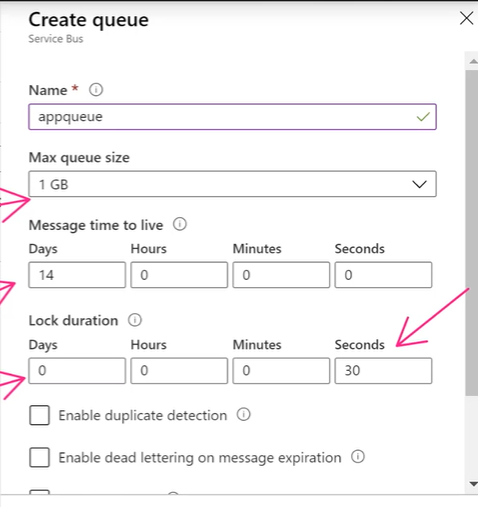
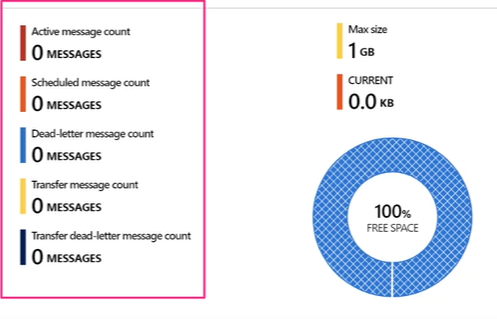
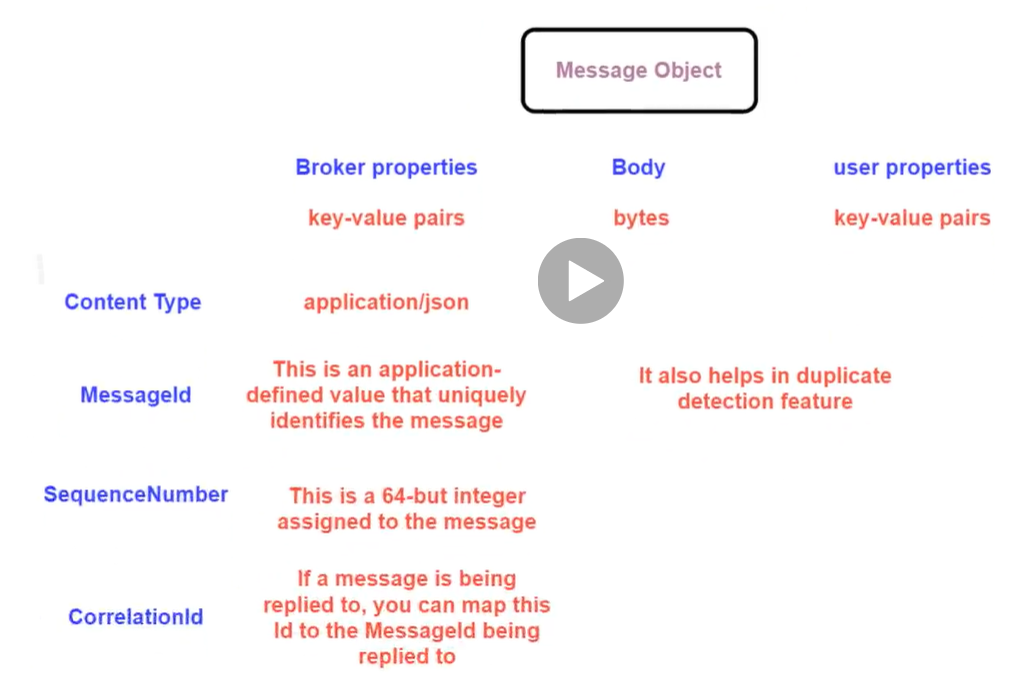
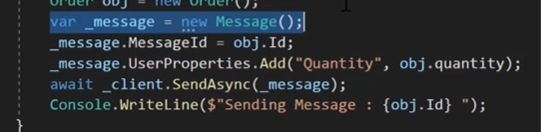
* **Connect to and consume Azure and third-party services**
* **Azure Service Bus**
* **Create Namespace first**
  + **Create Queue**

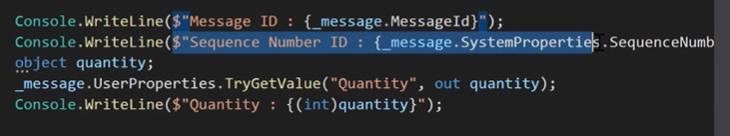




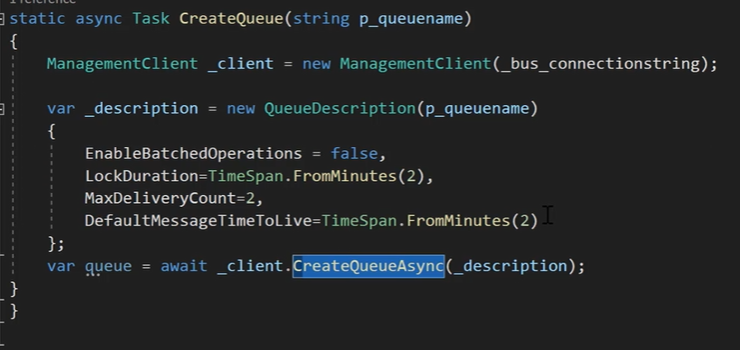
* + **Using ServiceBus Explorer messages can be published,read and peeked on queue**
  + **Using Shared Access Policy on namespace level and queue level access can be managed**
  + **Message type should be bytes**
  + **In code we can add additional properties which will override properties of queue for specific messages**



