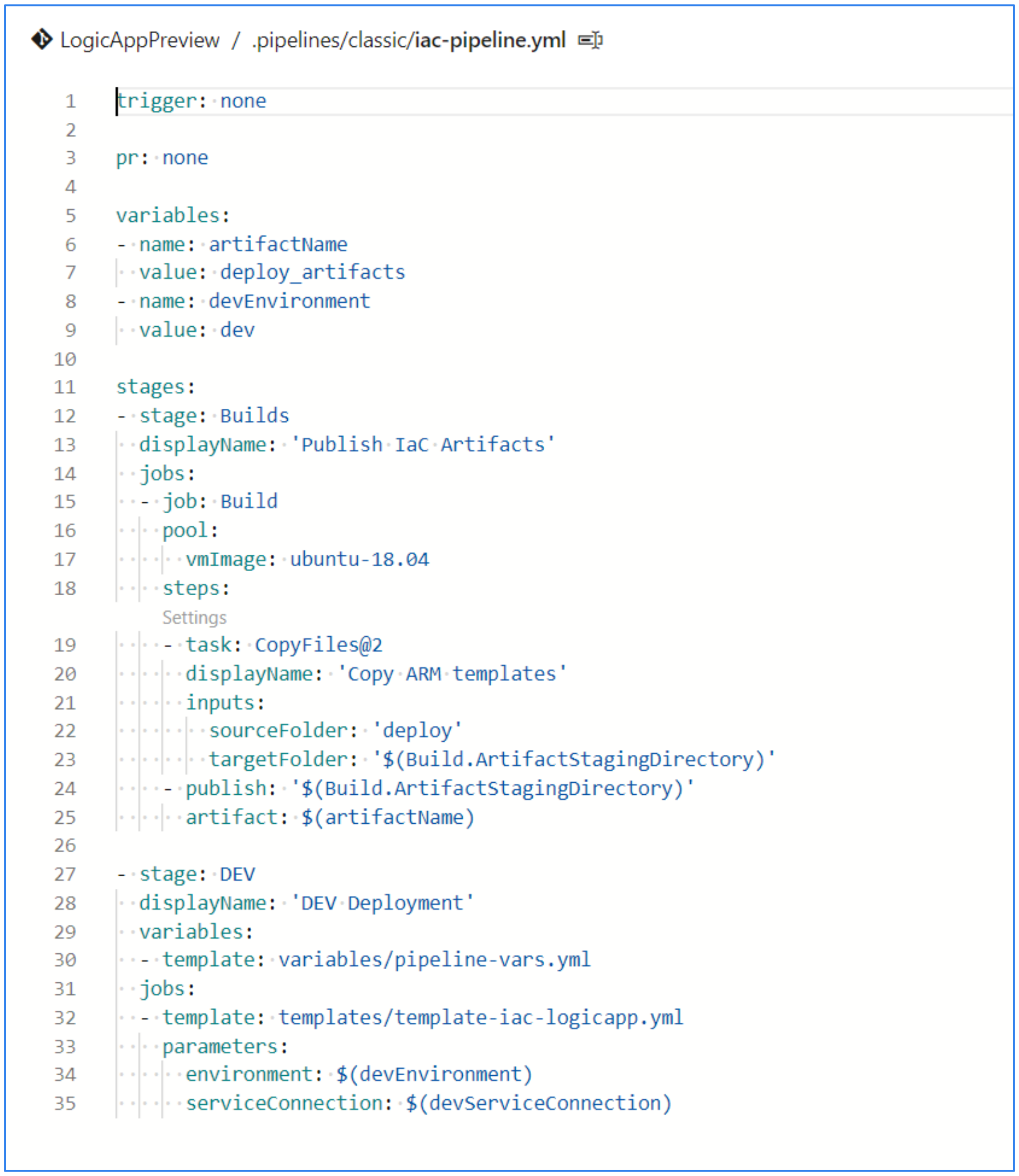
Azure DevOps or GitHub Actions Approach

An alternative to the deployment center approach is building an Azure Pipeline to deploy your workflows to the various environments or use GitHub Actions. During the preview, there wasn't much guidance for this. However, with the GA release, there are samples available. Through GitHub, you can find two options:

→ GitHub Actions: Logic Apps (Single-tenant) DevOps - GitHub Actions

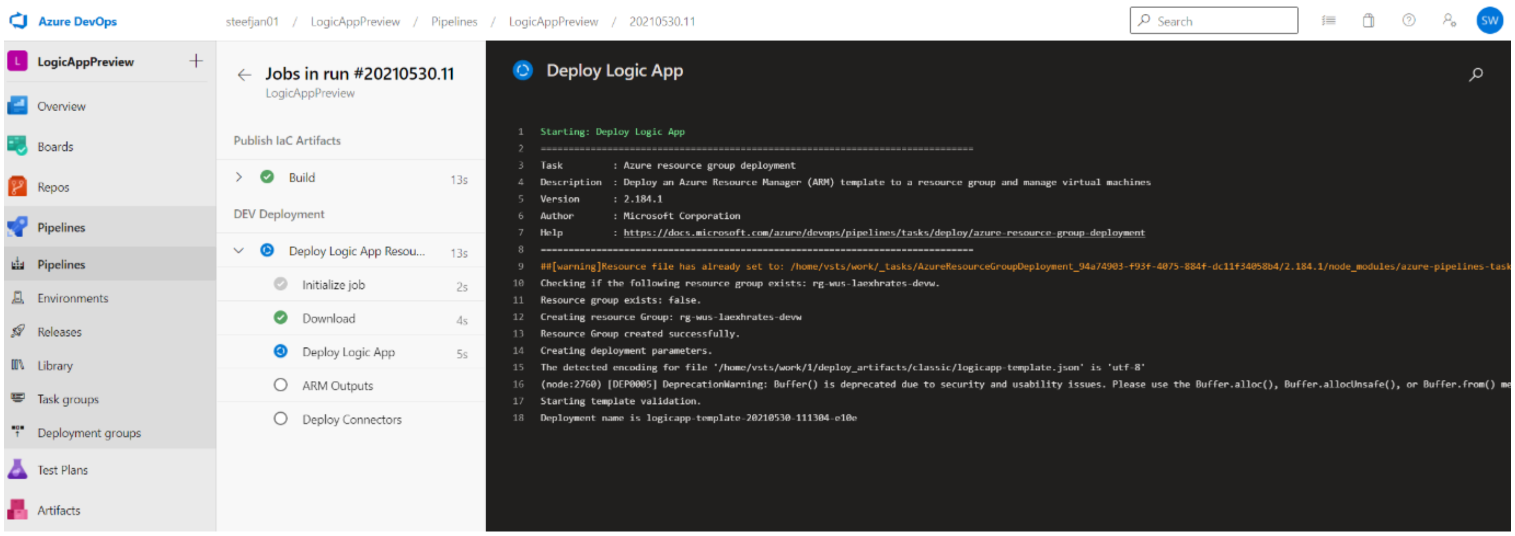
→ Azure Pipelines: Logic Apps (Single-tenant) - Azure DevOps

With the GitHub sample, we leveraged the code to deploy the necessary infrastructure for the single-tenant Logic App.


LogicAppPreview / .pipelines/classic/iac-pipeline.yml

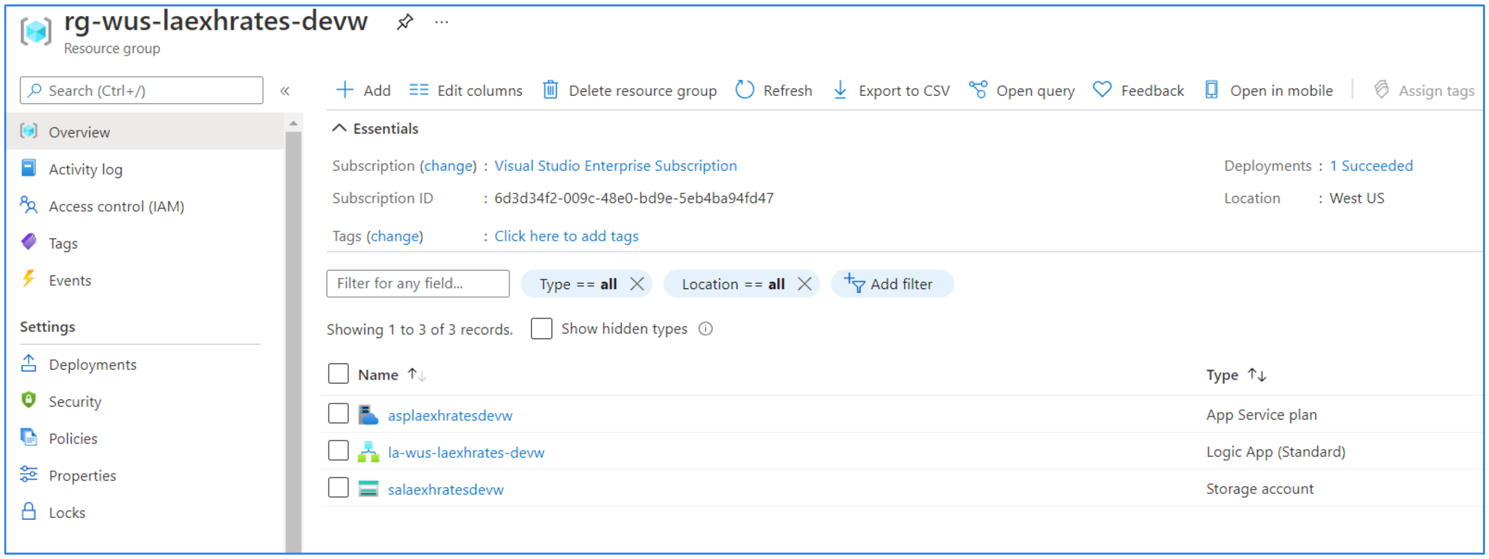
```yaml
trigger: none

pr: none

variables:
- name: artifactName
  value: deploy_artifacts
- name: devEnvironment
  value: dev

stages:
- stage: Builds
  displayName: 'Publish IaC Artifacts'
  jobs:
  - job: Build
    pool:
      vmImage: ubuntu-18.04
    steps:
    Settings
    - task: CopyFiles@2
      displayName: 'Copy ARM templates'
      inputs:
        sourceFolder: 'deploy'
        targetFolder: '$(Build.ArtifactStagingDirectory)'
    - publish: '$(Build.ArtifactStagingDirectory)'
      artifact: $(artifactName)

- stage: DEV
  displayName: 'DEV Deployment'
  variables:
  - template: variables/pipeline-vars.yml
  jobs:
  - template: templates/template-iac-logicapp.yml
    parameters:
      environment: $(devEnvironment)
      serviceConnection: $(devServiceConnection)
```

You can modify the sample(s) to your requirements.



```
Starting: Deploy Logic App
==============================================================================
Task         : Azure resource group deployment
Description  : Deploy an Azure Resource Manager (ARM) template to a resource group and manage virtual machines
Version      : 2.184.1
Author       : Microsoft Corporation
Help         : https://docs.microsoft.com/azure/devops/pipelines/tasks/deploy/azure-resource-group-deployment
==============================================================================
##[warning]Resource file has already set to: /home/vsts/work/_tasks/AzureResourceGroupDeployment_94a74903-f93f-4075-884f-dc11f34058b4/2.184.1/node_modules/azure-pipelines-task
Checking if the following resource group exists: rg-wus-laexhrates-devw.
Resource group exists: false.
Creating resource Group: rg-wus-laexhrates-devw
Resource Group created successfully.
Creating deployment parameters.
The detected encoding for file '/home/vsts/work/1/deploy_artifacts/classic/logicapp-template.json' is 'utf-8'
(node:2760) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() me
Starting template validation.
Deployment name is logicapp-template-20210530-111304-e10e
```

When running the pipeline, the resources necessary for the Logic App Standard are deployed.



Subscription (change) : Visual Studio Enterprise Subscription
Subscription ID : 6d3d34f2-009c-48e0-bd9e-5eb4ba94fd47
Tags (change) : Click here to add tags
Deployments : 1 Succeeded
Location : West US

Showing 1 to 3 of 3 records.

Name	Type
asplaexhratesdevw	App Service plan
la-wus-laexhrates-devw	Logic App (Standard)
salaexhratesdevw	Storage account

Lastly, the Microsoft documentation [Set up DevOps deployment for single-tenant Azure Logic Apps] provided details to set up DevOps.

