# ngle-tenant versus multi-tenant and egration service environment for Azure gic Apps

/2021 • 19 minutes to read •

s article



Logic Apps is a cloud-based platform for creating and running automated *logic app lows* that integrate your apps, data, services, and systems. With this platform, you can ly develop highly scalable integration solutions for your enterprise and business-to-ess (B2B) scenarios. To create a logic app, you use either the **Logic App sumption)** resource type or the **Logic App (Standard)** resource type. The umption resource type runs in the *multi-tenant* Azure Logic Apps or *integration service onment*, while the Standard resource type runs in *single-tenant* Azure Logic Apps onment.

e you choose which resource type to use, review this article to learn how the rces types and service environments compare to each other. You can then decide type is best to use, based on your scenario's needs, solution requirements, and the onment where you want to deploy, host, and run your workflows.

're new to Azure Logic Apps, review the following documentation:

What is Azure Logic Apps?
What is a *logic app workflow*?

## source types and environments

eate logic app workflows, you choose the **Logic App** resource type based on your rio, solution requirements, the capabilities that you want, and the environment where vant to run your workflows.

ollowing table briefly summarizes differences between the **Logic App (Standard)** rce type and the **Logic App (Consumption)** resource type. You'll also learn how the *-tenant* environment compares to the *multi-tenant* and *integration service onment (ISE)* for deploying, hosting, and running your logic app workflows.

| ource type | Benefits | Resource sharing and usage | Pricing and billing model | Limits management |
|---|---|---|---|---|
| ic App nsumption)<br><br>t ironment: lti-tenant | - Easiest to get started<br><br>- Pay-for-what-you-use<br><br>- Fully managed | A single logic app can have *only one* workflow.<br><br>Logic apps created by customers *across multiple tenants* share the same processing (compute), storage, network, and so on. | Consumption (pay-per-use) | Azure Logic Apps manages the default values for these limits, but you can change some of these values, if that option exists for a specific limit. |
| ic App ndard)<br><br>t ironment: gle-tenant | - Run anywhere that Azure Functions can run using the containerized single-tenant Azure Logic Apps runtime. Deployment | A single logic app can have multiple *stateful* and *stateless* workflows.<br><br>Workflows *in a single logic app and tenant* share the | Standard, based on a hosting plan with a selected pricing tier<br><br>If you run *stateful* workflows, which use external storage, the | You can change the default values for many limits, based on your scenario's needs.<br><br>**Important**: Some limits have hard upper maximums. In Visual Studio Code, the changes you make to the default limit values in your logic app project configuration |

| ource type | Benefits | | | |
|---|---|---|---|---|
| | slots are currently not supported.<br><br>- More built-in connectors for higher throughput and lower costs at scale<br><br>- More control and fine-tuning capability around runtime and performance settings<br><br>- Integrated support for virtual networks and private endpoints.<br><br>- Create your own built-in connectors. | same processing (compute), storage, network, and so on. | Azure Logic Apps runtime makes storage transactions that follow Azure Storage pricing . | files won't appear in the designer experience. For more information, see Edit app and environment settings for logic apps in single-tenant Azure Logic Apps. |

**Resource sharing and usage**

**Pricing and billing model**

**Limits management**

| ource type | Benefits | Resource sharing and usage | Pricing and billing model | Limits management |
|---|---|---|---|---|
| ic App nsumption)<br><br>t ironment: gration ice ironment ) | - Enterprise scale for large workloads<br><br>- 20+ ISE-specific connectors that connect directly to virtual networks<br><br>- Predictable pricing with included usage and customer-controlled scaling | A single logic app can have *only one* workflow.<br><br>Logic apps *in the same environment* share the same processing (compute), storage, network, and so on. | ISE (fixed) | Azure Logic Apps manages the default values for these limits, but you can change some of these values, if that option exists for a specific limit. |

# gic App (Standard) resource

**ogic App (Standard)** resource type is powered by the redesigned single-tenant Azure Apps runtime. This containerized runtime uses the Azure Functions extensibility el and is hosted as an extension on the Azure Functions runtime. This design provides bility, flexibility, and more performance for your logic app workflows plus other ilities and benefits inherited from the Azure Functions platform and Azure App ce ecosystem.