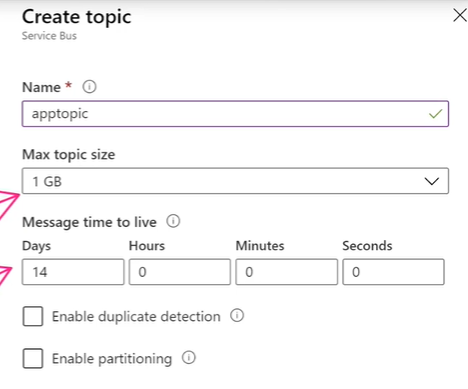


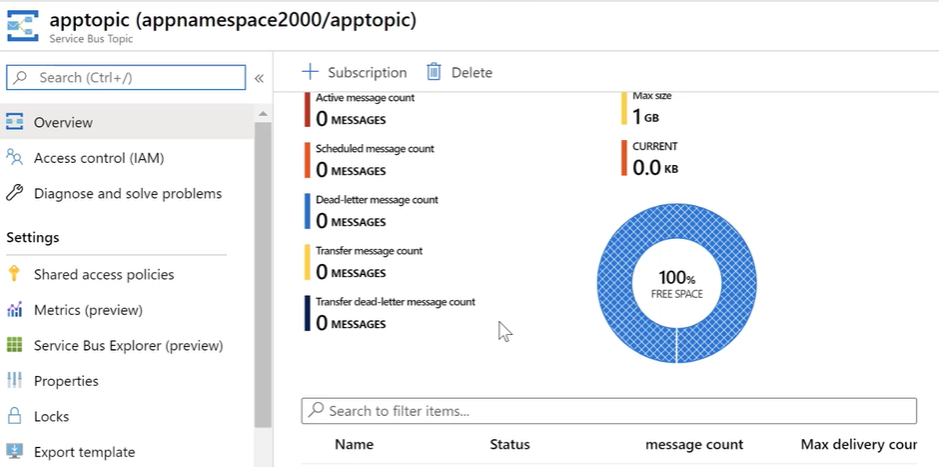


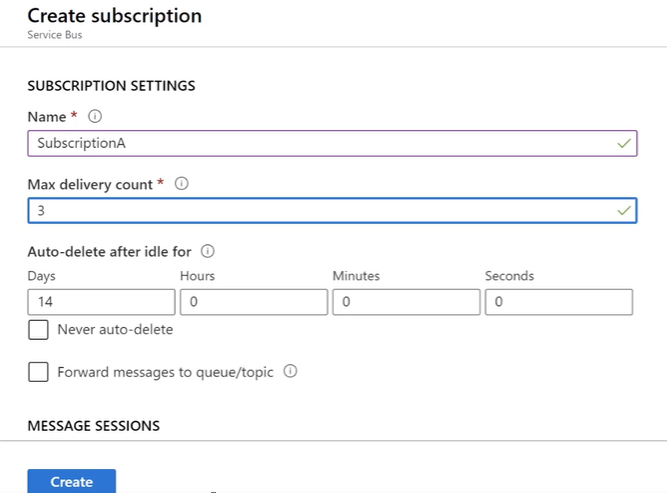
* **Peek and delete use complete async method, where to use receive and delete we can define receivemode in client connection**



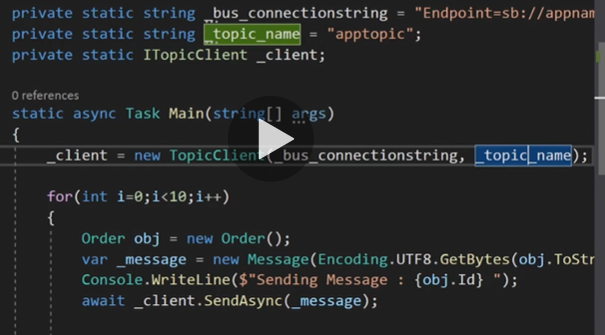
* **Create Topic**

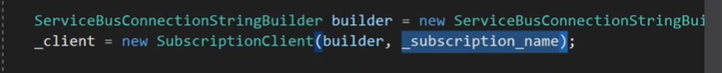


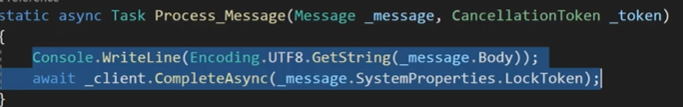




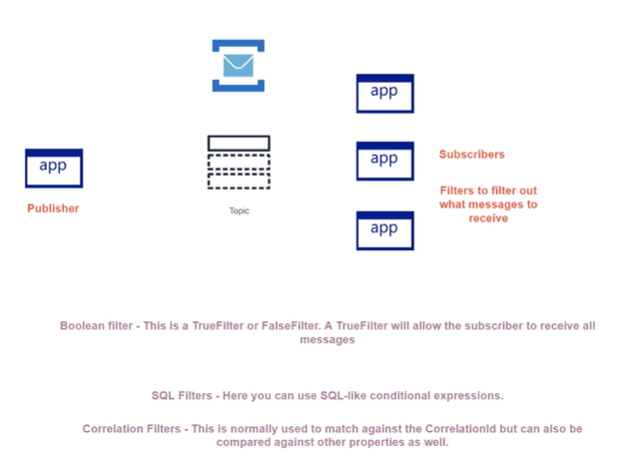
* **Policies can be used to manage access**
* **We can only send Byte message body**
* **To send and receive messages**