4.2.5

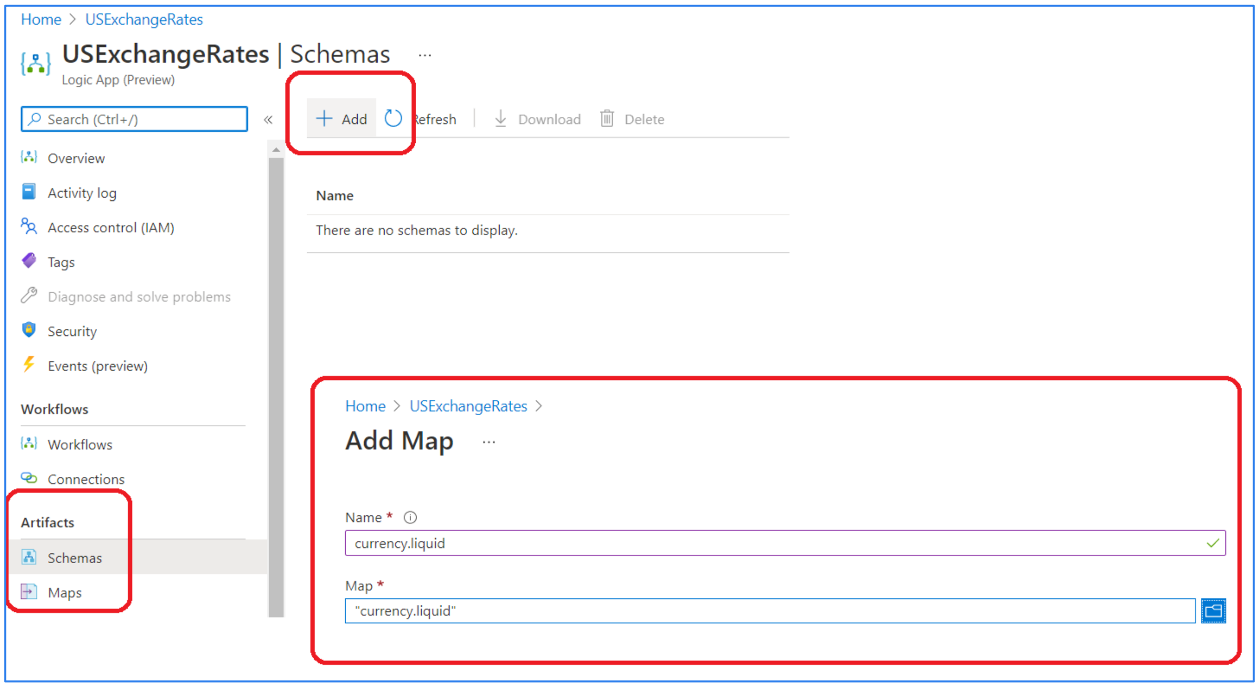
Bicep Language

Alternatively, you can try using Bicep, a Domain Specific Language (DSL), for deploying Azure resources declaratively. It is a transparent abstraction over ARM and ARM templates, which means anything can be done in an ARM Template you can do in Bicep (outside of known temporary limitations). Although still in preview, it is worth looking into Bicep to deploy Azure resources, including Logic App Standard and its dependencies. The Bicep Language is available as a VS-Code plug-in, and tutorials are available on GitHub.

4.3

Mapping in Logic Apps

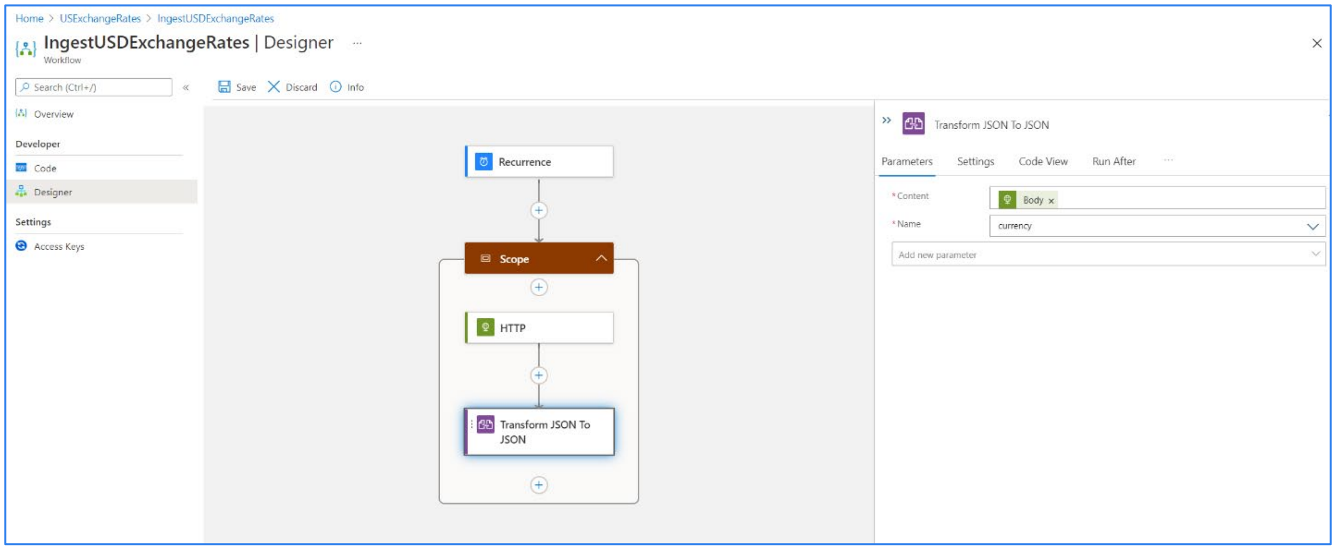
We haven't touched yet the ability to leverage maps and schema's in the new Logic App version. This version doesn't have a dependency on the integration account. You can create maps and schemas using tools you are familiar with – and upload them using the portal.



Name *
currency.liquid

Map *
"currency.liquid"

Once you have uploaded your schemas or maps, you can use them in your workflows:

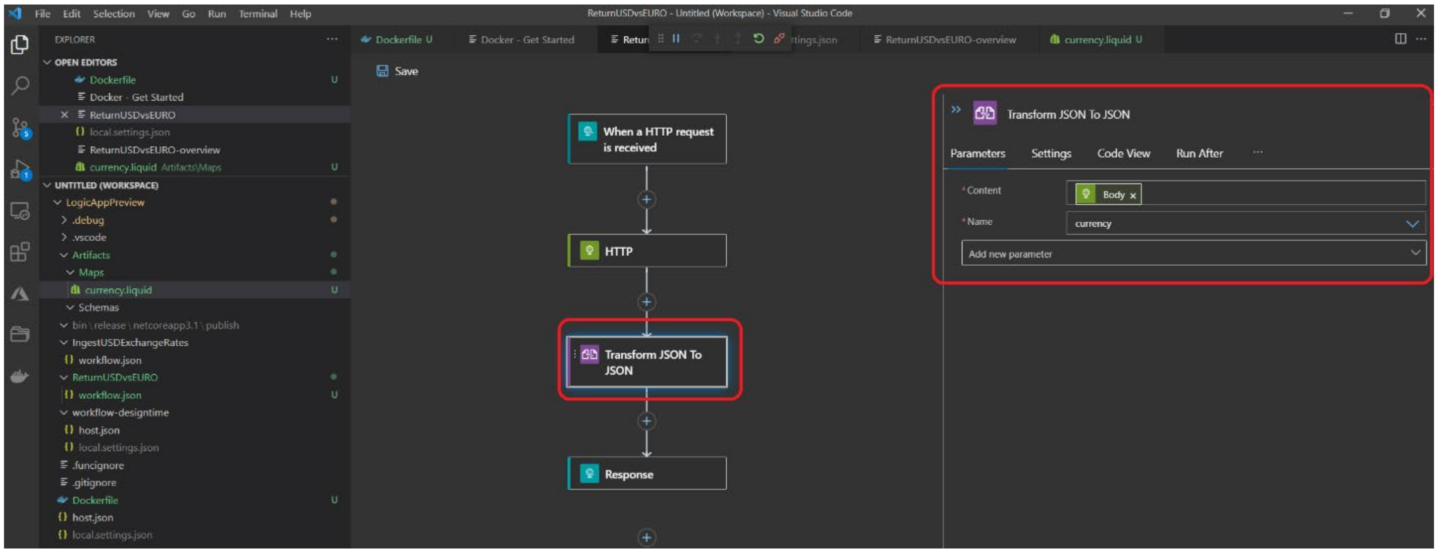


Another approach is to use the schema and maps in your project in VS Code. You will need to add the respective Maps and Schemas folders to your Visual Studio Code project Artifacts folder.

In the local solution we build fetching USD exchanges rates when invoking a workflow, we applied the following liquid map to receive a subset in our desired json format:

```
{
  "Timestamp" : "{{ content.timestamp }}",
  "Base" : "{{ content.base }}",
  "Date" : "{{ "now" | Date: "MM/dd/yyyy" }}",
  "Rates" : [
    {
      "EUR" : "{{ content.rates.EUR }}",
      "GBP" : {{ content.rates.GBP }}
    }
  ]
}
```

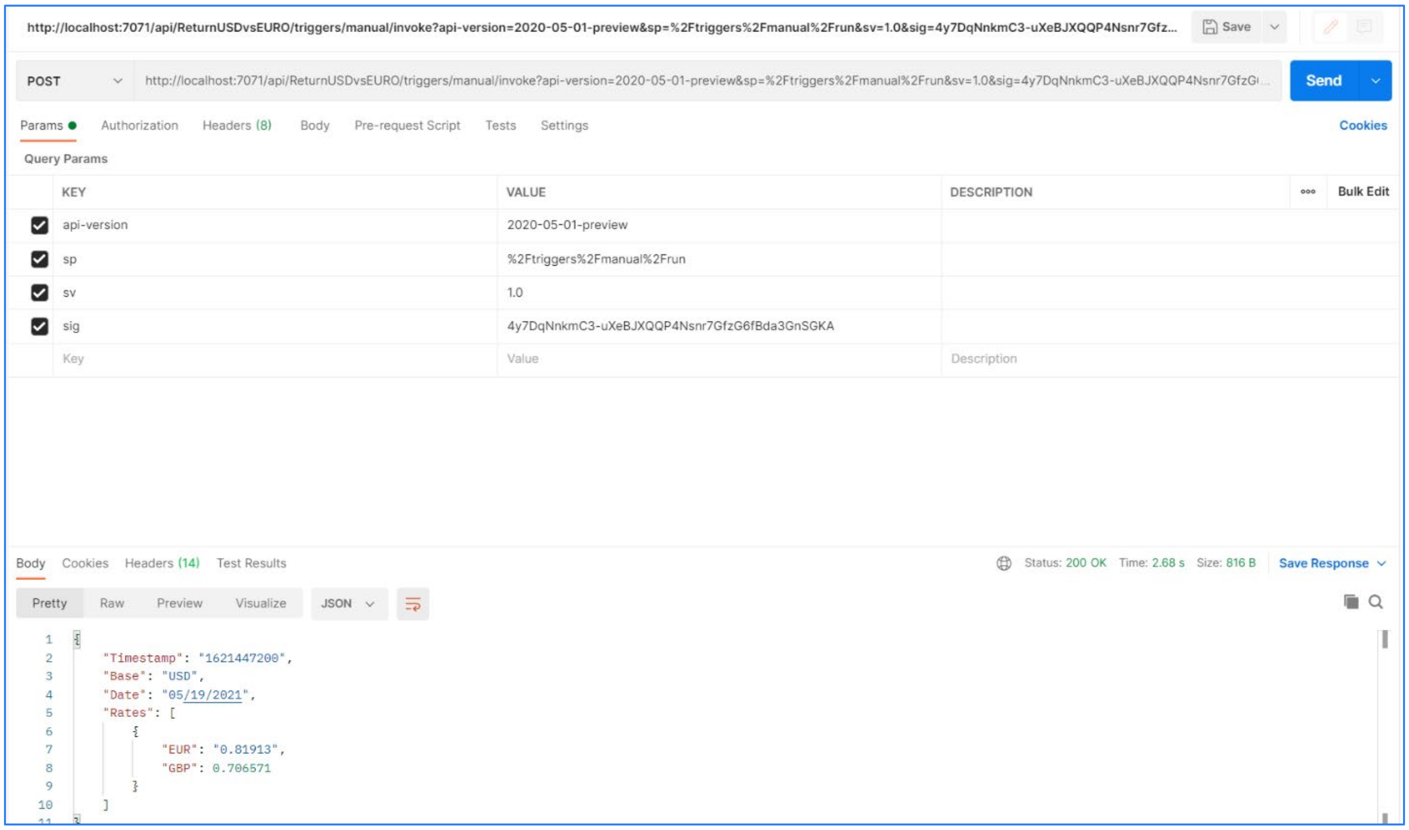
The map is placed in the maps folder. Next, with the designer, we can pick this map.


