**Azure App Service – Logic Apps**

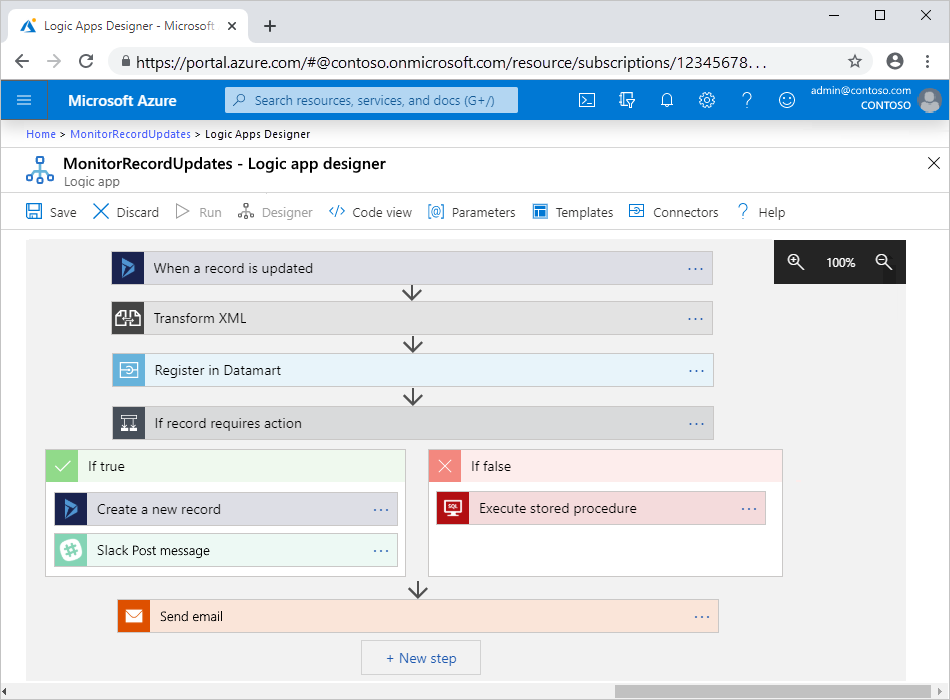
* Introduction
* Triggers and Actions
* Creating a Simple Logic App
* Handling array of Items using for-each and condition
* Logic App with Variables

**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 Microsoft Dynamics, process them and automatically send them Welcome message by SMS.
* Everyday morning at 9AM, 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 into 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 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**

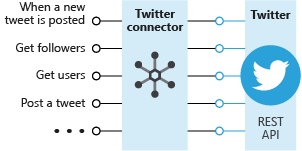
Connectors provide quick access from Azure Logic Apps to **events, data, and actions** across other apps, services, systems, protocols, and platforms.

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.

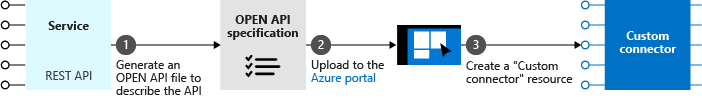
Managed connectors are organized into these groups

* + **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.

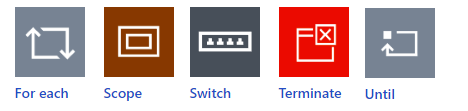


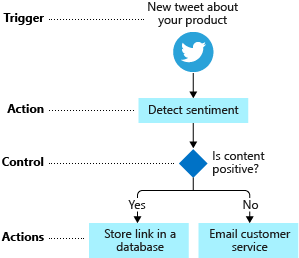
**Triggers and Actions**

* **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

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

1. Azure Portal 🡪 Search 🡪 SQL Databases 🡪 Create SQL Database
2. **Basics Tab**
   * Database name = DemoDb
   * Server = Click on **Create New**, Enter details as below

|  |
| --- |
| A screenshot of a computer  Description automatically generated with medium confidence |

* + Want to use SQL elastic pool = No
  + Workload environment = Development
  + Compute + Storage = Configure database 🡪 Service trier = Basic (For less demanding workloads)
  + Backup storage redundancy = Locally-redundant backup storage
  + Networking >