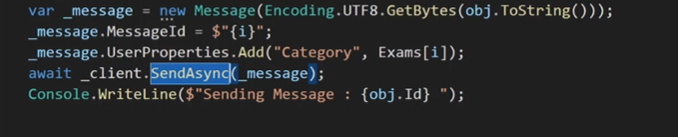


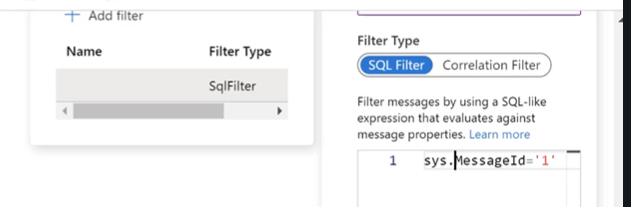




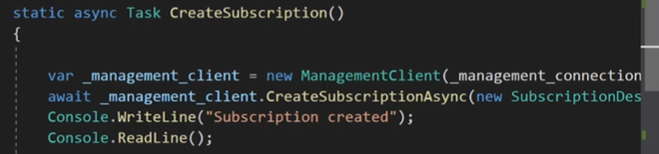
* **Using Filters**

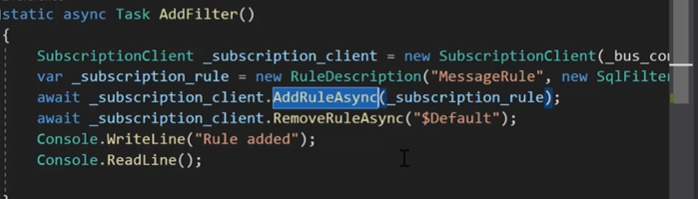




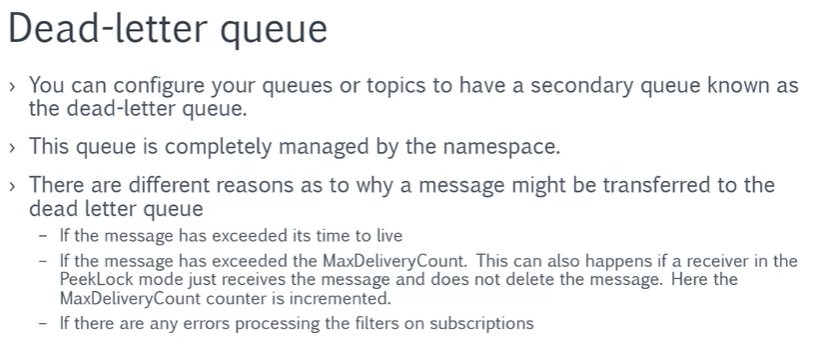


* **Creating new subscription from .net Program**
* **We need to use Management connection string and manage access**
* **We can create new subscription and after that add filters**





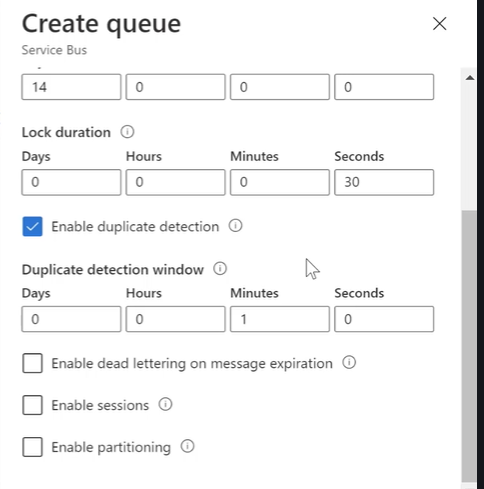
* **Dead letter Queue**



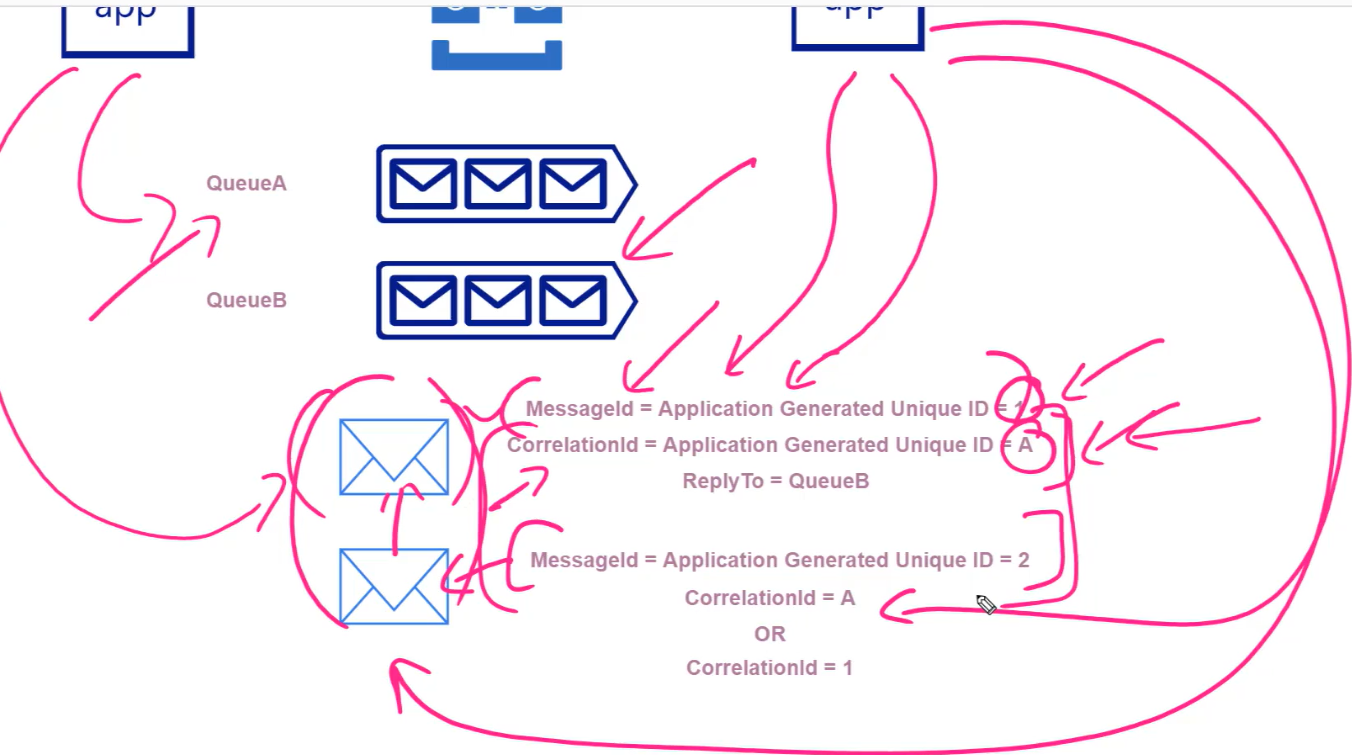
* **To get name of deadletter queue using connection of primary queue**