xample, you can run single-tenant based logic apps and their workflows anywhere Azure function apps and their functions can run. The Standard resource type duces a resource structure that can host multiple workflows, similar to how an Azure ion app can host multiple functions. With a 1-to-many mapping, workflows in the logic app and tenant share compute and processing resources, providing better rmance due to their proximity. This structure differs from the **Logic App**

**sumption)** resource where you have a 1-to-1 mapping between a logic app resource workflow.

arn more about portability, flexibility, and performance improvements, continue with llowing sections. Or, for more information about the single-tenant Azure Logic Apps ne and Azure Functions extensibility, review the following documentation:

Azure Logic Apps Running Anywhere - Runtime Deep Dive
Introduction to Azure Functions
Azure Functions triggers and bindings

## tability and flexibility

you create logic apps using the **Logic App (Standard)** resource type, you can run workflows anywhere that you can run Azure function apps and their functions, not just single-tenant service environment.

xample, when you use Visual Studio Code with the **Azure Logic Apps (Standard)** sion, you can *locally* develop, build, and run your workflows in your development onment without having to deploy to Azure. If your scenario requires containers, you ontainerize and deploy logic apps as containers.

capabilities provide major improvements and substantial benefits compared to the -tenant model, which requires you to develop against an existing running resource in . Also, the multi-tenant model for automating **Logic App (Consumption)** resource yment is completely based on Azure Resource Manager templates (ARM templates), combine and handle resource provisioning for both apps and infrastructure.

the **Logic App (Standard)** resource type, deployment becomes easier because you eparate app deployment from infrastructure deployment. You can package the single- t Azure Logic Apps runtime and workflows together as part of your logic app. You can eneric steps or tasks that build, assemble, and zip your logic app resources into -to-deploy artifacts. To deploy your infrastructure, you can still use ARM templates to ately provision those resources along with other processes and pipelines that you use ose purposes.

ploy your app, copy the artifacts to the host environment and then start your apps to our workflows. Or, integrate your artifacts into deployment pipelines using the tools rocesses that you already know and use. That way, you can deploy using your own n tools, no matter the technology stack that you use for development.

ing standard build and deploy options, you can focus on app development separately infrastructure deployment. As a result, you get a more generic project model where an apply many similar or the same deployment options that you use for a generic You also benefit from a more consistent experience for building deployment pipelines nd your app projects and for running the required tests and validations before shing to production.

## formance

the **Logic App (Standard)** resource type, you can create and run multiple workflows same single logic app and tenant. With this 1-to-many mapping, these workflows resources, such as compute, processing, storage, and network, providing better rmance due to their proximity.

**ogic App (Standard)** resource type and single-tenant Azure Logic Apps runtime de another significant improvement by making the more popular managed ectors available as built-in operations. For example, you can use built-in operations for Service Bus, Azure Event Hubs, SQL, and others. Meanwhile, the managed connector ons are still available and continue to work.

you use the new built-in operations, you create connections called *built-in ections* or *service provider connections*. Their managed connection counterparts are *API connections*, which are created and run separately as Azure resources that you have to then deploy by using ARM templates. Built-in operations and their ections run locally in the same process that runs your workflows. Both are hosted on ngle-tenant Azure Logic Apps runtime. As a result, built-in operations and their ections provide better performance due to proximity with your workflows. This design works well with deployment pipelines because the service provider connections are aged into the same build artifact.

## eate, build, and deploy options

eate a logic app based on the environment that you want, you have multiple options, ample:

**e-tenant environment**

| tion | Resources and tools | More information |
|---|---|---|

| tion | Resources and tools | More information |
|---|---|---|
| re tal | **Logic App (Standard)** resource type | Create integration workflows for single-tenant Logic Apps - Azure portal |
| ual dio de | **Azure Logic Apps (Standard)** extension | Create integration workflows for single-tenant Logic Apps - Visual Studio Code |
| re CLI | Logic Apps Azure CLI extension | Not yet available |

### -tenant environment

| tion | Resources and tools | More information |
|---|---|---|
| re tal | **Logic App (Consumption)** resource type | Quickstart: Create integration workflows in multi-tenant Azure Logic Apps - Azure portal |
| ual dio de | **Azure Logic Apps (Consumption)** extension | Quickstart: Create integration workflows in multi-tenant Azure Logic Apps - Visual Studio Code |
| re CLI | **Logic Apps Azure CLI** extension | - Quickstart: Create and manage integration workflows in multi-tenant Azure Logic Apps - Azure CLI<br><br>- az logic |
| re ource nager | **Create a logic app** Azure Resource Manager (ARM) template | Quickstart: Create and deploy integration workflows in multi-tenant Azure Logic Apps - ARM template |
| re verShell | Az.LogicApp module | Get started with Azure PowerShell |
| re T API | Azure Logic Apps REST API | Get started with Azure REST API reference |

