**Azure App Service – Logic Apps**

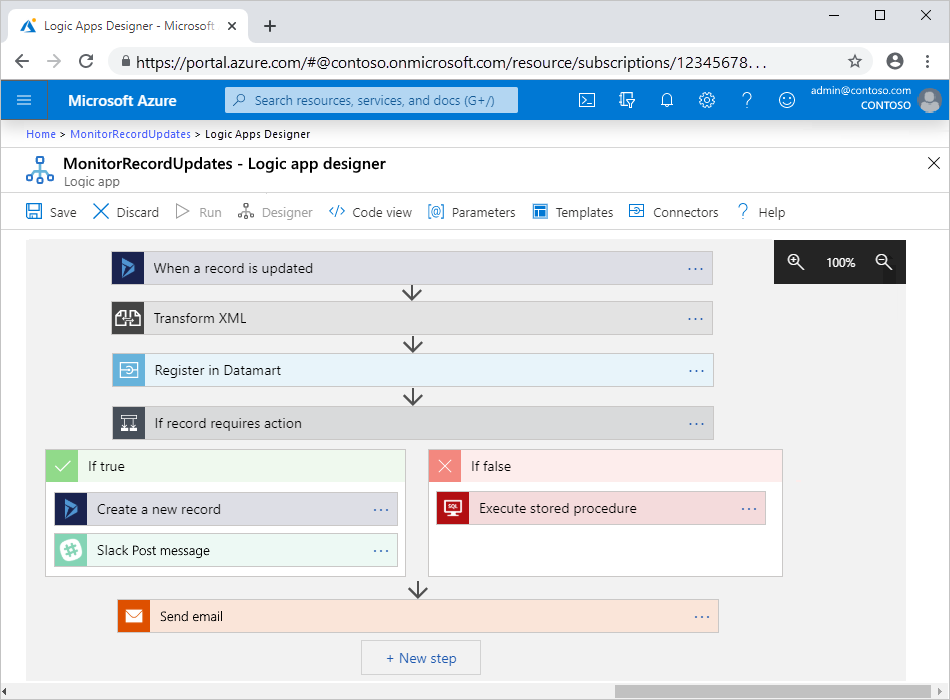
* Introduction
* Triggers and Actions
* Creating a Simple Logic App
* Handling array of Items using for-each and condition
* Logic App with Variables
* Add Custom Function with Azure Function
* Using Custom API App in Logic App
* Azure Functions vs Logic Apps

**Introduction to Logic Apps**

* Logic Apps are used to quickly build powerful solutionsintegrating various SaaS and enterprise applications.
* Logic apps use a **workflow** **engine** and a **visual designer** to design **business processes** graphically, and then connect them through connectors so that users can access data and required services. All this is achieved **without writing a single line of code.**
* The functionality of the out of the box connectors is based on the APIs that can **trigger** new instances of the workflow based on a specific event.
* **Each step** in the workflow is an **action** that accesses data or services through the **connector**.
* Best of all, Logic Apps can be combined with **built-in** **Managed APIs** to help solve even tricky integration scenarios with ease.

**Here are a few examples .**

* Fetch phone numbers of New Contacts added into the CRM system like Sales Force or Dyanimcs, process them and automatically send them Welcome message by SMS.
* We can automatically fetch new records in a SQL DB and then send email alert to users.
* Automatically find negative posts on facebook wall and insert the same to database and delete from facebook.
* Monitor tweets for a specific subject, analyze the sentiment, and create alerts or tasks for items that need review.



**Why Logic Apps?**

* Logic Apps allow developers to design workflows that start from a **trigger** and then execute a series of **steps**. Each step invokes an API while securely taking care of authentication and best practices, like check pointing and durable execution.
* You don't have to worry about **Developing,** **hosting, scaling, managing, maintaining, and monitoring** your apps. Logic Apps handles these concerns for you.
* You pay only for what you use based on a **consumption pricing model**.
* In many cases, you **won't have to write code**. But if you must write some code, you can create code snippets with **Azure Functions** and run that code on-demand from logic apps.