* **Duplicate Detection**
  + **Set duplicate detection feature on queue**
  + **Set unique messageid for each message**



* **CorelationID**
* **To link multiple messages togather**



Azure Service Bus - Best Practices - Notes

**Azure Service Bus - Best Practices**

* When you have clients that send or receive messages from the Azure Service Bus, the clients normally of the type IQueueClient.
* In the background these objects make use of the MessagingFactory object. This provides the internal management of connections.
* Don’t close the connections directly after sending or receiving messages, since establishing a connection is an expensive operation.
* Hence use the same client object for multiple operation

**Client batching**can be done with the prior version of the SDK – WindowsAzure.ServiceBus.SDK. Client batching delays the sending of messages and instead sends pending messages as a batch.

**Batched store access** is when the queue itself batches multiple messages before it is written to the internal store. This helps in better throughput.

**Enable Prefetch** – Here the receiver quietly acquires more messages from the queue or topic subscription. This is up to a value defined by the PrefetchCount limit.

When a receiver wants to receive messages, the messages are received from the buffer based on the number of prefetched messages. Then additional messages are prefetched again in the background.

The issues with Prefetch. Here in ReceiveAndDelete mode , remember when the messages are prefetched, they are removed from the queue. If the application crashes before the messages are processed, the messages are lost.

You can set the Prefetch count for the clients. QueueClient.PrefetchCount or SubscriptionClient.PrefetchCount

**If you need to implement a high throughput queue with a small number of senders and receivers**

* Use multiple message factories to create senders
* Use asynchronous operations

**If you want to decrease the latency of sending or receiving messages.**

* You can disable client batching of messages and batched store access.
* If you have a single client, consider using a prefetch count up to 20 times the processing rate of the receiver.

**If you need to implement a high throughput queue with a large number of senders and small number of receivers**

* For the sender that resides in a different process, use only one single factory per process.
* Use asynchronous operations and take advantage of client-side batching.
* Leave the setting of batched store access enabled.
* Set the prefetch count to 20 times that of the maximum processing rates of all receivers of a factory.

**If you need to implement a high throughput queue with a small number of senders and large number of receivers**

* If each receiver is in a different process , use only a single factory per process.
* Leave batched store access enabled.
* Set the prefetch count to a lower value. This is because you have multiple receivers.

**If you need to implement a high throughput topic with a small number of senders and small number of subscriptions**

* Increase the overall send rate by using multiple message factories to create senders
* Increase the overall receive rate by using multiple message factories to create receivers
* Use asynchronous operations and client-side batching.
* Leave batched store access enabled.
* Set the prefetch count to 20 times the maximum processing rates of all receivers of a factory
* **Azure Service Bus - Azure CLI**

You can use the following commands to work with the Azure Service Bus service

* **// You can set the following variables for your script**

$namespace="appnamespace4000"

$resourcegrp="demogrp1"

$location="Central US"

$queuename="appqueue"

$topicname="apptopic"

$subscriptionName="SubscriptionA"

* **// You can use the following command to create a service bus namespace**

az servicebus namespace create --name $namespace --resource-group $resourcegrp --location $location --sku Standard

* **// You can use the following command to create a service bus queue**

az servicebus queue create --resource-group $resourcegrp --namespace-name $namespace --name $queuename --max-size 1024

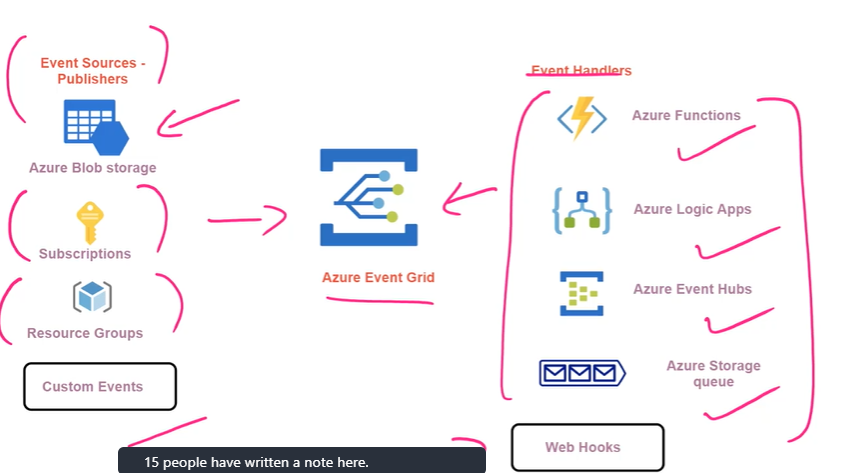
* **// You can use the following command to create a service bus topic**

az servicebus topic create --resource-group $resourcegrp --namespace-name $namespace --name $topicname --max-size 1024

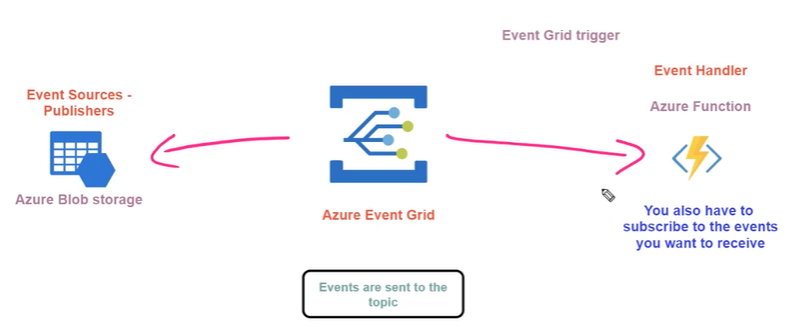
* **// You can use the following command to create a service bus subscription**