### ration service environment

| tion | Resources and tools | More information |
| --- | --- | --- |
| ıre tal | **Logic App (Consumption)** resource type with an existing ISE resource | Same as Quickstart: Create integration workflows in multi-tenant Azure Logic Apps - Azure portal, but select an ISE, not a multi-tenant region. |

ugh your development experiences differ based on whether you create **Consumption** **ındard** logic app resources, you can find and access all your deployed logic apps r your Azure subscription.

xample, in the Azure portal, the **Logic apps** page shows both **Consumption** and **lard** logic app resource types. In Visual Studio Code, deployed logic apps appear r your Azure subscription, but they are grouped by the extension that you used, ly **Azure: Logic Apps (Consumption)** and **Azure: Logic Apps (Standard)**.

## ıteful and stateless workflows

the **Logic App (Standard)** resource type, you can create these workflow types within ame logic app:

*Stateful*

Create stateful workflows when you need to keep, review, or reference data from previous events. These workflows save the inputs and outputs for each action and their states in external storage, which makes reviewing the run details and history possible after each run finishes. Stateful workflows provide high resiliency if outages happen. After services and systems are restored, you can reconstruct interrupted runs from the saved state and rerun the workflows to completion. Stateful workflows can continue running for much longer than stateless workflows.

*Stateless*

Create stateless workflows 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 this data to external storage. As a result, stateless workflows have shorter runs that are typically no longer than 5 minutes, faster performance with quicker response times, higher throughput, and reduced running costs because the run

details and history aren't kept in external storage. However, if outages happen, interrupted runs aren't automatically restored, so the caller needs to manually resubmit interrupted runs. These workflows can only run synchronously.

For easier debugging, you can enable run history for a stateless workflow, which has some impact on performance, and then disable the run history when you're done. For more information, see Create single-tenant based workflows in Visual Studio Code or Create single-tenant based workflows in the Azure portal.

> **Note**
>
> Stateless workflows currently support only *actions* for **managed connectors**, which are deployed in Azure, and not triggers. To start your workflow, select either the **built-in Request, Event Hubs, or Service Bus trigger**. These triggers run natively in the Azure Logic Apps runtime. For more information about limited, unavailable, or unsupported triggers, actions, and connectors, see **Changed, limited, unavailable, or unsupported capabilities**.

## sted behavior differences between stateful and teless workflows

an make a workflow callable from other workflows that exist in the same **Logic App dard)** resource by using the Request trigger, HTTP Webhook trigger, or managed ector triggers that have the ApiConnectionWebhook type and can receive HTTPS sts.

are the behavior patterns that nested workflows can follow after a parent workflow a child workflow:

Asynchronous polling pattern

The parent doesn't wait for a response to their initial call, but continually checks the child's run history until the child finishes running. By default, stateful workflows follow this pattern, which is ideal for long-running child workflows that might exceed request timeout limits.

Synchronous pattern ("fire and forget")

The child acknowledges the call by immediately returning a `202 ACCEPTED` response, and the parent continues to the next action without waiting for the results from the

child. Instead, the parent receives the results when the child finishes running. Child stateful workflows that don't include a Response action always follow the synchronous pattern. For child stateful workflows, the run history is available for you to review.

To enable this behavior, in the workflow's JSON definition, set the `operationOptions` property to `DisableAsyncPattern`. For more information, see Trigger and action types - Operation options.

Trigger and wait

For a child stateless workflow, the parent waits for a response that returns the results from the child. This pattern works similar to using the built-in HTTP trigger or action to call a child workflow. Child stateless workflows that don't include a Response action immediately return a `202 ACCEPTED` response, but the parent waits for the child to finish before continuing to the next action. These behaviors apply only to child stateless workflows.

able specifies the child workflow's behavior based on whether the parent and child ateful, stateless, or are mixed workflow types:

| ent rkflow | Child workflow | Child behavior |
|---|---|---|
| eful | Stateful | Asynchronous or synchronous with `"operationOptions": "DisableAsyncPattern"` setting |
| eful | Stateless | Trigger and wait |
| eless | Stateful | Synchronous |
| eless | Stateless | Trigger and wait |

# her single-tenant model capabilities

ingle-tenant model and **Logic App (Standard)** resource type include many current ew capabilities, for example:

Create logic apps and their workflows from 400+ managed connectors for Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) apps and services plus

connectors for on-premises systems.