**Advantages:**

* Logic Apps can be designed **end-to-end in the browser** using the design tool provided in Azure portal.
* Logic Apps make it an easy to connect **disparate systems**. Eg: Want to create a task in your CRM software that is based on the activity from your Facebook or Twitter accounts.
* Gallery of **Templates** are provided to rapidly create common solutions.
* Logic Apps is designed to work **with API apps**; you can easily create your own API app to use as a custom API. Build a new app just for you, or share and monetize in the marketplace.
* Logic Apps can easily leverage the power of **BizTalk**, Microsoft's industry leading **integration solution** to enable integration professionals to build the solutions they need.

### **Connectors for Azure Logic Apps**

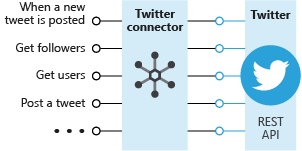
Connectors provide quick access from Azure Logic Apps to **events, data, and actions** across other apps, services, systems, protocols, and platforms.

Connectors are available either as built-in triggers and actions or as managed connectors:

* **Built-ins:** These built-in triggers and actions are “native” to Azure Logic Apps and help you create logic apps that run on custom schedules, communicate with other endpoints, receive and respond to requests, and call Azure functions, Azure API Apps (Web Apps), your own APIs managed and published with Azure API Management, and nested logic apps that can receive requests. You can also use built-in actions that help you organize and control your logic app's workflow, and also work with data.
* **Managed connectors:** These connectors provide **triggers** and **actions** for accessing other services and systems. Some connectors require that you first create connections that are managed by logic apps.

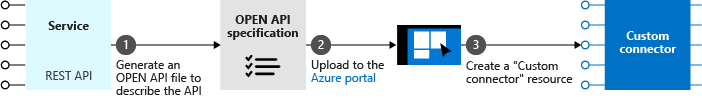
Managed connectors are organized into these groups

* + **Managed API connectors.** This includes Azure Blob Storage, Microsoft 365, Dynamics 365, Power BI, OneDrive, Salesforce, and SharePoint Online.
  + **On-premises connectors.** After you install and set up the on-premises data gateway, these connectors help your logic apps access on-premises systems such as SQL Server, SharePoint Server, Oracle DB, file shares, and others.
  + **Integration account connectors.** Available when you create and pay for an integration account, these connectors transform and validate XML, encode, and decode flat files, and process business-to-business (B2B) messages with AS2, EDIFACT, and X12 protocols.
  + **Enterprise connectors**. Provide access to enterprise systems such as SAP and IBM MQ for an additional cost.



**Custom APIs and connectors:**

To call APIs that run custom code or aren't available as connectors, you can extend the Logic Apps platform by creating custom API Apps. You can also create custom connectors for any REST or SOAP based APIs, which make those APIs available to any logic app in your Azure subscription.

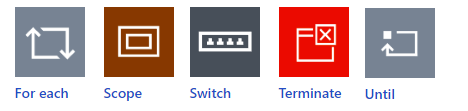


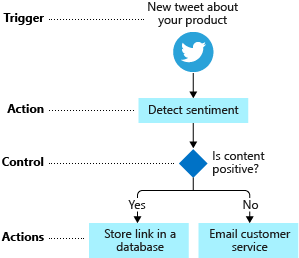
**Triggers and Actions**

* **Triggers** - A trigger starts a new instance of a workflow based on a specific event, like the arrival of an e-mail or a change in your Azure Storage account or a Post on your Facebook wall.

There are two types of triggers.

* **Polling Triggers:** These triggers call your service at a specified frequency to check for new data. When new data is available, the trigger causes a new run of your workflow instance with the data as input.
* **Push Triggers:** These triggers listen for data on an endpoint, that is, they wait for an event to occur. The occurrence of this event causes a new run of your workflow instance.
* **Actions** - Each step after the trigger in a workflow is called an action. Each action typically maps to an operation on your managed or custom API apps. There are built-in actions for structuring and controlling the actions in your logic app's workflow. For example, you could insert a Condition to evaluate a condition and run different actions based on whether the condition is true or false. Other built-in actions are: For each, Scope, Switch, Terminate, and Until.