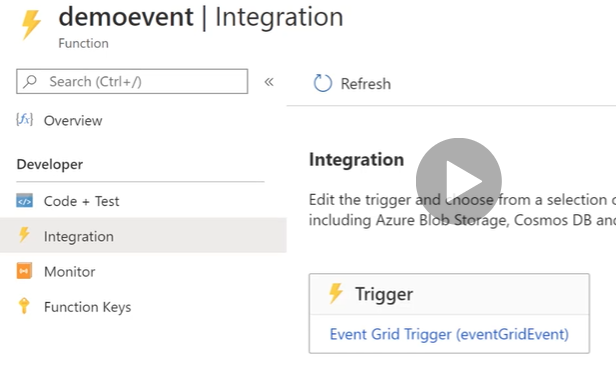
az servicebus topic subscription create --resource-group $resourcegrp --namespace-name $namespace --topic-name $topicname --name $subscriptionName

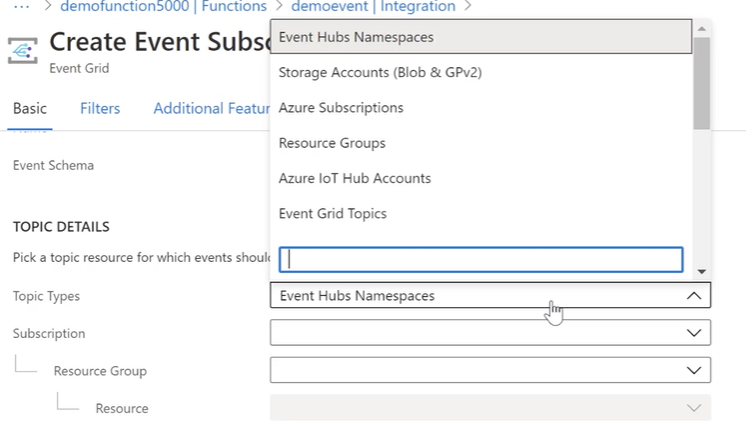
* **Azure Event Grid Service**



**Integrating Event Grid to Azure Function**







* **Event grid schema**



* + **Function Code:**

#r "Microsoft.Azure.EventGrid"

using Microsoft.Azure.EventGrid.Models;

public static void Run(EventGridEvent eventGridEvent, ILogger log)

{

log.LogInformation(eventGridEvent.Data.ToString());

log.LogInformation($"Event Id {eventGridEvent.Id}");

log.LogInformation($"Event Topic {eventGridEvent.Topic}");

log.LogInformation($"Event Subject {eventGridEvent.Subject}");

}

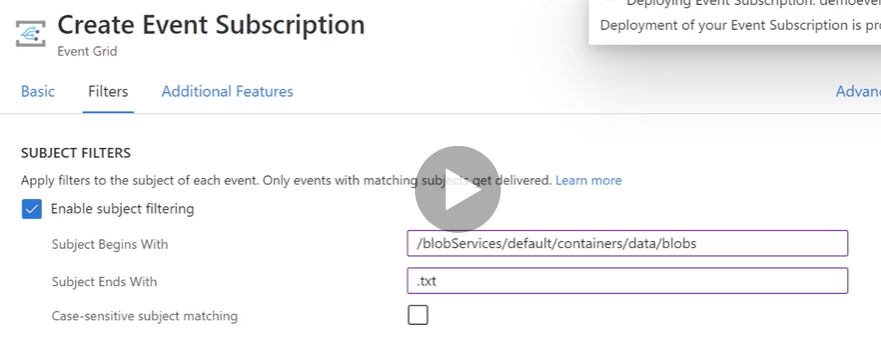
* + **Integrating Event Grid to Azure Service Bus (Queue)**

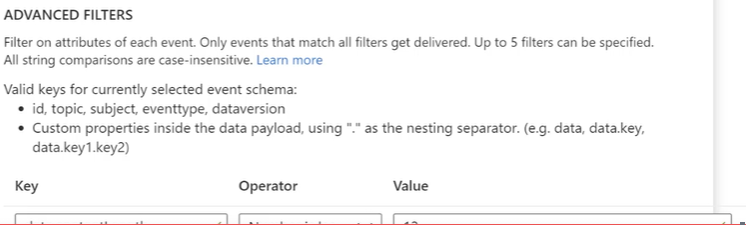
**CLI Command**

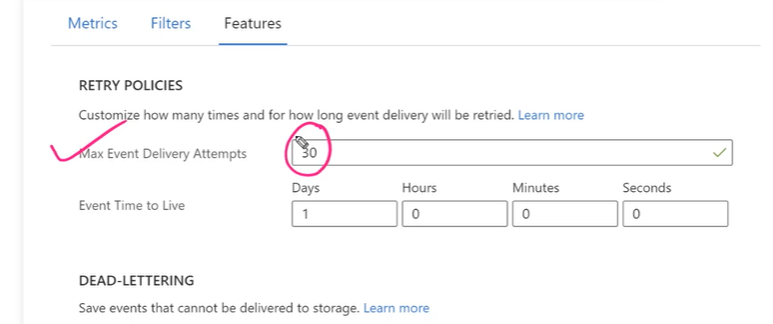
az eventgrid event-subscription create --name queue-subscription --source-resource-id /subscriptions/20c6eec9-2d80-4700-b0f6-4fde579a8783/resourceGroups/demogrp1/providers/Microsoft.Storage/storageAccounts/demostore4000 --endpoint-type servicebusqueue --endpoint /subscriptions/20c6eec9-2d80-4700-b0f6-4fde579a8783/resourceGroups/demogrp1/providers/Microsoft.ServiceBus/namespaces/appnamespace4000/queues/appqueue