ReturnUSDvsEURO - Untitled (Workspace) - Visual Studio Code
When a HTTP request is received
HTTP
Transform JSON To JSON
Response
Save
Parameters
Settings
Code View
Run After

Transform JSON To JSON

* Content	Body
* Name	currency
Add new parameter	

When testing the workflow locally, we see the response with the desired format:


POST http://localhost:7071/api/ReturnUSDvsEURO/triggers/manual/invoke?api-version=2020-05-01-preview&sp=%2Ftriggers%2Fmanual%2Frun&sv=1.0&sig=4y7DqNnkmC3-uXeBJXQQP4Nsnr7GfzG…

Query Params

	KEY	VALUE	DESCRIPTION
✓	api-version	2020-05-01-preview	
✓	sp	%2Ftriggers%2Fmanual%2Frun	
✓	sv	1.0	
✓	sig	4y7DqNnkmC3-uXeBJXQQP4Nsnr7GfzG6fBda3GnSGKA	

Status: 200 OK Time: 2.68 s Size: 816 B

```
{
    "Timestamp": "1621447200",
    "Base": "USD",
    "Date": "05/19/2021",
    "Rates": [
        {
            "EUR": "0.81913",
            "GBP": 0.706571
        }
    ]
```

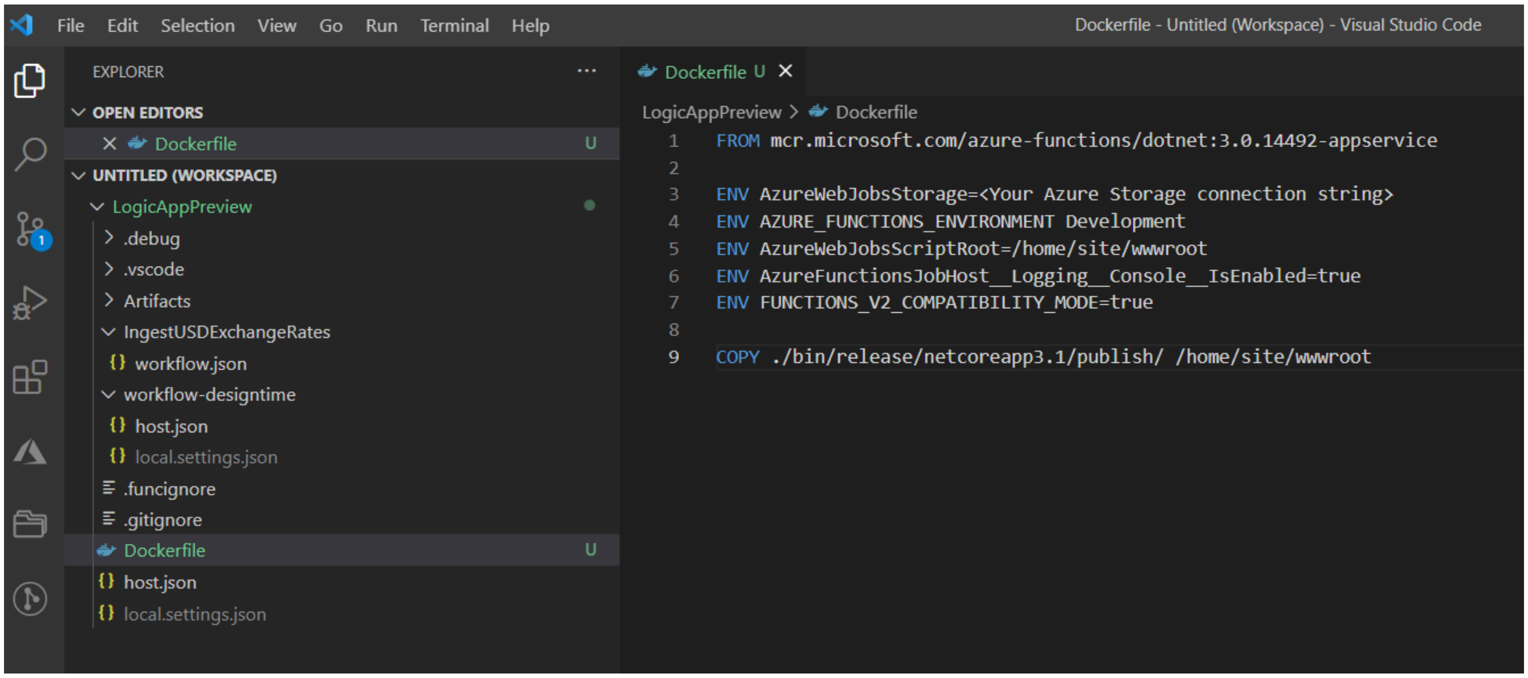
With the ability to edit schemas and maps locally, you can also test them locally, enhancing productivity.

5
Running Logic Apps in a Container

With Logic App (Preview) running in a Function App Service Runtime also opens up the possibility to host a Logic App in a container. The current Func App Runtime brings support for Docker. To host your Logic App in a container, you will need the latest Azure Functions Core tools, Azure CLI, and Docker on your developer machine.

In VS Code, create a Dockerfile in the project of the Logic App. Note that the file has no file extension and is just called Dockerfile. It should have the following contents:

```
FROM mcr.microsoft.com/azure-functions/dotnet:3.0.14492-appservice

ENV AzureWebJobsStorage=<Your Azure Storage connection string>
ENV AZURE_FUNCTIONS_ENVIRONMENT Development
ENV AzureWebJobsScriptRoot=/home/site/wwwroot
ENV AzureFunctionsJobHost__Logging__Console__IsEnabled=true
ENV FUNCTIONS_V2_COMPATIBILITY_MODE=true

COPY ./bin/release/netcoreapp3.1/publish/ /home/site/wwwroot
```



```
FROM mcr.microsoft.com/azure-functions/dotnet:3.0.14492-appservice

ENV AzureWebJobsStorage=<Your Azure Storage connection string>
ENV AZURE_FUNCTIONS_ENVIRONMENT Development
ENV AzureWebJobsScriptRoot=/home/site/wwwroot
ENV AzureFunctionsJobHost__Logging__Console__IsEnabled=true
ENV FUNCTIONS_V2_COMPATIBILITY_MODE=true

COPY ./bin/release/netcoreapp3.1/publish/ /home/site/wwwroot
```

Next, you fill in your Azure Storage connection string for the AzureWebJobsStorage environment variable so that the Logic App in the container can use it. Once that is done, you can open the Terminal in VS Code and, according to the documentation, build your Docker container image by using your Docker file and running this command:

```
docker build --tag local/workflowcontainer .
```

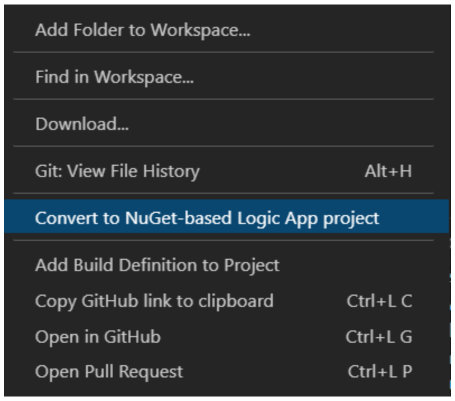
However, in the docker file, the last line has the command

COPY ./bin/release/netcoreapp3.1/publish/

This folder doesn't exist yet; hence you need to build and release your project using:

dotnet build -c release
dotnet publish -c release

Yet, the Logic App project doesn't contain a .proj or .sln file, at least not if you have created one. You can make the */bin/release/netcoreapp3.1/publish* folder yourself and execute the Docker build command again. The key here is to convert the project to a Nu-Get based Logic App project:

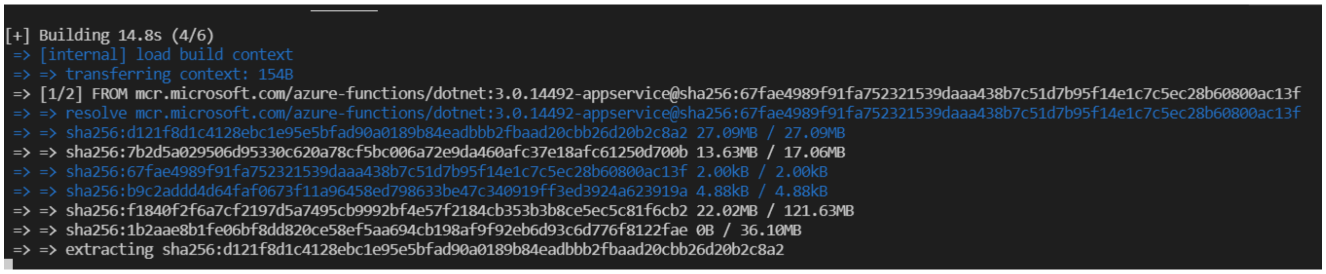

Add Folder to Workspace...
Find in Workspace...
Download...
Git: View File History Alt+H
Convert to NuGet-based Logic App project
Add Build Definition to Project
Copy GitHub link to clipboard Ctrl+L C
Open in GitHub Ctrl+L G
Open Pull Request Ctrl+L P

The commands then will work!

With the Logic App we build, we executed:

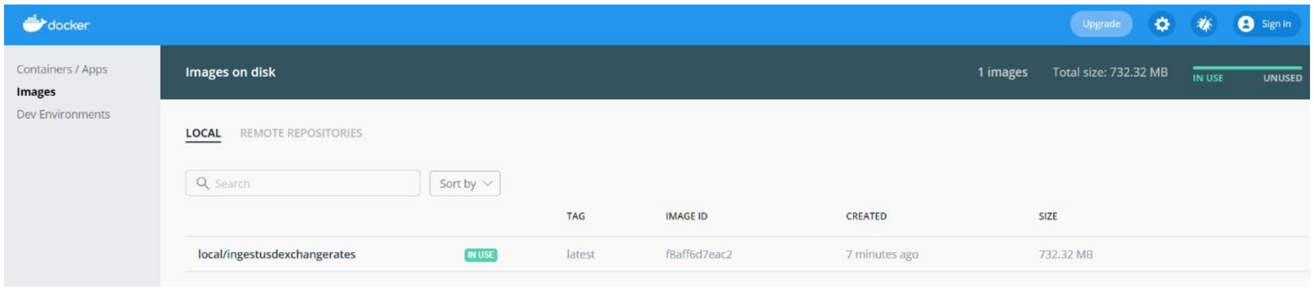
docker build --tag local/ingestusdexchangerates .

And led to building a container:



```
[+] Building 14.8s (4/6)
 => [internal] load build context
 => => transferring context: 154B
 => [1/2] FROM mcr.microsoft.com/azure-functions/dotnet:3.0.14492-appservice@sha256:67fae4989f91fa752321539daaa438b7c51d7b95f14e1c7c5ec28b60800ac13f
 => => resolve mcr.microsoft.com/azure-functions/dotnet:3.0.14492-appservice@sha256:67fae4989f91fa752321539daaa438b7c51d7b95f14e1c7c5ec28b60800ac13f
 => => sha256:d121f8d1c4128ebc1e95e5bfad90a0189b84eadbbb2fbaad20cbb26d20b2c8a2 27.09MB / 27.09MB
 => => sha256:7b2d5a029506d95330c620a78cf5bc006a72e9da460afc37e18afc61250d700b 13.63MB / 17.06MB
 => => sha256:67fae4989f91fa752321539daaa438b7c51d7b95f14e1c7c5ec28b60800ac13f 2.00kB / 2.00kB
 => => sha256:b9c2addd4d64faf0673f11a96458ed798633be47c340919ff3ed3924a623919a 4.88kB / 4.88kB
 => => sha256:f1840f2f6a7cf2197d5a7495cb9992bf4e57f2184cb353b3b8ce5ec5c81f6cb2 22.02MB / 121.63MB
 => => sha256:1b2aae8b1fe06bf8dd820ce58ef5aa694cb198af9f92eb6d93c6d776f8122fae 0B / 36.10MB
 => => extracting sha256:d121f8d1c4128ebc1e95e5bfad90a0189b84eadbbb2fbaad20cbb26d20b2c8a2
```

Since the Logic App doesn't contain an endpoint as it is a Logic App with a recurrence.



		TAG	IMAGE ID	CREATED	SIZE
local/ingestusdexchangerates	IN USE	latest	f8aff6d7eac2	7 minutes ago	732.32 MB

When exposing an endpoint with your workflow and package it in a container, see the following tip:

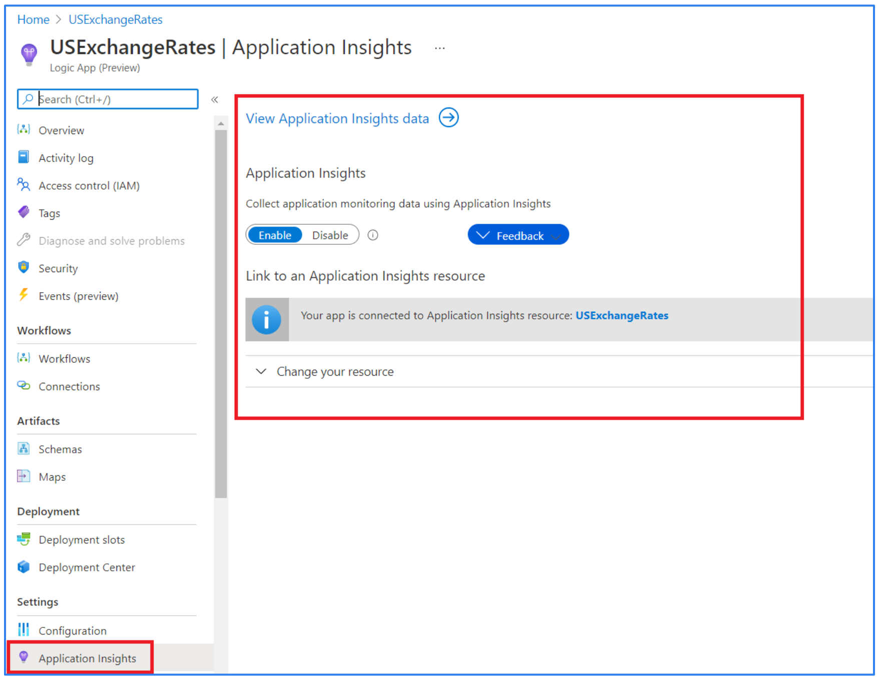
→ https://microsoft.GitHub.io/AzureTipsAndTricks/blog/tip311.html

→ And a tutorial on the Microsoft Docs: Create stateful and stateless workflows in Visual Studio Code with the Azure Logic Apps (Preview) extension

6

Monitoring Logic Apps with Application Insights

Up to now, we have discussed developing and deploying Logic App workflows. Next, we will discuss monitoring. With the support for the Function runtime, you can benefit from the built-in integration with Application Insights. You can navigate to Logic App (Preview), and under **Settings**, select **Application Insights** in the Azure Portal.