1. Networking Tab
   * Connectivity method = **Public endpoint**
   * Firewall rules
     + Check **Allow Azure services and resources access this server = Yes**
     + Click **Add your client IP address = Yes**

A screenshot of a computer

Description automatically generated

1. Skip remaining tabs
2. Review + Create 🡪 Create

**Wait for the database to be created 🡪 Go to Resource**

**Create database tables and sample data**

1. Select Overview 🡪 click Connection strings: Show database connection strings 🡪 Copy **ADO.NET (SQL authentication)** and replace password as in below string:

Server=tcp:sandeepdemoserver.database.windows.net,1433;Initial Catalog=DemoDb;Persist Security Info=False;User ID=dssadmin;Password=**{your\_password}**;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;

1. Select Database 🡪 **Query editor** 🡪 Enter username and password 🡪 OK
2. Copy command below and 🡪 Run

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

Insert some records into the above table.

INSERT INTO EmailAlerts (ToAddress, MailSubject, MailBody, EmailSent) VALUES ('<some email id>','Sub1','This is message body - 1', 0)

GO

INSERT INTO EmailAlerts (ToAddress, MailSubject, MailBody, EmailSent) VALUES ('<some email id>','Sub2','This is message body - 2', 0)

GO

INSERT INTO dbo.EmailAlerts (ToAddress, MailSubject, MailBody, EmailSent) VALUES ('<some email id>','Sub3','This is message body - 3', 0)

GO

**Create Logic App**

1. Azure Portal 🡪 Search Logic Apps 🡪 +Add
   1. Logic App name = sandeepdemo-logicapp
   2. Plan type = **Consumption**
   3. Review + Create 🡪 Create

A screenshot of a computer

Description automatically generated

1. Select Logic App 🡪 Logic App Designer 🡪 Blank Logic App
2. Click Add a trigger 🡪 Search **Recurrence 🡪** select **Schedule 🡪 Recurrence**
   1. Interval=3, Frequency = Minute
3. Click + New Step 🡪 Search **SQL Server** 🡪 Actions Tab 🡪 **Get rows (V2),**
   1. Provide the SQL Connection Details 🡪 Create New

A screenshot of a computer

Description automatically generated

* 1. Select Server Name, Database Name, Table name = "**EmailAlerts**" and Add new Parameters = **Filter Query**
  2. Click on Show mew parameter 🡪 Check Filter Query
  3. Filter Query=**EmailSent eq false** (Note: false should be in lowercase)

A screenshot of a computer

Description automatically generated with medium confidence

1. Save 🡪 Run Trigger 🡪 Run 🡪 View the Workflow as it executed and note in the outputs of Get rows (V2) that some records are returned from the database table.
2. Switch back to **Designer**
3. Click + New step 🡪 Search **Control** 🡪 Add a **For each** step
   1. Select an output from previous steps = Get rows, values
   2. Add an Action 🡪 Search Send Email 🡪 Outlook.com 🡪 Send an email (V2) 🡪 Sign In 🡪 Use any Personal Microsoft ID
      * 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 Server, 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

1. Verify that the emails are received by ID mentioned in records.
2. Verify that EmailSent is changed to true in SQL Database
   1. Sql Database 🡪 Query Editor 🡪 Login and execute query as below:

A screenshot of a 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.

**Examples:**

* Duration of 1 hour: PT1H
* Duration of 1 day: P1D
* Duration of 1 week: P1W
* Duration of 1 month: P1M
* Duration of 1 year: P1Y

Logic App With Variables and For Each

Create a Logic App which triggers when an email is received in the inbox.   
It should then send email to another email id with body having the

A screenshot of a computer

Description automatically generated

**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

<https://learn.microsoft.com/en-us/azure/logic-apps/tutorial-build-schedule-recurring-logic-app-workflow>

## 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** |
| **Development** | **Code-first (imperative)** | **Designer-first (declarative)** |
| State | Normally stateless, but Durable Functions provide state | Stateful |
| 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,** Visual Studio |
| Execution context | Can run locally or in the cloud | Runs only in the cloud. |