* **Event Grid Filters**

To specify subject (i.e container name from storage account)





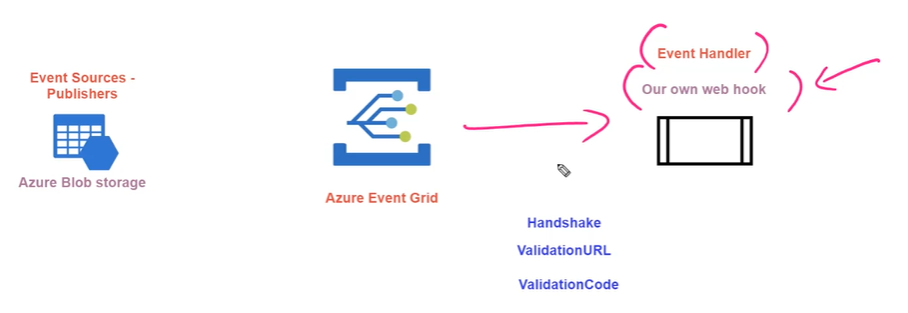


* **Integrating Event Grid to custom Application(Webhook)**

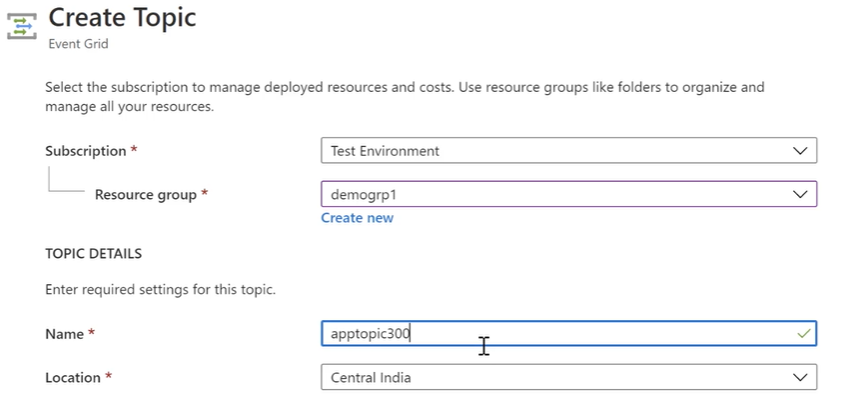
- First handshake need to be done

- Event Grid will send validation URL and Code to custom application and application has to send it back

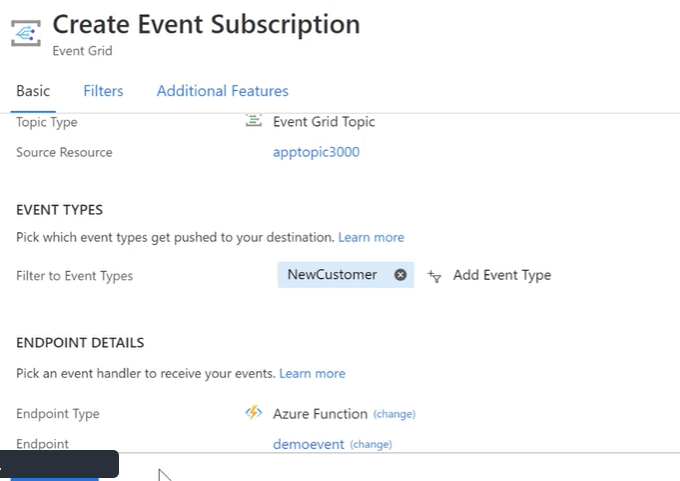
- Once handshake is established events happening on blob service can be received by custom application



* **Adding Custom Event Grid Topics**



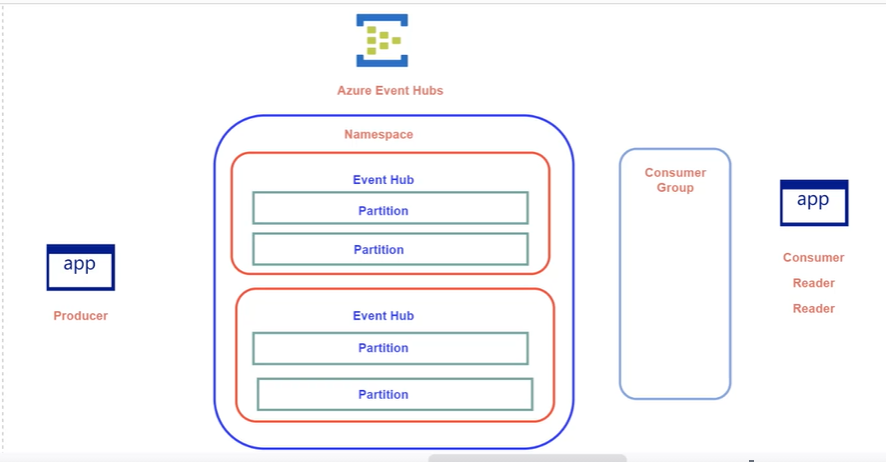
* **Adding Event subscription using custom topic connecting to Azure Function**