Home > USExchangeRates

USExchangeRates | Application Insights ...

Logic App (Preview)

View Application Insights data

Application Insights

Collect application monitoring data using Application Insights

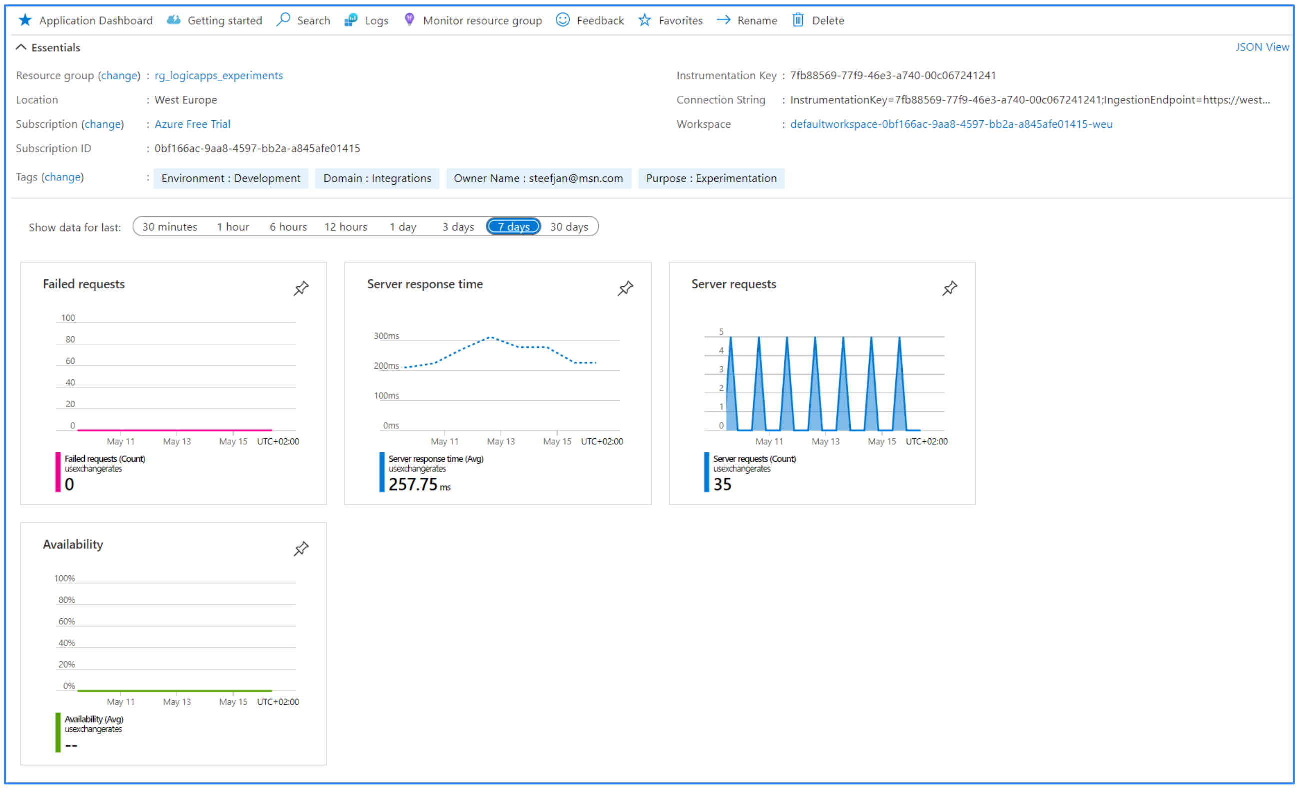
Enable | Disable

Link to an Application Insights resource

Your app is connected to Application Insights resource: USExchangeRates

Change your resource

In our case with the Application Insights was already enabled, and we could look at the data. Note that you can enable/disable the application monitoring, change the resource, i.e., select a different instance of Application Insights. Below you see data from our Logic App (containing one workflow):



Resource group (change) : rg_logicapps_experiments
Location : West Europe
Subscription (change) : Azure Free Trial
Subscription ID : 0bf166ac-9aa8-4597-bb2a-a845afe01415
Tags (change) : Environment : Development | Domain : Integrations | Owner Name : steefjan@msn.com | Purpose : Experimentation

Instrumentation Key : 7fb88569-77f9-46e3-a740-00c067241241
Connection String : InstrumentationKey=7fb88569-77f9-46e3-a740-00c067241241;IngestionEndpoint=https://west...
Workspace : defaultworkspace-0bf166ac-9aa8-4597-bb2a-a845afe01415-weu

Show data for last: 30 minutes | 1 hour | 6 hours | 12 hours | 1 day | 3 days | [✓ 7 days] | 30 days

Failed requests (Count) usexchangerates 0

Server response time (Avg) usexchangerates 257.75 ms

Server requests (Count) usexchangerates 35

Availability (Avg) usexchangerates --

According to a tech community blog post on Application Insights for the preview Logic App, each time a workflow-related event happens, for example, when a workflow is triggered, or an action runs, the runtime emits various traces. These traces cover the lifetime of the workflow run and include, but aren't limited to, the following types:

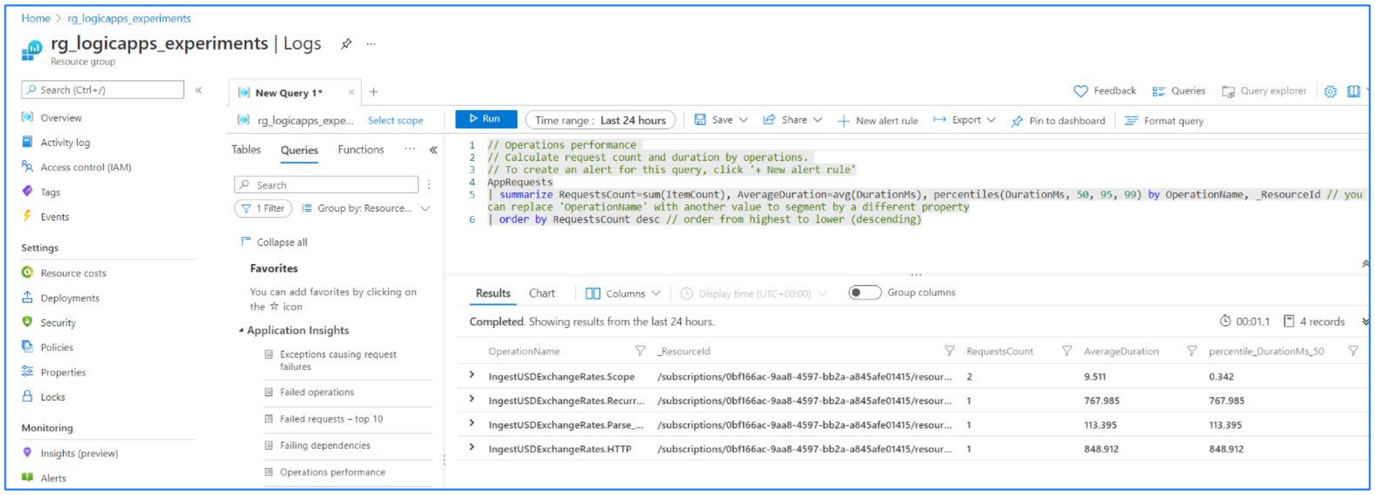
- Service activity, such as start, stop, and errors.
- Jobs and dispatcher activity.

- Workflow activity, such as trigger, action, and run.
- Storage request activity, such as success or failure.
- HTTP request activity, such as inbound, outbound, success, and failure.
- Ad-hoc development traces, such as debug messages.

The post also details the severity level applicable to the trace types and how to set the level in the *host.json* file.

```
{
   "version": "2.0",
   "logging": {
      "logLevel": {
         "Host.Triggers.Workflows": "Trace"
      }
   }
}
```

Next to instrumenting the Logic App (and its workflows), you want to examine logs and telemetry leveraging the Application Insights attached to it. You can select **Logs** under **Monitoring** in your **Application Insights**. Run available queries or create your own. You can, for instance, select a general query to view the performance.

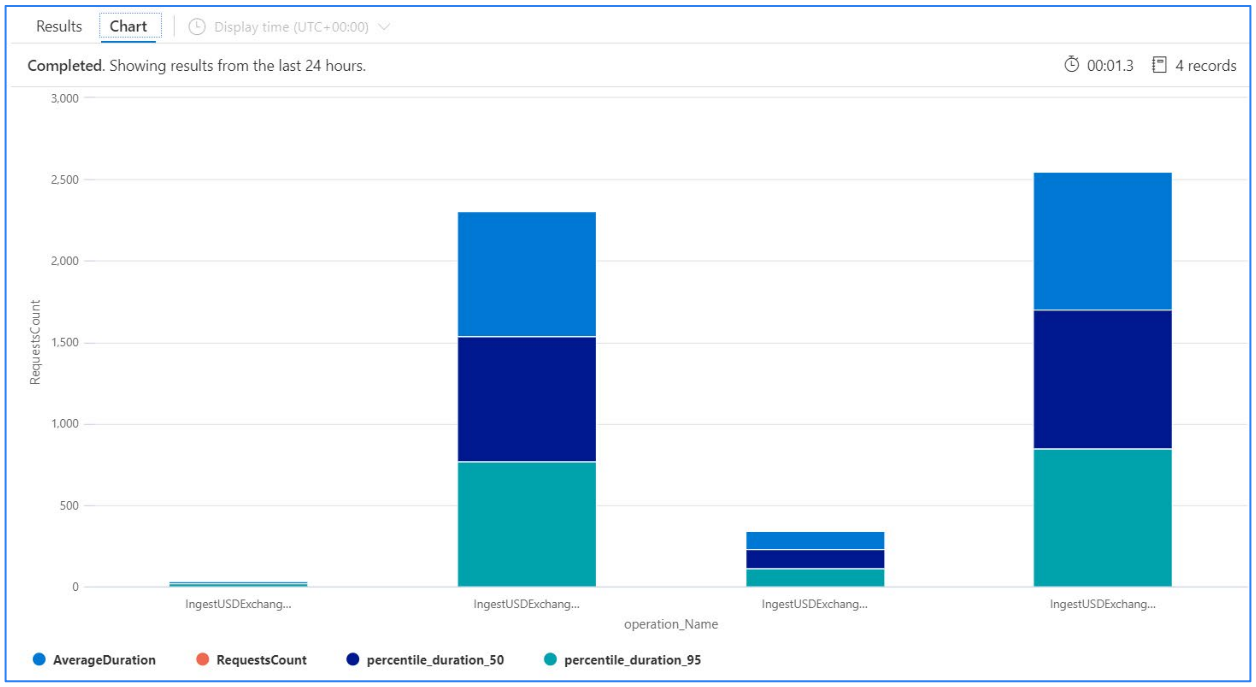


```
// Operations performance
// Calculate request count and duration by operations.
// To create an alert for this query, click '+ New alert rule'
AppRequests
| summarize RequestsCount=sum(ItemCount), AverageDuration=avg(DurationMs), percentiles(DurationMs, 50, 95, 99) by OperationName, _ResourceId // you can replace 'OperationName' with another value to segment by a different property
| order by RequestsCount desc // order from highest to lower (descending)
```

Completed. Showing results from the last 24 hours. 00:01.1 4 records

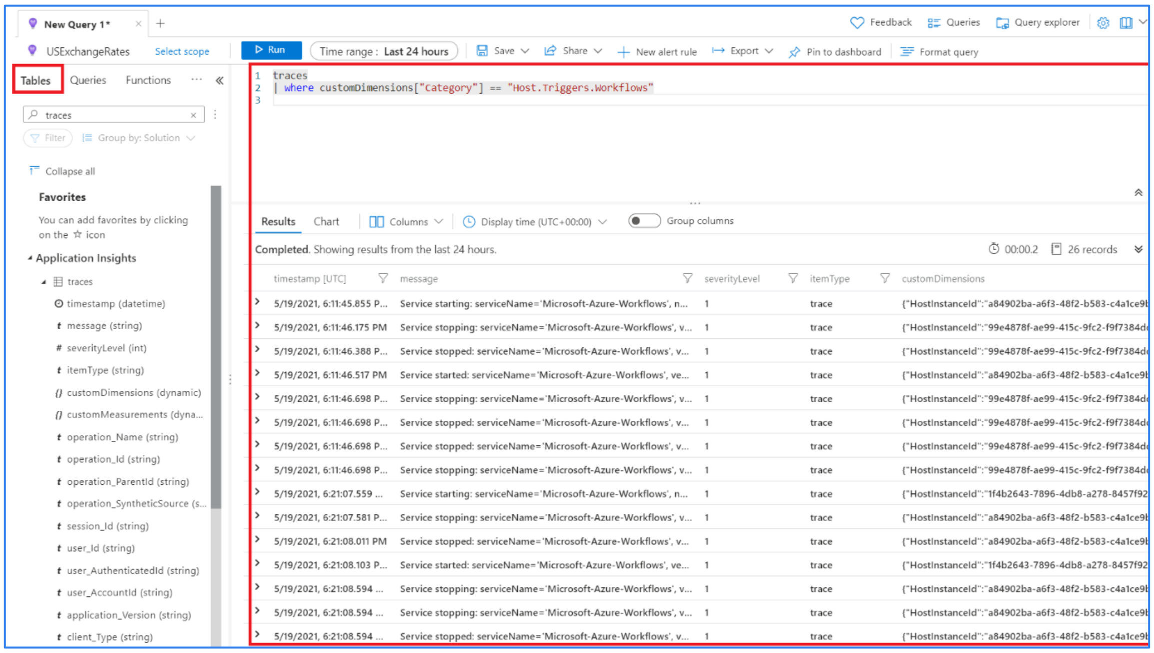
OperationName	_ResourceId	RequestsCount	AverageDuration	percentile_DurationMs_50
IngestUSDExchangeRates.Scope	/subscriptions/0bf166ac-9aa8-4597-bb2a-a845afe01415/resour...	2	9.511	0.342
IngestUSDExchangeRates.Recurr...	/subscriptions/0bf166ac-9aa8-4597-bb2a-a845afe01415/resour...	1	767.985	767.985
IngestUSDExchangeRates.Parse_...	/subscriptions/0bf166ac-9aa8-4597-bb2a-a845afe01415/resour...	1	113.395	113.395
IngestUSDExchangeRates.HTTP	/subscriptions/0bf166ac-9aa8-4597-bb2a-a845afe01415/resour...	1	848.912	848.912

And look at Chart if you like:



You can also dive into the traces depending on how the level is set in the host.json file (as explained in the tech community blog post).