****

## Creating a Logic App

**Example: Handling Array of Items using ForEach Step.**

**Logic App for Sending Emails to all rows of EmailAlerts table having AlertSent=False**

1. Create Table as below

CREATE TABLE EmailAlerts

(

ID int NOT NULL PRIMARY KEY IDENTITY (1, 1),

ToAddress varchar(50) NOT NULL,

MailSubject varchar(50) NOT NULL,

MailBody varchar(MAX) NOT NULL,

EmailSent bit NOT NULL

)

GO

1. Insert some records into the above table.

INSERT INTO EmailAlerts (ToAddress, MailSubject, MailBody, EmailSent) VALUES ('decnsoft@hotmail.com','Sub1','This is message body - 1', 0)

GO

INSERT INTO EmailAlerts (ToAddress, MailSubject, MailBody, EmailSent) VALUES ('sansoni@gmail.com','Sub2','This is message body - 2', 0)

GO

INSERT INTO dbo.EmailAlerts (ToAddress, MailSubject, MailBody, EmailSent) VALUES ('info@deccansoft.com','Sub3','This is message body - 3', 0)

GO

1. Select New, Web + Mobile, and select Logic App
2. Select Logic App 🡪 Settings 🡪 Triggers and Actions 🡪 Logic App Designer
3. Click + 🡪 Select **Recurrence**, Frequency = Minute, Interval=5
4. Click + 🡪 Add Action 🡪 SQL Get rows,
   1. Provide the SQL Connection Details
   2. Table name = "EmailAlerts",
   3. Click on Show Advanced Options, Filter Query=**EmailSent eq false** (Note: false should be in lowercase)
5. Click + New step 🡪 More 🡪 Add a for each
   1. Select an output from previous steps = Get rows, values
   2. Add an Action 🡪 Search Send Email 🡪 Provide details for sending email
      * From Address = <Email address whose settings are provide in email connector>
      * Set ToAddress, Subject and Body with Outputs from Get rows.
   3. Add an action search SQL – Update row
      * Select Existing Connection / Create a Connection
      * TableName = EmailAlerts
      * ToAddress, MailSubject, MailBody = values from Get row action
      * AlertSent = true
   4. Save and Run the Trigger.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Retry Policy:**

P is the duration designator (for period) placed at the start of the duration representation.

T is the time designator that precedes the time components of the representation.

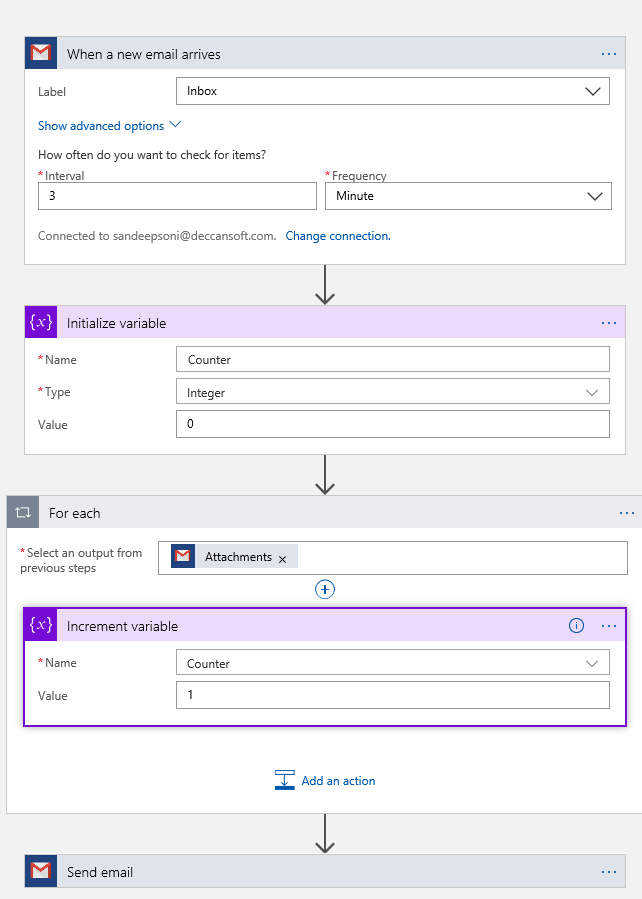
Y is the year designator that follows the value for the number of years.

M is the month designator that follows the value for the number of months.

W is the week designator that follows the value for the number of weeks.

D is the day designator that follows the value for the number of days.