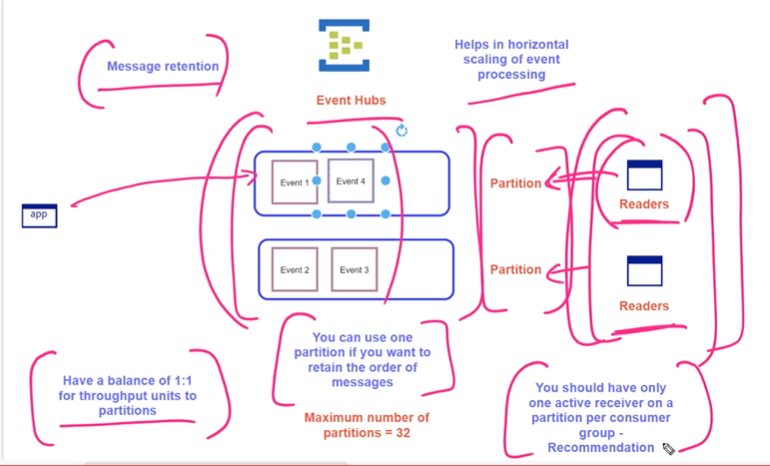


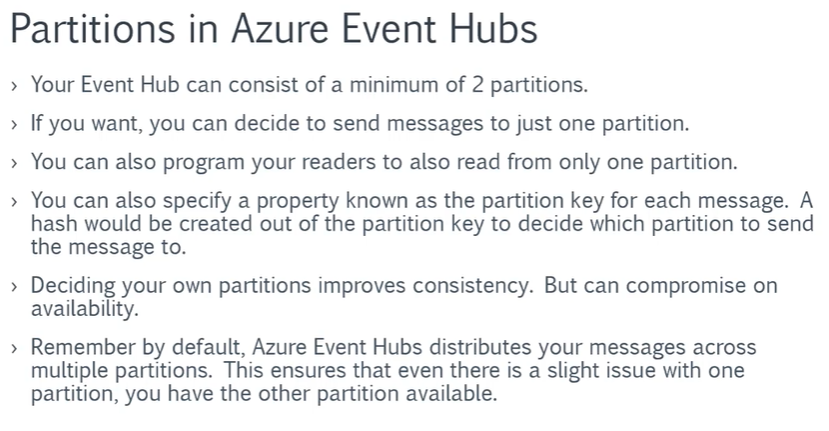
A .net program can form a custom event using event grid schema when this event is published azure function will receive this event.