- More managed connectors are now available as built-in operations and run similarly to other built-in operations, such as Azure Functions. Built-in operations run natively on the single-tenant Azure Logic Apps runtime. For example, new built-in operations include Azure Service Bus, Azure Event Hubs, SQL Server, and MQ.

  > **Note**
  >
  > For the built-in SQL Server version, only the **Execute Query** action can directly connect to Azure virtual networks without using the **on-premises data gateway**.

- You can create your own built-in connectors for any service that you need by using the single-tenant Azure Logic Apps extensibility framework . Similarly to built-in operations such as Azure Service Bus and SQL Server but unlike custom managed connectors, which aren't currently supported, custom built-in connectors provide higher throughput, low latency, and local connectivity because they run in the same process as the single-tenant runtime.

  The authoring capability is currently available only in Visual Studio Code, but isn't enabled by default. To create these connectors, switch your project from extension bundle-based (Node.js) to NuGet package-based (.NET). For more information, see Azure Logic Apps Running Anywhere - Built-in connector extensibility .

- You can use the B2B actions for Liquid Operations and XML Operations without an integration account. To use these actions, you need to have Liquid maps, XML maps, or XML schemas that you can upload through the respective actions in the Azure portal or add to your Visual Studio Code project's **Artifacts** folder using the respective **Maps** and **Schemas** folders.

- **Logic app (Standard)** resources can run anywhere because the Azure Logic Apps service generates Shared Access Signature (SAS) connection strings that these logic apps can use for sending requests to the cloud connection runtime endpoint. The Logic Apps service saves these connection strings with other application settings so that you can easily store these values in Azure Key Vault when you deploy in Azure.

  > **Note**

> By default, a **Logic App (Standard)** resource has the **system-assigned managed identity** automatically enabled to authenticate connections at run time. This identity differs from the authentication credentials or connection string that you use when you create a connection. If you disable this identity, connections won't work at run time. To view this setting, on your logic app's menu, under **Settings**, select **Identity**.

Stateless workflows run only in memory so that they finish more quickly, respond faster, have higher throughput, and cost less to run because the run histories and data between actions don't persist in external storage. Optionally, you can enable run history for easier debugging. For more information, see Stateful versus stateless workflows.

You can locally run, test, and debug your logic apps and their workflows in the Visual Studio Code development environment.

Before you run and test your logic app, you can make debugging easier by adding and using breakpoints inside the **workflow.json** file for a workflow. However, breakpoints are supported only for actions at this time, not triggers. For more information, see Create single-tenant based workflows in Visual Studio Code.

Directly publish or deploy logic apps and their workflows from Visual Studio Code to various hosting environments such as Azure and containers.

Enable diagnostics logging and tracing capabilities for your logic app by using Application Insights when supported by your Azure subscription and logic app settings.

Access networking capabilities, such as connect and integrate privately with Azure virtual networks, similar to Azure Functions when you create and deploy your logic apps using the Azure Functions Premium plan. For more information, review the following documentation:

- Azure Functions networking options
- Azure Logic Apps Running Anywhere - Networking possibilities with Azure Logic Apps

Regenerate access keys for managed connections used by individual workflows in a **Logic App (Standard)** resource. For this task, follow the same steps for the **Logic**

**Apps (Consumption)** resource but at the individual workflow level, not the logic app resource level.

## anged, limited, unavailable, or unsupported pabilities

ıe **Logic App (Standard)** resource, these capabilities have changed, or they are ntly limited, unavailable, or unsupported:

**Triggers and actions**: Built-in triggers and actions run natively in the single-tenant Azure Logic Apps runtime, while managed connectors are hosted and run in Azure. Some built-in triggers are unavailable, such as Sliding Window and Batch. To start a stateful or stateless workflow, use the built-in Recurrence, Request, HTTP, HTTP Webhook, Event Hubs, or Service Bus trigger. In the designer, built-in triggers and actions appear under the **Built-in** tab.