Logic App With Variables and For Each

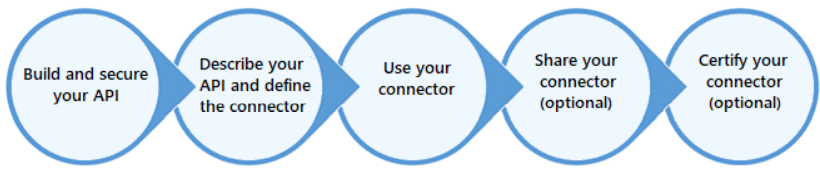


[Build approval-based automated workflows - Azure Logic Apps | Microsoft Docs](https://docs.microsoft.com/en-us/azure/logic-apps/tutorial-process-mailing-list-subscriptions-workflow)

**Custom Connectors in Logic Apps**

While Azure Logic Apps, Microsoft Flow, and PowerApps offer over 180+ connectors to connect to Microsoft and non-Microsoft services, you may want to communicate with services that are not available as prebuilt connectors. Custom connectors address this scenario by allowing you to to create (and even share) a connector with its own triggers and actions.

The following diagram shows the high-level tasks involved in creating and using custom connectors:



1. Create ASP.NET Core Web API Project
2. Add a reference to **Swashbuckle.AspNetCore**
3. **Add the following ConfigureService Method of Startup class.**

services.**AddSwaggerGen**(options =>

{

options.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo() {

Title = "HTTP Web API Demo",

Version = "v1",

Description = "The is a demo of how to use Swagger in Web API",

});

});

1. **Add the following to Configure Method of Startup class**

app.UseSwagger(c =>

{

c.SerializeAsV2 = true;

})

.UseSwaggerUI(c => {

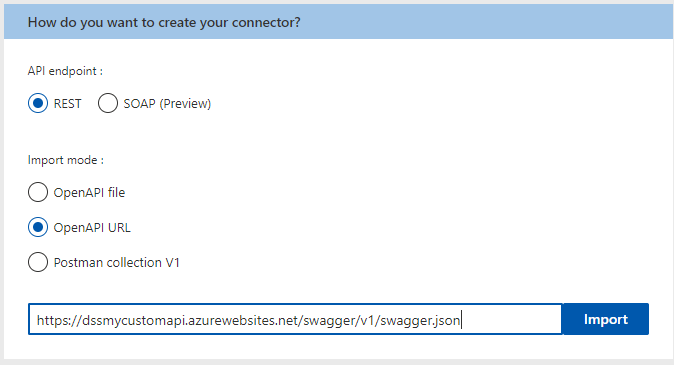
c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");

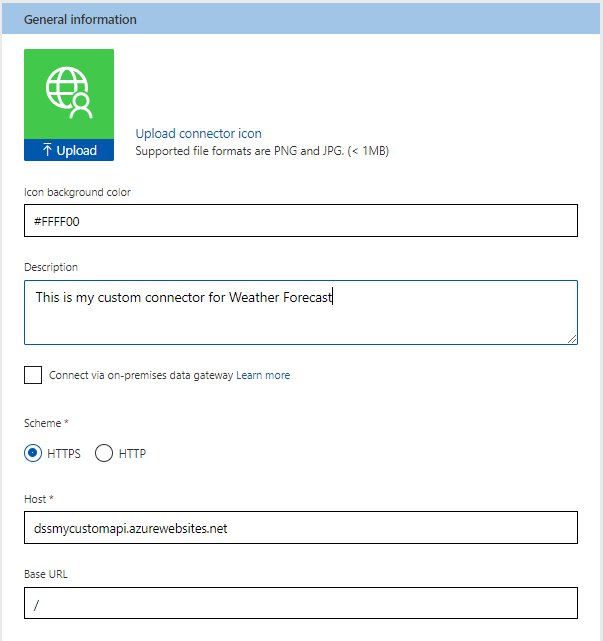
});

1. Publish to App Service and Copy the Swagger URL. Eg: <https://dssmycustomapi.azurewebsites.net/swagger/v1/swagger.json>