As explained in a subsequent blog post, your Logic App is integrated with Application Insights giving you access to all telemetry data in the form of tables in a database. Workflow execution-specific logs are available under the traces table.



New Query 1*

USExchangeRates Select scope | Run | Time range : Last 24 hours | Save | Share | New alert rule | Export | Pin to dashboard | Format query

Tables | Queries | Functions

traces

Filter | Group by: Solution

Collapse all

Favorites

You can add favorites by clicking on the ☆ icon

Application Insights
- traces
  - timestamp (datetime)
  - message (string)
  - severityLevel (int)
  - itemType (string)
  - customDimensions (dynamic)
  - customMeasurements (dyna...
  - operation_Name (string)
  - operation_Id (string)
  - operation_ParentId (string)
  - operation_SyntheticSource (s...
  - session_Id (string)
  - user_Id (string)
  - user_AuthenticatedId (string)
  - user_AccountId (string)
  - application_Version (string)
  - client_Type (string)

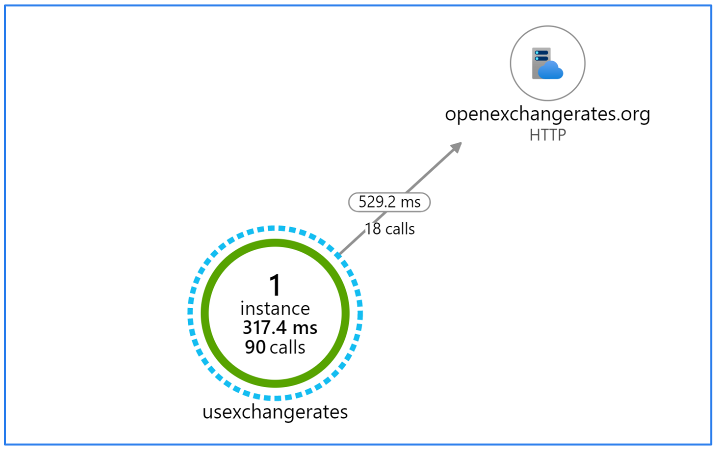
```
traces
| where customDimensions["Category"] == "Host.Triggers.Workflows"
```

Results | Chart | Columns | Display time (UTC+00:00) | Group columns

Completed. Showing results from the last 24 hours. 00:00.2 26 records

timestamp [UTC]	message	severityLevel	itemType	customDimensions
5/19/2021, 6:11:45.855 P...	Service starting: serviceName='Microsoft-Azure-Workflows', n...	1	trace	{"HostInstanceId":"a84902ba-a6f3-48f2-b583-c4a1ce9b
5/19/2021, 6:11:46.175 PM	Service stopping: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"99e4878f-ae99-415c-9fc2-f9f7384d
5/19/2021, 6:11:46.388 P...	Service stopped: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"99e4878f-ae99-415c-9fc2-f9f7384d
5/19/2021, 6:11:46.517 PM	Service started: serviceName='Microsoft-Azure-Workflows', ve...	1	trace	{"HostInstanceId":"a84902ba-a6f3-48f2-b583-c4a1ce9b
5/19/2021, 6:11:46.698 P...	Service stopping: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"99e4878f-ae99-415c-9fc2-f9f7384d
5/19/2021, 6:11:46.698 P...	Service stopped: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"99e4878f-ae99-415c-9fc2-f9f7384d
5/19/2021, 6:11:46.698 P...	Service stopped: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"99e4878f-ae99-415c-9fc2-f9f7384d
5/19/2021, 6:11:46.698 P...	Service stopping: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"99e4878f-ae99-415c-9fc2-f9f7384d
5/19/2021, 6:21:07.559 ...	Service starting: serviceName='Microsoft-Azure-Workflows', n...	1	trace	{"HostInstanceId":"1f4b2643-7896-4db8-a278-8457f92
5/19/2021, 6:21:07.581 P...	Service stopping: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"a84902ba-a6f3-48f2-b583-c4a1ce9b
5/19/2021, 6:21:08.011 PM	Service stopped: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"a84902ba-a6f3-48f2-b583-c4a1ce9b
5/19/2021, 6:21:08.103 P...	Service started: serviceName='Microsoft-Azure-Workflows', ve...	1	trace	{"HostInstanceId":"1f4b2643-7896-4db8-a278-8457f92
5/19/2021, 6:21:08.594 ...	Service stopping: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"a84902ba-a6f3-48f2-b583-c4a1ce9b
5/19/2021, 6:21:08.594 ...	Service stopping: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"a84902ba-a6f3-48f2-b583-c4a1ce9b
5/19/2021, 6:21:08.594 ...	Service stopped: serviceName='Microsoft-Azure-Workflows', v...	1	trace	{"HostInstanceId":"a84902ba-a6f3-48f2-b583-c4a1ce9b

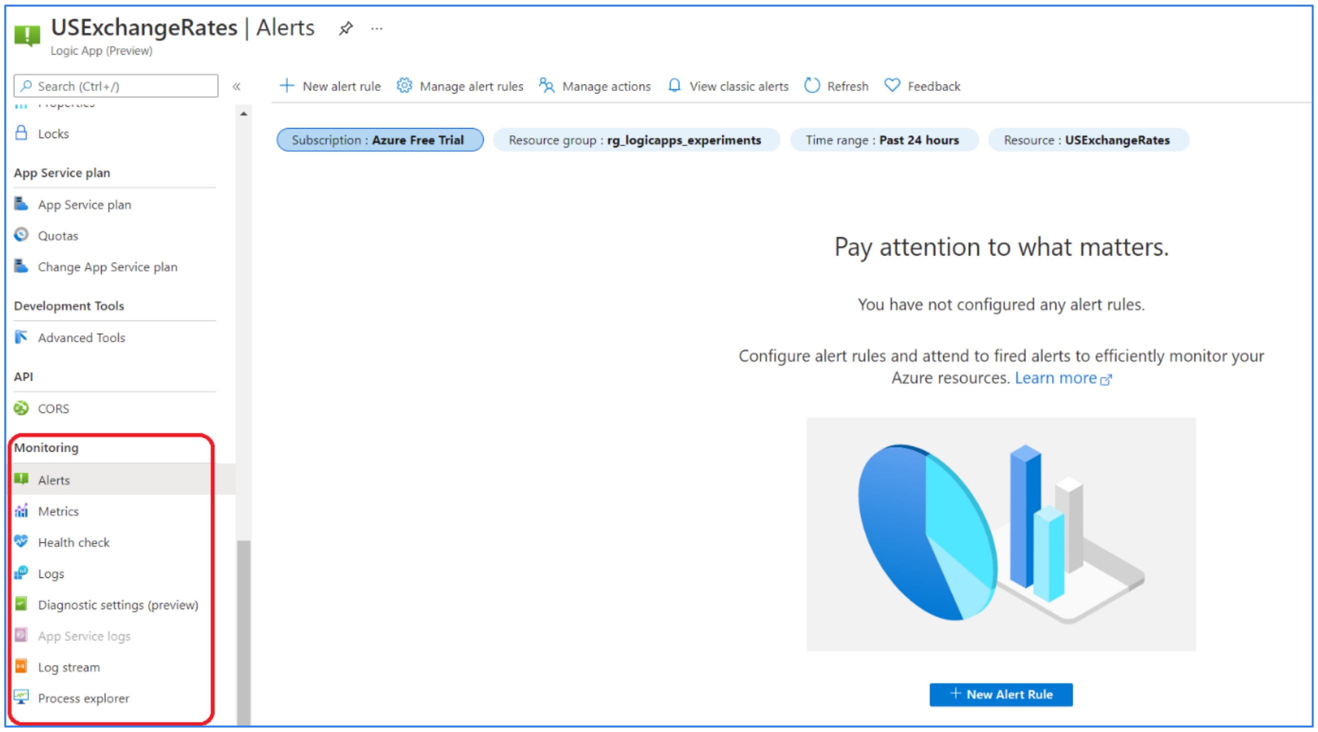
The screenshot shows that you can see the triggered workflows and have the ability to drill down in each of the runs. And this is just one of many other things that Application Insights has to offer. For instance, with the application map feature, you can see the call duration, the number of calls, and dependencies with other components (in case there are any).



The integration with Application Insights with Logic Apps is a significant improvement and enhances the observability of your integration solutions. To learn more about Application Insights, see the Microsoft DevOps Labs and Azure Monitoring Landing page.

6.1
Other Monitoring capabilities

Under the monitoring part on the left-hand side of the portal showing your Logic App, you can see various options:



Pay attention to what matters.

You have not configured any alert rules.

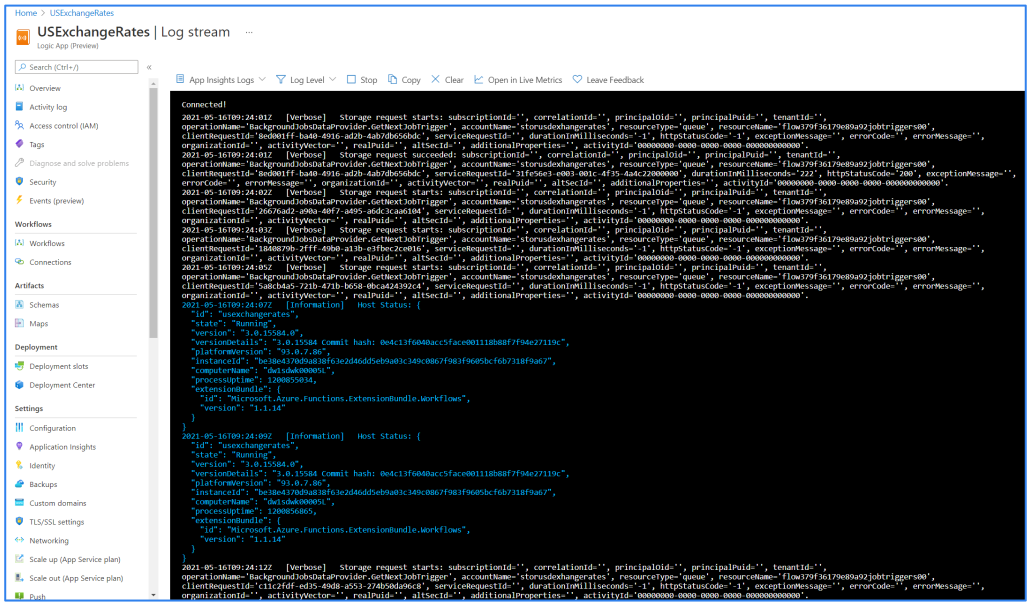
Configure alert rules and attend to fired alerts to efficiently monitor your Azure resources. Learn more

You are familiar with some of them as they also appear when creating Logic Apps (consumption-based – multi-tenant). New or now available are Log Stream and Process Explorer. Let's explore those. Note that App Service logs are greyed out and not available.