For *stateful* workflows, managed connector triggers and actions appear under the **Azure** tab, except for the unavailable operations listed below. For *stateless* workflows, the **Azure** tab doesn't appear when you want to select a trigger. You can select only managed connector *actions*, not triggers. Although you can enable Azure-hosted managed connectors for stateless workflows, the designer doesn't show any managed connector triggers for you to add.

> **Note**
>
> To run locally in Visual Studio Code, webhook-based triggers and actions require additional setup. For more information, see **Create single-tenant based workflows in Visual Studio Code**.

- These triggers and actions have either changed or are currently limited, unsupported, or unavailable:
  - On-premises data gateway *triggers* are unavailable, but gateway actions *are* available.
  - The built-in action, Azure Functions - Choose an Azure function is now **Azure Function Operations - Call an Azure function**. This action currently works only for functions that are created from the **HTTP Trigger** template.

In the Azure portal, you can select an HTTP trigger function that you can access by creating a connection through the user experience. If you inspect the function action's JSON definition in code view or the **workflow.json** file using Visual Studio Code, the action refers to the function by using a `connectionName` reference. This version abstracts the function's information as a connection, which you can find in your logic app project's **connections.json** file, which is available after you create a connection in Visual Studio Code.

> **Note**
>
> In the single-tenant model, the function action supports only query string authentication. Azure Logic Apps gets the default key from the function when making the connection, stores that key in your app's settings, and uses the key for authentication when calling the function.
>
> As in the multi-tenant model, if you renew this key, for example, through the Azure Functions experience in the portal, the function action no longer works due to the invalid key. To fix this problem, you need to recreate the connection to the function that you want to call or update your app's settings with the new key.

- The built-in Inline Code action is renamed **Inline Code Operations**, no longer requires an integration account, and has updated limits.
- The built-in action, Azure Logic Apps - Choose a Logic App workflow is now **Workflow Operations - Invoke a workflow in this workflow app**.
- Some built-in B2B triggers and actions for integration accounts are unavailable, for example, the **Flat File** encoding and decoding actions.
- Custom managed connectors currently aren't currently supported. However, you can create *custom built-in operations* when you use Visual Studio Code. For more information, review Create single-tenant based workflows using Visual Studio Code.

**Breakpoint debugging in Visual Studio Code**: Although you can add and use breakpoints inside the **workflow.json** file for a workflow, breakpoints are supported only for actions at this time, not triggers. For more information, see Create single-tenant based workflows in Visual Studio Code.

**Trigger history and run history**: For the **Logic App (Standard)** resource type, trigger history and run history in the Azure portal appears at the workflow level, not the logic app level. For more information, review Create single-tenant based workflows using the Azure portal.

**Zoom control**: The zoom control is currently unavailable on the designer.

**Deployment targets**: You can't deploy the **Logic App (Standard)** resource type to an integration service environment (ISE) nor to Azure deployment slots.

**Azure API Management**: You currently can't import the **Logic App (Standard)** resource type into Azure API Management. However, you can import the **Logic App (Consumption)** resource type.

## ict network and firewall traffic permissions

ır environment has strict network requirements or firewalls that limit traffic, you have ɔw access for any trigger or action connections in your logic app workflows. To find ılly qualified domain names (FQDNs) for these connections, review the corresponding ɔns in these topics:

Firewall permissions for single tenant logic apps - Visual Studio Code
Firewall permissions for single tenant logic apps - Azure portal

## xt steps

Create single-tenant based workflows in the Azure portal
Create single-tenant based workflows in Visual Studio Code

also like to hear about your experiences with single-tenant Azure Logic Apps!

For bugs or problems, create your issues in GitHub .
For questions, requests, comments, and other feedback, use this feedback form .

## ıis page helpful?

ʏes No

# commended content

### reate workflows with single-tenant Azure Logic Apps (Standard) in the Azure ortal - Azure Logic Apps

eate automated workflows to integrate apps, data, services, and systems with single-tenant zure Logic Apps (Standard) in the Azure portal.

### lanaged connectors for Azure Logic Apps - Azure Logic Apps

se Microsoft-managed triggers and actions to create automated workflows that integrate her apps, data, services, and systems using Azure Logic Apps.

### lling & pricing models - Azure Logic Apps

verview about how pricing and billing models work in Azure Logic Apps

### uilt-in triggers and actions for Azure Logic Apps - Azure Logic Apps

se built-in triggers and actions to create automated workflows that integrate apps, data, rvices, and systems, to control workflows, and to manage data using Azure Logic Apps.

Show more