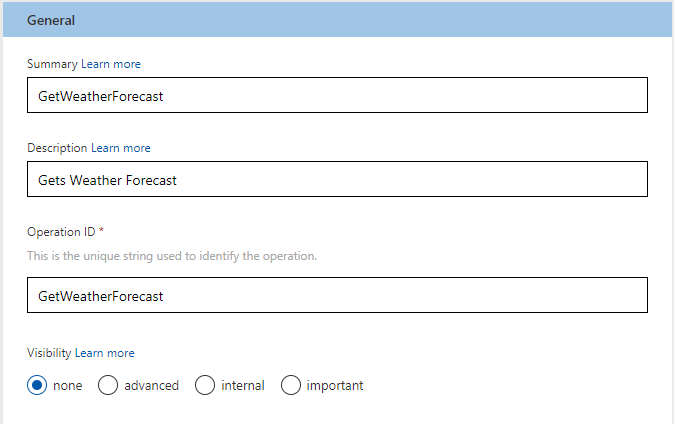
# Azure Portal 🡪 **Create a resource = Logic Apps Custom Connector**

1. Import the OpenAPI definition for Logic Apps (Edit the custom logic app) (<https://docs.microsoft.com/en-us/connectors/custom-connectors/define-openapi-definition>)
2. Step1: General





1. Step2: Security 🡪 (Skip)
2. Step 3: Definition 🡪 +New action



1. Request 🡪 + Import from sample 🡪 Verb = Get, URL = <https://dssmycustomapi.azurewebsites.net/WeatherForecast> 🡪 Import
2. Response 🡪 + Add default response 🡪 Enter body as below 🡪 Import

[

{

"date": "2020-11-25T18:03:11.0835475+00:00",

"temperatureC": -1,

"temperatureF": 31,

"summary": "Hot"

}

]

1. Use the custom connector from a Logic App.
2. Run and view the output.

**Building Logic App In Visual Studio**

[Edit and manage logic apps by using Visual Studio with Cloud Explorer - Azure Logic Apps | Microsoft Docs](https://docs.microsoft.com/en-us/azure/logic-apps/manage-logic-apps-with-visual-studio)

**Check traffic on a schedule with Azure Logic Apps**

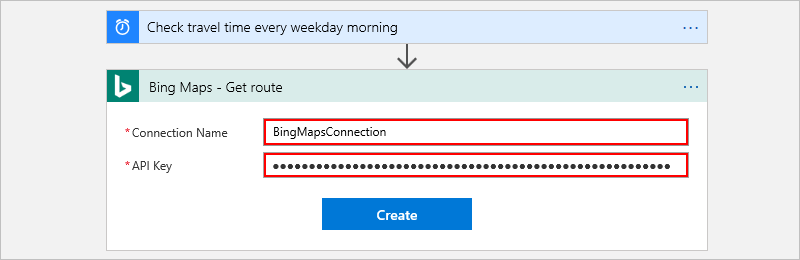
## Step1: Creating a Bing Maps Key

1. Go to the Bing Maps Dev Center at <https://www.bingmapsportal.com/>.
   * If you have a Bing Maps account, sign in with the Microsoft account that you used to create the account or create a new one. For new accounts, follow the instructions in [Creating a Bing Maps Account](https://docs.microsoft.com/en-us/bingmaps/getting-started/bing-maps-dev-center-help/creating-a-bing-maps-account).
2. Select **My keys** under **My Account**.
3. Select the option to create a new key.

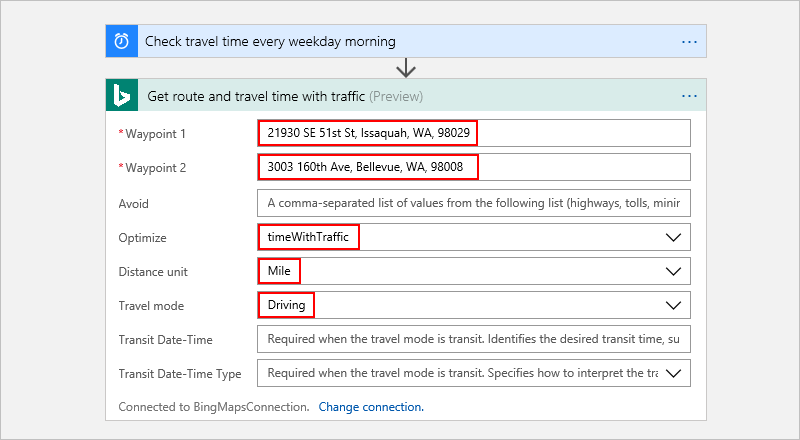
**Sample Key**: ArnSE6qQ6QDVWURsyUKYYgptwXJQ5WlahKNxQCOPUIOE6Bd06ttSZWuoSSSItHaF