6.2

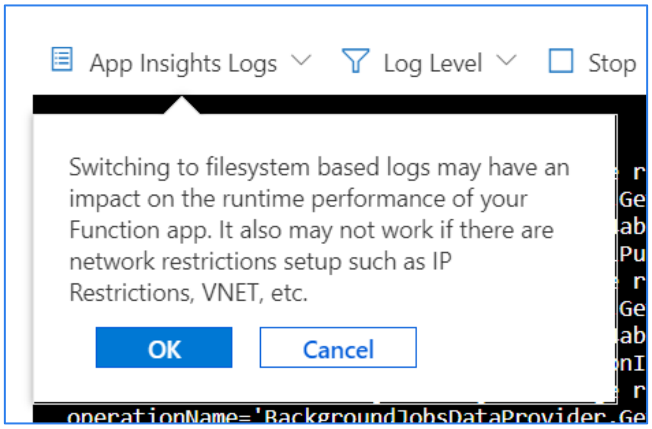
Log Stream

To access the **Log Stream**, you can navigate to Logic App (Standard), and under **Monitoring**, select it in the Azure Portal. The feature allows you to connect to the App Insights log stream or filesystem-based log stream, set the log level (*verbose*, *informational*, *error*, and *warning*), choose to stop/start, and switch to **Live Metrics**.



```
Connected!
2021-05-16T09:24:01Z   [Verbose]   Storage request starts: subscriptionId='', correlationId='', principalOid='', principalPuid='', tenantId='',
operationName='BackgroundJobsDataProvider.GetNextJobTrigger', accountName='storusdexhangerates', resourceType='queue', resourceName='flow379f36179e89a92jobtriggers00',
clientRequestId='8ed001ff-ba40-4916-ad2b-4ab7db656bdc', serviceRequestId='', durationInMilliseconds='-1', httpStatusCode='-1', exceptionMessage='', errorCode='', errorMessage='',
organizationId='', activityVector='', realPuid='', altSecId='', additionalProperties='', activityId='00000000-0000-0000-0000-000000000000'.
2021-05-16T09:24:01Z   [Verbose]   Storage request succeeded: subscriptionId='', correlationId='', principalOid='', principalPuid='', tenantId='',
operationName='BackgroundJobsDataProvider.GetNextJobTrigger', accountName='storusdexhangerates', resourceType='queue', resourceName='flow379f36179e89a92jobtriggers00',
clientRequestId='8ed001ff-ba40-4916-ad2b-4ab7db656bdc', serviceRequestId='31fe56e3-e003-001c-4f35-4a4c22000000', durationInMilliseconds='222', httpStatusCode='200', exceptionMessage='',
errorCode='', errorMessage='', organizationId='', activityVector='', realPuid='', altSecId='', additionalProperties='', activityId='00000000-0000-0000-0000-000000000000'.
2021-05-16T09:24:02Z   [Verbose]   Storage request starts: subscriptionId='', correlationId='', principalOid='', principalPuid='', tenantId='',
operationName='BackgroundJobsDataProvider.GetNextJobTrigger', accountName='storusdexhangerates', resourceType='queue', resourceName='flow379f36179e89a92jobtriggers00',
clientRequestId='26676ad2-a90a-40f7-a495-a6dc3caa6104', serviceRequestId='', durationInMilliseconds='-1', httpStatusCode='-1', exceptionMessage='', errorCode='', errorMessage='',
organizationId='', activityVector='', realPuid='', altSecId='', additionalProperties='', activityId='00000000-0000-0000-0000-000000000000'.
2021-05-16T09:24:03Z   [Verbose]   Storage request starts: subscriptionId='', correlationId='', principalOid='', principalPuid='', tenantId='',
operationName='BackgroundJobsDataProvider.GetNextJobTrigger', accountName='storusdexhangerates', resourceType='queue', resourceName='flow379f36179e89a92jobtriggers00',
clientRequestId='1840879b-2fff-49b0-a13b-e3fbec2ce016', serviceRequestId='', durationInMilliseconds='-1', httpStatusCode='-1', exceptionMessage='', errorCode='', errorMessage='',
organizationId='', activityVector='', realPuid='', altSecId='', additionalProperties='', activityId='00000000-0000-0000-0000-000000000000'.
2021-05-16T09:24:05Z   [Verbose]   Storage request starts: subscriptionId='', correlationId='', principalOid='', principalPuid='', tenantId='',
operationName='BackgroundJobsDataProvider.GetNextJobTrigger', accountName='storusdexhangerates', resourceType='queue', resourceName='flow379f36179e89a92jobtriggers00',
clientRequestId='5a8cb4a5-721b-471b-b658-0bca424392c4', serviceRequestId='', durationInMilliseconds='-1', httpStatusCode='-1', exceptionMessage='', errorCode='', errorMessage='',
organizationId='', activityVector='', realPuid='', altSecId='', additionalProperties='', activityId='00000000-0000-0000-0000-000000000000'.
2021-05-16T09:24:07Z   [Information]   Host Status: {
  "id": "usexchangerates",
  "state": "Running",
  "version": "3.0.15584.0",
  "versionDetails": "3.0.15584 Commit hash: 0e4c13f6040acc5face001118b88f7f94e27119c",
  "platformVersion": "93.0.7.86",
  "instanceId": "be38e4370d9a838f63e2d46dd5eb9a03c349c0867f983f9605bcf6b7318f9a67",
  "computerName": "dw1sdwk00005L",
  "processUptime": 1200855034,
  "extensionBundle": {
    "id": "Microsoft.Azure.Functions.ExtensionBundle.Workflows",
    "version": "1.1.14"
  }
}
2021-05-16T09:24:09Z   [Information]   Host Status: {
  "id": "usexchangerates",
  "state": "Running",
  "version": "3.0.15584.0",
  "versionDetails": "3.0.15584 Commit hash: 0e4c13f6040acc5face001118b88f7f94e27119c",
  "platformVersion": "93.0.7.86",
  "instanceId": "be38e4370d9a838f63e2d46dd5eb9a03c349c0867f983f9605bcf6b7318f9a67",
  "computerName": "dw1sdwk00005L",
  "processUptime": 1200856865,
  "extensionBundle": {
    "id": "Microsoft.Azure.Functions.ExtensionBundle.Workflows",
    "version": "1.1.14"
  }
}
2021-05-16T09:24:12Z   [Verbose]   Storage request starts: subscriptionId='', correlationId='', principalOid='', principalPuid='', tenantId='',
operationName='BackgroundJobsDataProvider.GetNextJobTrigger', accountName='storusdexhangerates', resourceType='queue', resourceName='flow379f36179e89a92jobtriggers00',
clientRequestId='c11c2fdf-ed35-49d8-a553-274b50da96c8', serviceRequestId='', durationInMilliseconds='-1', httpStatusCode='-1', exceptionMessage='', errorCode='', errorMessage='',
organizationId='', activityVector='', realPuid='', altSecId='', additionalProperties='', activityId='00000000-0000-0000-0000-000000000000'.
```

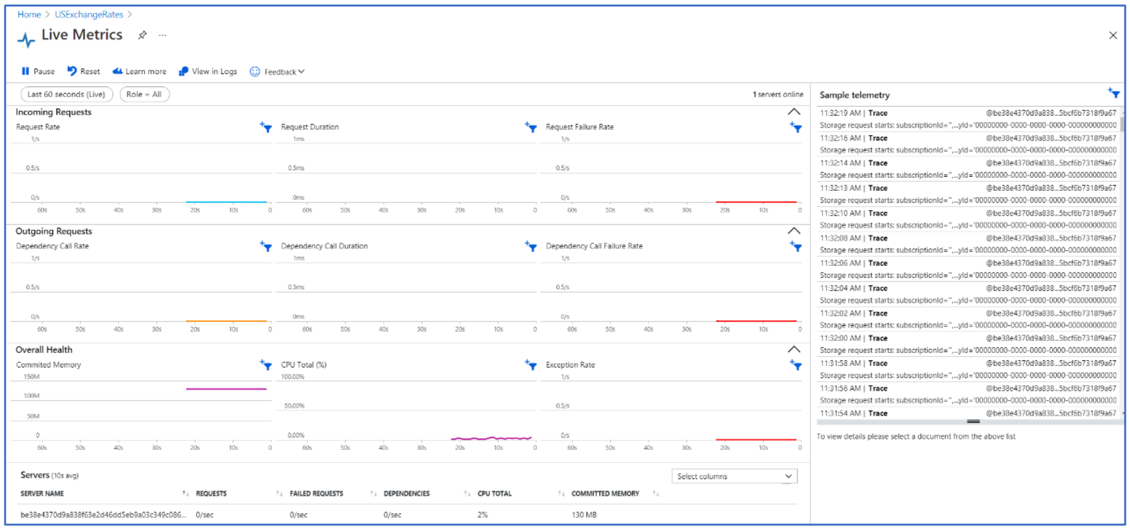
Note that you will receive a warning that it can impact performance when you toggle to the filesystem-based log stream.



Switching to filesystem based logs may have an impact on the runtime performance of your Function app. It also may not work if there are network restrictions setup such as IP Restrictions, VNET, etc.

```
operationName='BackgroundJobsDataProvider.Ge
```

With Log Stream, you can look at the actual log stream of your Logic App runtime – useful when to troubleshoot issues. Also, you open Live Metrics simultaneously to get further insights.



```
11:32:19 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:16 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:14 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:13 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:10 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:08 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:06 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:04 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:02 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:32:00 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:31:58 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:31:56 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
Storage request starts: subscriptionId='',...yId='00000000-0000-0000-0000-000000000000
11:31:54 AM | Trace    @be38e4370d9a838...5bcf6b7318f9a67
```

To view details please select a document from the above list

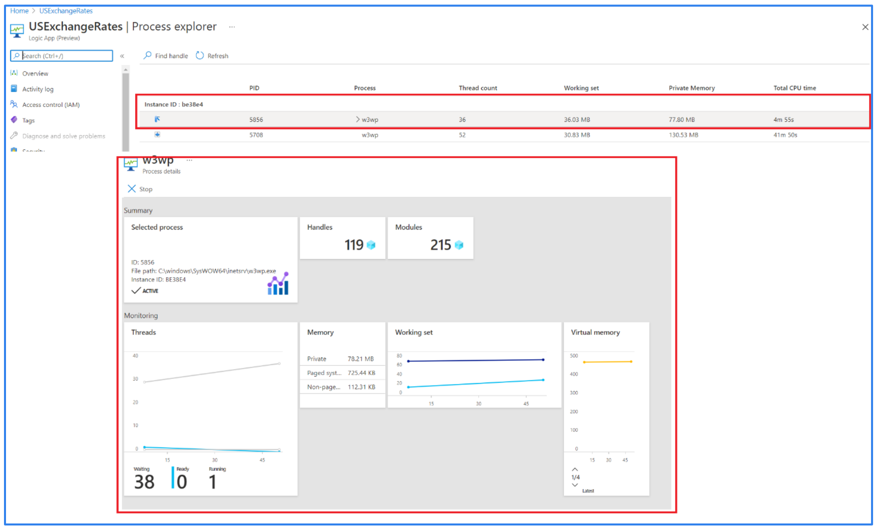
Servers (10s avg)

SERVER NAME	REQUESTS	FAILED REQUESTS	DEPENDENCIES	CPU TOTAL	COMMITTED MEMORY
be38e4370d9a838f63e2d46dd5eb9a03c349c086...	0/sec	0/sec	0/sec	2%	130 MB

6.3

Process Explorer

In the previous paragraph, we discussed the Log Stream feature. The other feature is process explorer, which provides information about the running processes (w3wp) in the runtime. You can examine each process by clicking on it. You will then get access to a new page providing you with an overview of memory consumption, threads, and working sets.

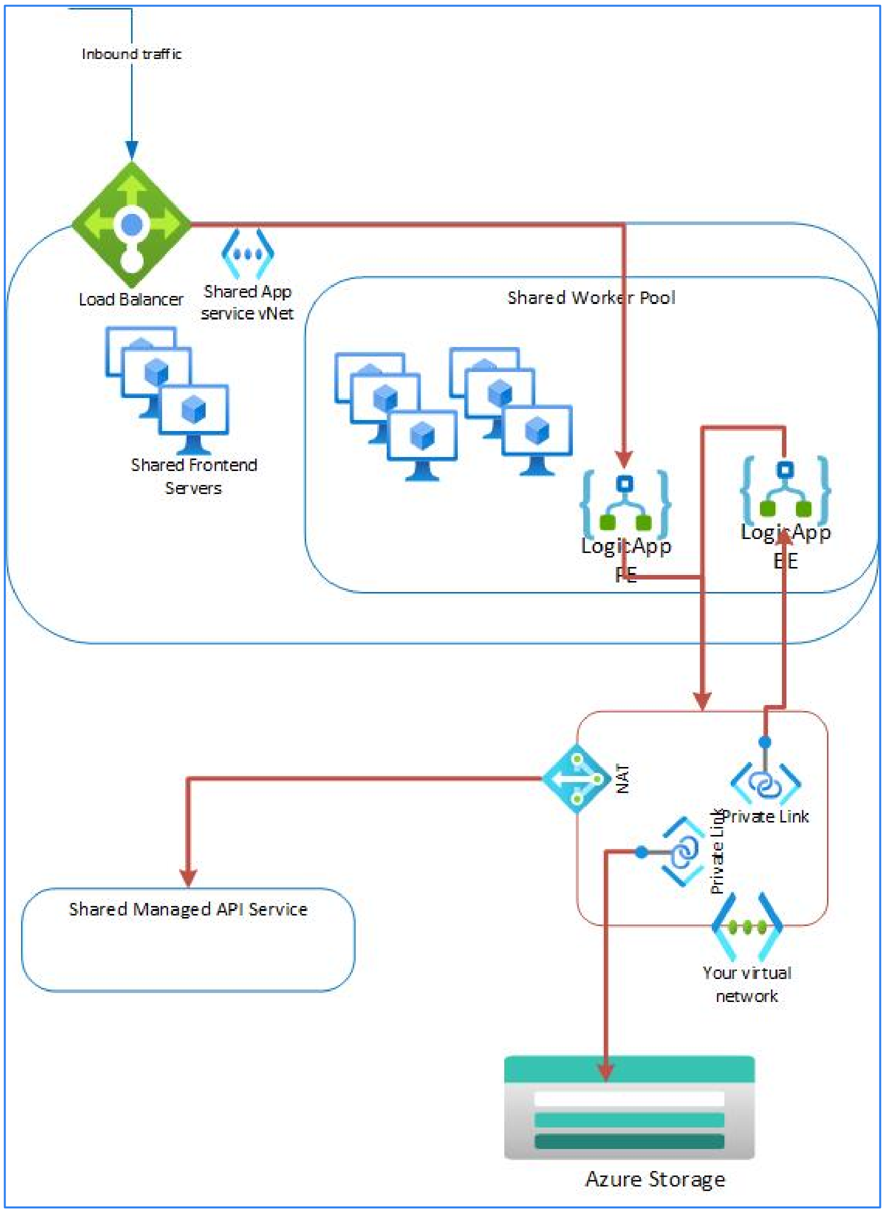


PID	Process	Thread count	Working set	Private Memory	Total CPU time
Instance ID : be38e4					
5856	w3wp	36	36.03 MB	77.80 MB	4m 55s
5708	w3wp	52	30.83 MB	130.53 MB	41m 50s

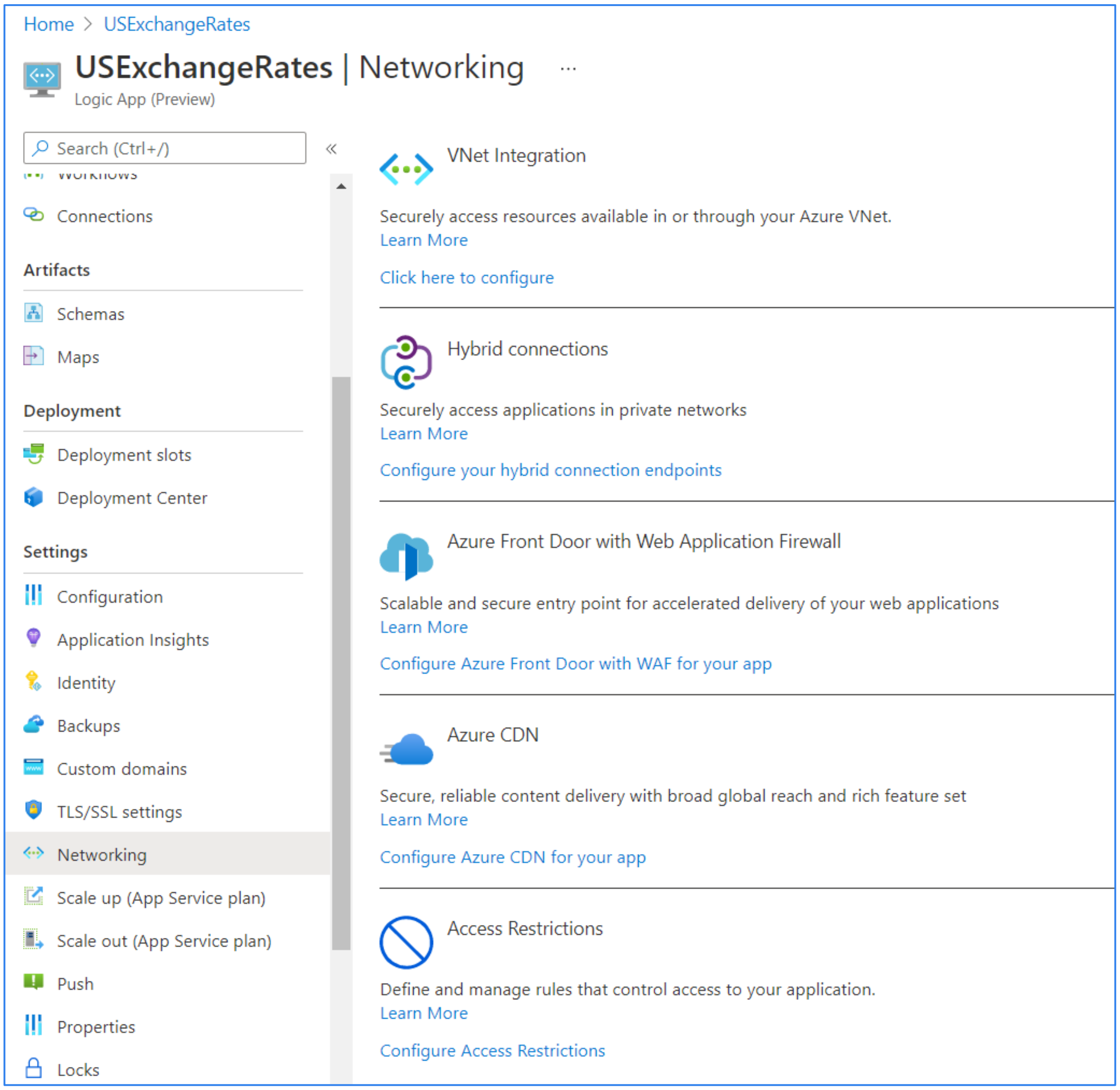
7

Networking and Security

The Logic App Standard runs on top of the Function App runtime – basically on top of the App Service Infrastructure and thus inherits the platform capabilities it offers. The tech community blog post Networking Possibilities with Logic App Preview provides examples of securing inbound traffic to HTTP-triggered Logic Apps with Private Endpoints and VNET integration.



Furthermore, it references a GitHub page containing arm templates and application code for setting two workflows in two Logic App Preview resources in a virtual network with the storage accounts locked into the VNET. Note that for VNET Integration, the Logic App requires a standard or premium plan. Under **Settings** in the Portal, you can access the **Networking** features for Logic App (Standard)



Home > USExchangeRates

USExchangeRates | Networking

Logic App (Preview)

Search (Ctrl+/)

Workflows
Connections

Artifacts
Schemas
Maps

Deployment
Deployment slots
Deployment Center

Settings
Configuration
Application Insights
Identity
Backups
Custom domains
TLS/SSL settings
Networking
Scale up (App Service plan)
Scale out (App Service plan)
Push
Properties
Locks

VNet Integration

Securely access resources available in or through your Azure VNet.
Learn More

Click here to configure

Hybrid connections

Securely access applications in private networks
Learn More

Configure your hybrid connection endpoints

Azure Front Door with Web Application Firewall

Scalable and secure entry point for accelerated delivery of your web applications
Learn More

Configure Azure Front Door with WAF for your app

Azure CDN

Secure, reliable content delivery with broad global reach and rich feature set
Learn More

Configure Azure CDN for your app

Access Restrictions

Define and manage rules that control access to your application.
Learn More

Configure Access Restrictions

As shown above, you can configure VNET integration, hybrid connections, and so on from this point on.

Other security aspects of Logic Apps Standard are that you can access **Security** in the Portal on the left hand near **Overview** and **Access Control**. This feature provides recommendations and alerts. The **Access Control** provides control over who can access the resource and **Identity** under **Settings,** where you can configure managed identities (system- or user-defined).

The single-tenant Logic App (Standard) has many more capabilities for networking and security – which shouldn't surprise you when the runtime inherits all the App Service infrastructure. Moreover, consider this aspect when looking at the three options available for Logic Apps (more considerations are discussed in the following paragraphs). Examine the Microsoft documentation for the networking and security aspects (look at App Service specifically) – as it is pretty extensive and warrants a paper on its own.

Lastly, you can also read the Microsoft document Secure traffic between virtual networks and single-tenant workflows in Azure Logic Apps using private endpoints.

8
Considerations

With a new version of Logic Apps, you will have three available options to host your workflows in Azure. Each with its characteristics and also limitations. This paragraph will discuss a few aspects you need to consider with these options: pricing, scenarios, training, deployment strategy, and those limitations.

8.1
Pricing

First, let us look at pricing, which differs for each option. The multi-tenant pay-as-you-go version pricing is based upon executions (consumption). You pay for each trigger action, action, and usage of connectors. Below is an example of Pricing in U.S. Dollars as per the pricing page:

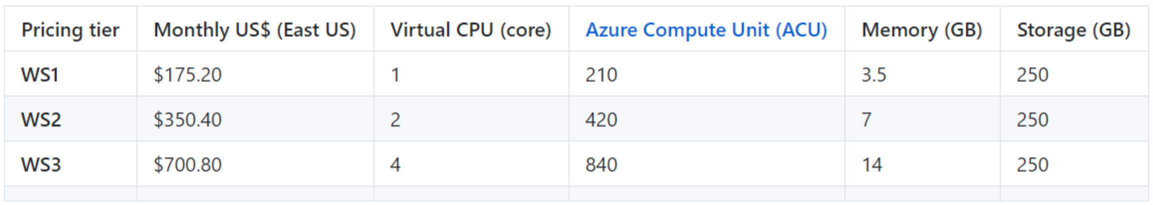
	Per execution
Actions	$0.000025
Standard Connectors	$0.000125
Enterprise Connectors	$0.001

Furthermore, when you require mapping support in your Logic App and want to leverage the integration account, you will incur additional costs. Pricing of the integration account is available on the same pricing page. You choose between basic ($0.42/hour) and standard ($1.37/hour) plan.

Second, you have the option of choosing an integrated service environment (ISE) to support a more isolated scenario (VNET support, scaling, and predictable costs). With ISE, the pricing is different – it's hourly costs and not consumption-based. Below an example of Pricing in U.S. Dollars as per the pricing page:

	Developer	**Premium**
Base Unit	$1.03 Hour	$6.64 Hour
Scale Unit	N.A.	$3.32 Hour

And lastly, we have the new Logic App (Standard) running in Function App runtime. The pricing details are available through the Pricing and billing models for the Azure Logic Apps GitHub page:



Pricing tier	Monthly US$ (East US)	Virtual CPU (core)	Azure Compute Unit (ACU)	Memory (GB)	Storage (GB)
WS1	$175.20	1	210	3.5	250
WS2	$350.40	2	420	7	250
WS3	$700.80	4	840	14	250

Furthermore, when leveraging Application Insights (part of Azure Monitor), you also will incur charges. To conclude for pricing of the new Logic App, you will have to examine the costs for:

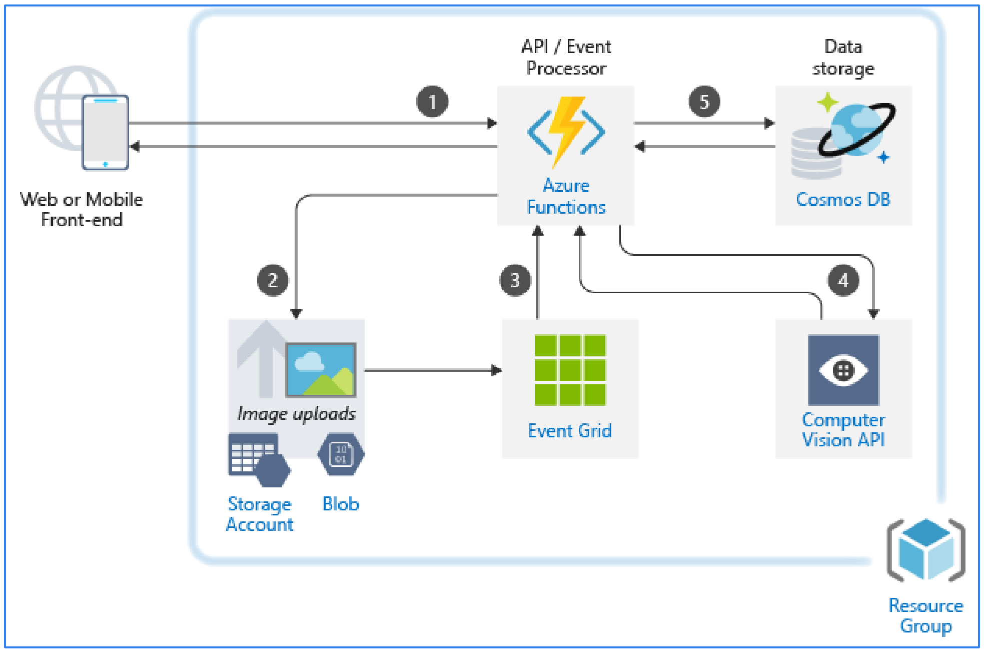
- App Service Pricing details;
- Azure Storage Pricing details;
- And monitoring pricing.

We suggest you also read the Pricing and billing models for Azure Logic Apps Microsoft document for a good understanding of the pricing and the Estimate storage costs for workflows in the single-tenant Azure Logic Apps document.

Pricing is a critical consideration for any Azure service you want to leverage for your solution. This consideration is also applicable when using Logic Apps for your integration needs. Next to pricing (which is tied to hosting), relevant scenarios for using Logic App are essential.