**Step 2: Create a Logic App**

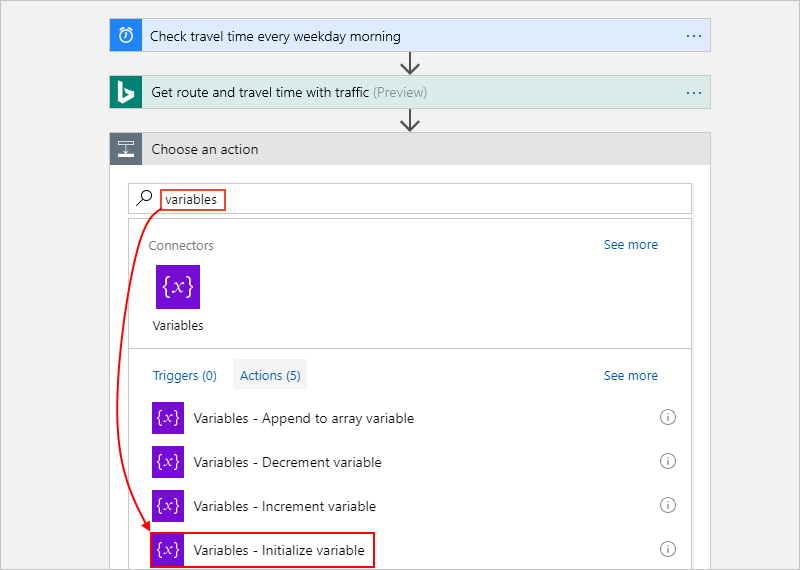
1. Create a blank logic app.
2. Add a recurrence trigger that works as a scheduler for your logic app.
3. Search for "maps", and select this action: **Bing Maps - Get route**



1. Provide details for the **Get route** action as shown and described here, for example:



1. Add an action that creates a variable, converts the travel time from seconds to minutes, and saves that result in the variable.

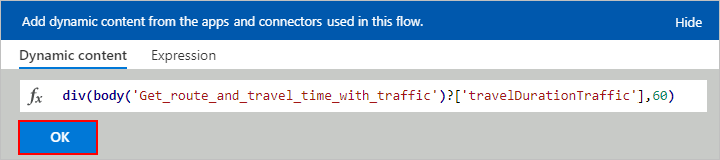
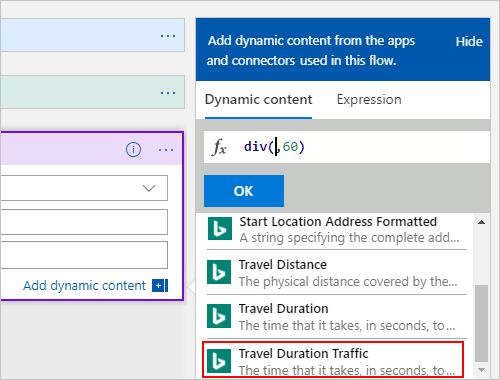