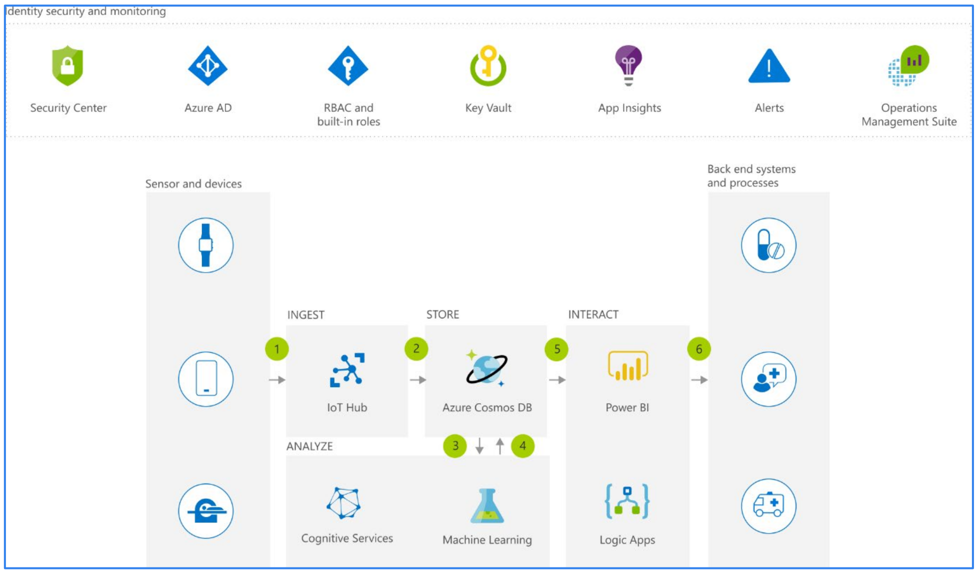
8.2 Scenarios

Logic App is an Azure service, which you leverage in various cloud solution architectures. An obvious one is when integrating systems in the Cloud and/or on-premise (hybrid). Yet, you can use Logic Apps in other scenarios too. The Azure Architecture center provides many strategies where you can leverage a Logic App. For instance, the Image Classification Example. While the scenario discusses the use of Functions, as shown below, it does mention using Logic Apps when you don't need to react in real-time on added files to a blob. A logic app that can check if a file was added might be started by the recurrence trigger or sliding windows trigger.



Logic App has a recurrence trigger action that has several configurable options. Although the function does provide a trigger binding, you could also choose a function – yet you could select a Logic App to support another requirement.

Another example of the Azure Architecture Center is a Remote Patient Monitoring Solution (see diagram below).



The workflow according to the scenario is as follows:

- Securely ingest medical sensor and device data using Azure IoT Hub.
- Securely store sensor and device data in Cosmos DB.
- Analyze sensor and device data using a pre-trained Cognitive Services API or a custom-developed Machine Learning model.
- Store Artificial Intelligence (AI) and Machine Learning result in Cosmos DB.
- Interact AI and Machine Learning results using Power BI while preserving Azure role-based access control (Azure RBAC).
- Integrate data insights with backend systems and processes using **Logic Apps**.

Logic Apps are a part of the solution for integration purposes. With the scenario in mind, you can think of the workload the Logic Apps will have to deal with – which hosting option would be best in this case, i.e., multi-, single-tenant version of Logic Apps or an integrated service environment. Since we are dealing with patient data, compliance and security are critical – enterprise capabilities such as VNET integration will become apparent. Therefore, the multi-tenant Logic App is the least interesting option – leaving the single-tenant or ISE option open.

When looking at your requirements and starting a design for your solution that will include Logic Apps, it is essential to look at:

- Workload (hosting)
- Patterns (look at similar solutions in the architecture center)
- Non-functionals (scale, networking, security)

8.3 Training

In the previous two sections, we discussed pricing and scenarios. You can now understand that with various options of Logic Apps, you will need a good grasp of the technology and characteristics. Fortunately, the Azure Architecture Center is a good source of information and detail. Moreover, this also accounts for the Microsoft documentation. And there is more with Microsoft Learn offering a learning experience.

The following lists a set of resources that can support you in getting yourself trained and well-versed in Logic Apps:

- Microsoft Learn – Intro into Logic Apps
- Microsoft Learn – Route and Process data automatically with Logic Apps
- On-demand – Pluralsight
- Documentation Landing Page

- Serverless Tips – Logic Apps
- Azure Architecture Center – Logic Apps
- GitHub – Enterprise Integration

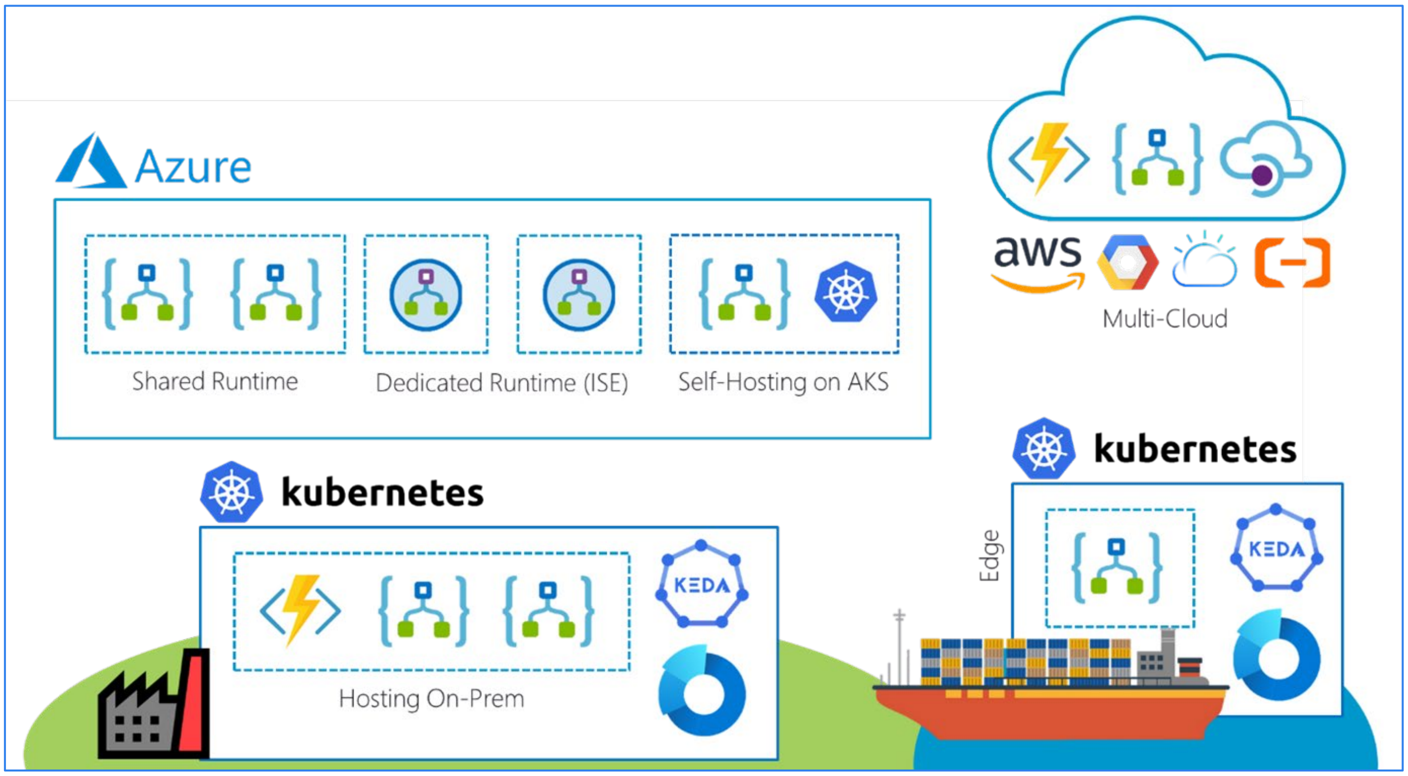
8.4 Deployment Strategies

An essential aspect of building workflows using the new Logic App (Standard) is deployment. First, you need to think about branching strategies, pipelines (deployment of the infrastructure and workflows), and sizing of the environment (App Service Plan). Next, you have several options like deployment center, GitHub Actions, Azure DevOps Pipelines, and Bicep.

You can think of leveraging the Bicep language to deploy your infrastructure (Logic App and its dependencies – Storage Account, Application Insights, and App Service Plan) and separately the workflows. And although Bicep is still preview and early days – the single-tenant Logic Apps is just out; thus, it's worth the effort. Lastly, it would be best to think about workloads for your workflows hosted in the Logic App. Also, think about if the workflows combined have a purpose for a single business process (integration).

With deployment, think about:

- Sizing Logic App
- Single responsibility, i.e., have one or more workflows supporting a single process or integration
- The Logic App and its workflows should be in a single project in Visual Studio
- Location of your Logic App – it can be containerized and thus deployed on the Edge or other platforms – see also Tom Kerkhove's blog about running Logic Apps everywhere!

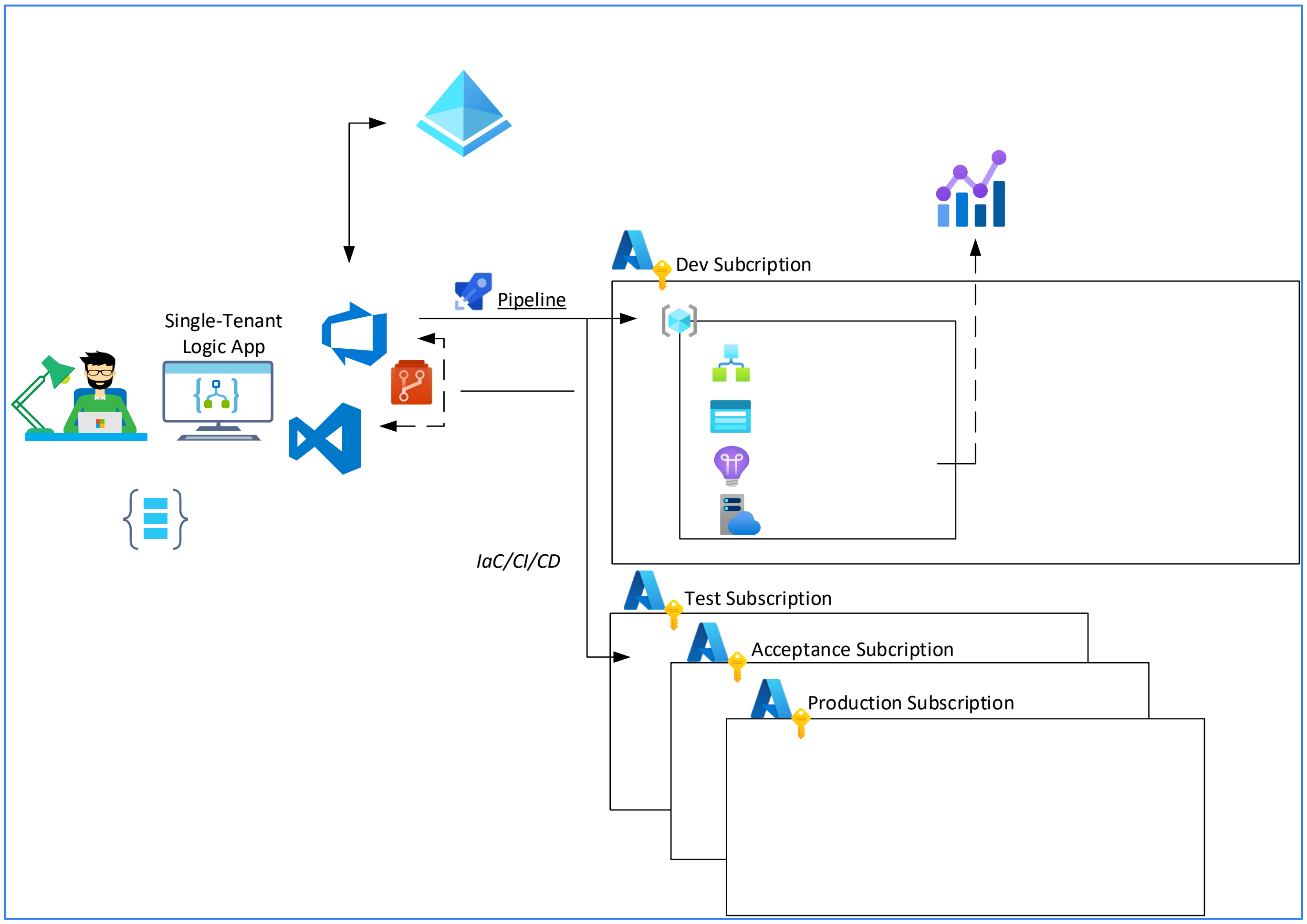


8.4.1
Limitations

Lastly, the limitation of each Logic App option is crucial to understand when you implement them into your solution. Fortunately, the Microsoft documentation is up to date with the limitations of each option. The Limits and configuration information for Azure Logic Apps documentation page lists them all. Furthermore, some known issues with the single-tenant Logic App version (preview) are available on GitHub.

9
Summary

In this eBook, we discussed the new Logic App version – the single-tenant Standard (previously in preview). This Logic App version allows you to build so-called single-tenant workflows hosted in an App Service Plan or container. We discussed the stateless and stateful workflows, the mapping capabilities, networking, security, deployment, monitoring, and considerations. Most of the topics in the eBook are summarized in the diagram below – an example of building workflow(s) locally and push them to several environments.



I hope the eBook will provide you with a starting point to learn this new technology and apply it to your solutions.

About Serverless360

Serverless360 is a one platform tool to operate, manage and monitor Azure Serverless components. It provides efficient tooling that is not and likely to be not available in Azure Portal. Manageability is one of the key challenges with Serverless implementations. Hundreds of small, discrete Serverless functionalities are catered in various places – managing and operating such solutions is complex. Serverless360 solves these challenges with a rich set of sophisticated tools.

Start your 15 days free trial

TRY IT FOR FREE

Visit Serverless360 for more information.

Kovai.co

SERVERLESS360