1. Provide the details for your variable as described here:

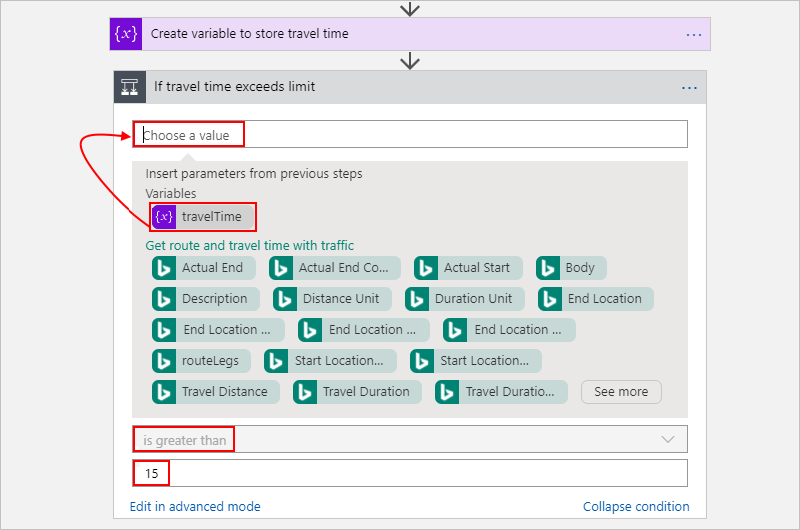
**Name** = travelTime

**Type** = Integer

**Value =** An expression that converts the current travel time from seconds to minutes (see steps under this table

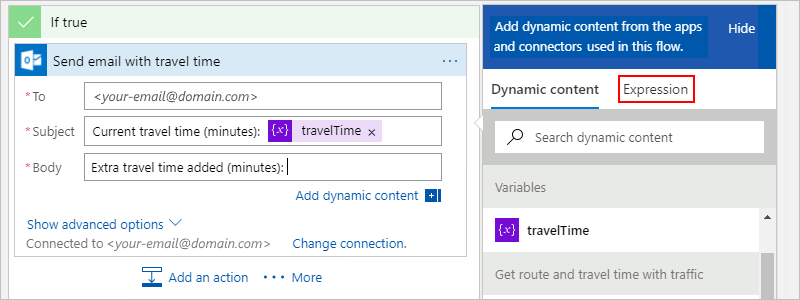


1. Add a condition that compares the travel time against a specified limit.



1. Add an action that sends email if the travel time exceeds the limit.

Body: Extra travel time added (minutes) = **sub(variable('travelTime'),15)**



1. **Run your logic app.**

**Add Custom Code with Azure Function**

1. **Create and Azure Function App 🡪 Add Azure Function (Http Trigger) = CounterCharacters**

using System.Net;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Primitives;

using Newtonsoft.Json;

public static async Task<int> **Run**(**HttpRequest** req, ILogger log)

{

string name = req.Query["name"];

string requestBody = await new StreamReader(req.Body).ReadToEndAsync();

dynamic data = JsonConvert.DeserializeObject(requestBody);

name = name ?? data?.name;

int cnt = name != null?name.Length:0;

return cnt;

}

1. From the **Function Apps** list, select your function app > **Platform features** > **CORS**. 🡪 Under **CORS**, add the \* wildcard character, but remove all the other origins in the list, and choose **Save**.
2. **Either to an existing Logic App or a New Logic App: Add Action = Azure Function 🡪 Add existing functions to Logic App**

**Using API App in Logic Apps**

1. API App 🡪 Settings 🡪 **CORS** 🡪 Allowed Origins=\* 🡪 Save
2. API App 🡪 Settings 🡪 API definition 🡪 Copy URL
3. Create a New Logic App
4. Add Triggers and Actions as per the requirement
5. To add an API App, Under Microsoft Managed API, **Search HTTP + Swagger**
6. For Swagger endpoint url = URL Copied in step2 🡪 Select the appropriate method and Continue…

## Azure Functions vs. Logic Apps

Functions and Logic Apps can both create complex orchestrations. An orchestration is a collection of functions or steps, that are executed to accomplish a complex task. With Azure Functions, you write code to complete each step, with Logic Apps, you use a GUI to define the actions and how they relate to one another.

You can mix and match services when you build an orchestration, calling functions from logic apps and calling logic apps from functions. Here are some common differences between the two.

|  |  |  |
| --- | --- | --- |
| **-** | **Functions** | **Logic Apps** |
| State | Normally stateless, but Durable Functions provide state | Stateful |
| **Development** | **Code-first (imperative)** | **Designer-first (declarative)** |
| Connectivity | About a dozen built-in binding types, write code for custom bindings | Large collection of connectors, Enterprise Integration Pack for B2B scenarios, build custom connectors |
| Actions | Each activity is an Azure function; write code for activity functions | Large collection of ready-made actions |
| Monitoring | Azure Application Insights | Azure portal, Log Analytics |
| Management | REST API, Visual Studio | Azure portal, REST API, PowerShell, Visual Studio |
| Execution context | Can run locally or in the cloud | Runs only in the cloud. |