* **Azure Event Hubs**

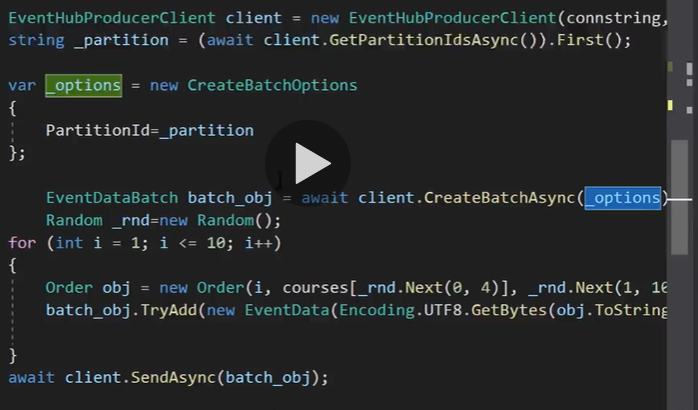
**Architecture**

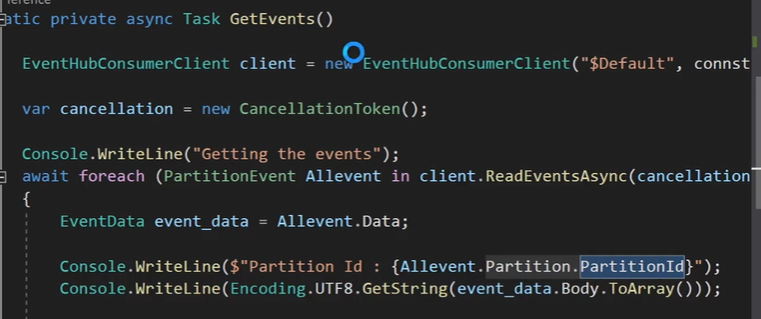






**Sending Messages to specific partition and reading from that partition**

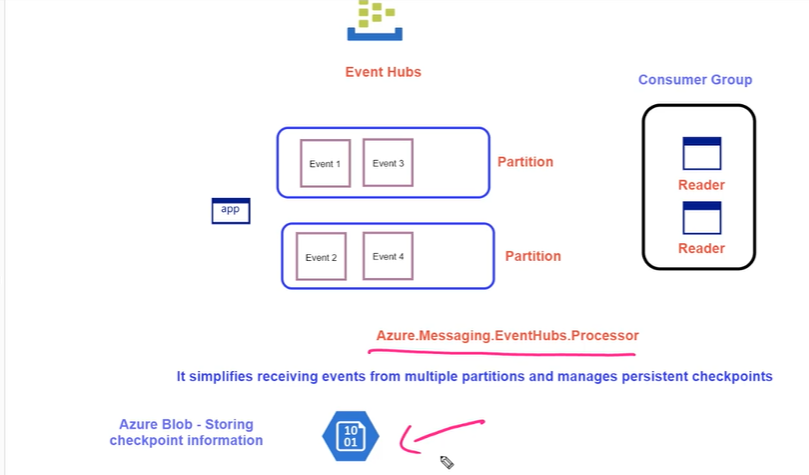


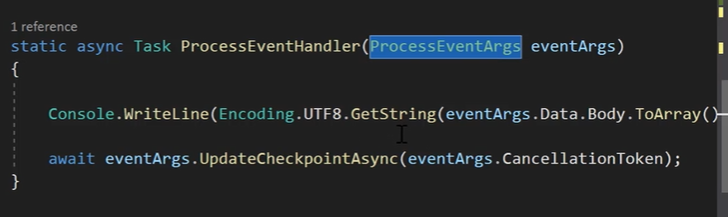


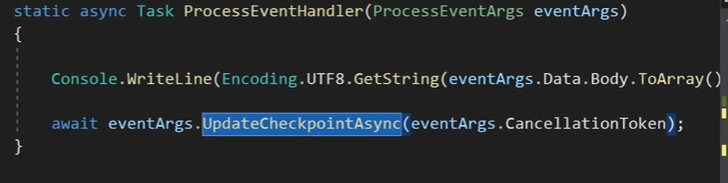
* **Offset**



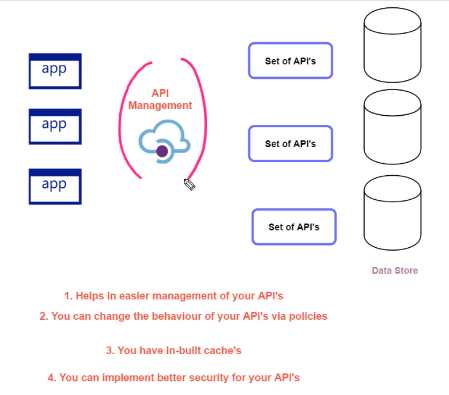
* **Event Processor**
  + **To manage offset**

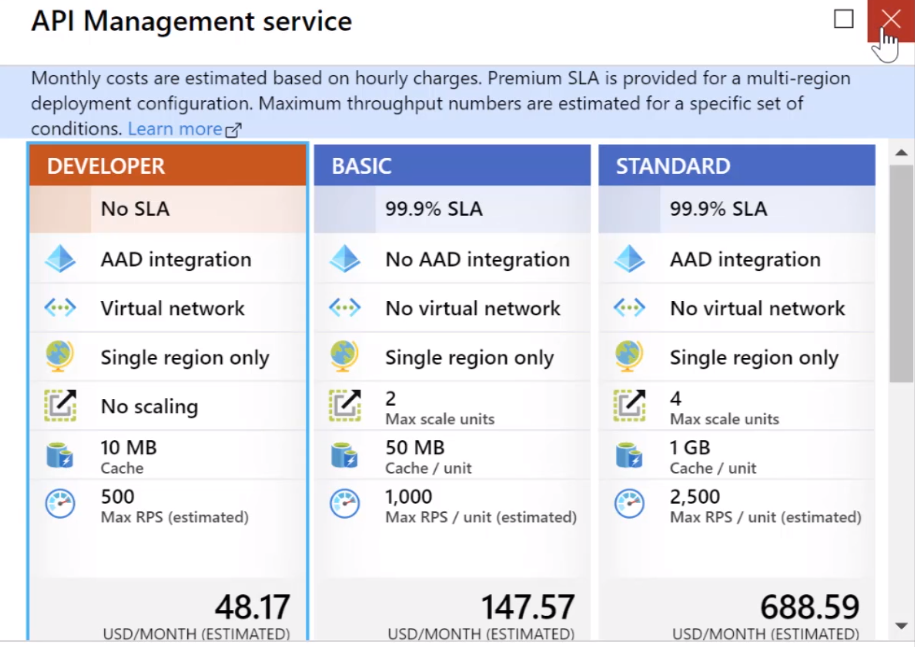






* **API Management Service**







* **API Policies**

[**https://docs.microsoft.com/en-us/azure/api-management/api-management-advanced-policies**](https://docs.microsoft.com/en-us/azure/api-management/api-management-advanced-policies)

**To rewrite URL to return all courses**

**<policies>**

**<inbound>**

**<base />**

**<rewrite-uri template="@{**

**return "/api/Courses"; }" />**

**</inbound>**

**<backend>**

**<base />**

**</backend>**

**<outbound>**

**<base />**

**</outbound>**

**<on-error>**

**<base />**

**</on-error>**

**</policies>**

**To add query param**

**<policies>**

**<inbound>**

**<base />**

**<set-variable name="CourseId" value="1" />**

**<rewrite-uri template="@{**

**return "/api/Courses/"+ context.Variables.GetValueOrDefault<string>("CourseId"); }" />**

**</inbound>**

**<backend>**

**<base />**

**</backend>**

**<outbound>**

**<base />**

**</outbound>**

**<on-error>**

**<base />**

**</on-error>**

**</policies>**

**Policy Conditions:**

**<policies>**

**<inbound>**

**<base />**

**<set-variable name="Total" value="@(context.Request.Headers["total"][0])" />**

**<choose>**

**<when condition="@(int.Parse(context.Variables.GetValueOrDefault<string>("Total"))<10)">**

**<rewrite-uri template="@{return "/api/Courses";}" />**

**</when>**

**<otherwise />**

**</choose>**

**</inbound>**

**<backend>**

**<base />**

**</backend>**

**<outbound>**

**<base />**

**</outbound>**

**<on-error>**

**<base />**

**</on-error>**

**</policies>**

**<policies>**

**<inbound>**

**<base />**

**<set-variable name="Total" value="@(context.Request.Headers["total"][0])" />**

**<choose>**

**<when condition="@(int.Parse(context.Variables.GetValueOrDefault<string>("Total"))<10)">**

**<rewrite-uri template="@{return "/api/Courses";}" />**

**</when>**

**<otherwise>**

**<rewrite-uri template="@{return "/Home/Details/1";}" />**

**<set-backend-service base-url="https://demostaging4000.azurewebsites.net"/>**

**</otherwise>**

**</choose>**

**</inbound>**

**<backend>**

**<base />**

**</backend>**

**<outbound>**

**<base />**

**</outbound>**

**<on-error>**

**<base />**

**</on-error>**

**</policies>**

**Outbound Policy:**

**<policies>**

**<inbound>**

**<base />**

**<rewrite-uri template="@{**

**return "/api/Courses"; }" />**

**</inbound>**

**<backend>**

**<base />**

**</backend>**

**<outbound>**

**<base />**

**<choose>**

**<when condition="@(context.Response.StatusCode == 200)">**

**<set-body>**

**@{**

**return "Error";**

**}**

**</set-body>**

**</when>**

**</choose>**

**</outbound>**

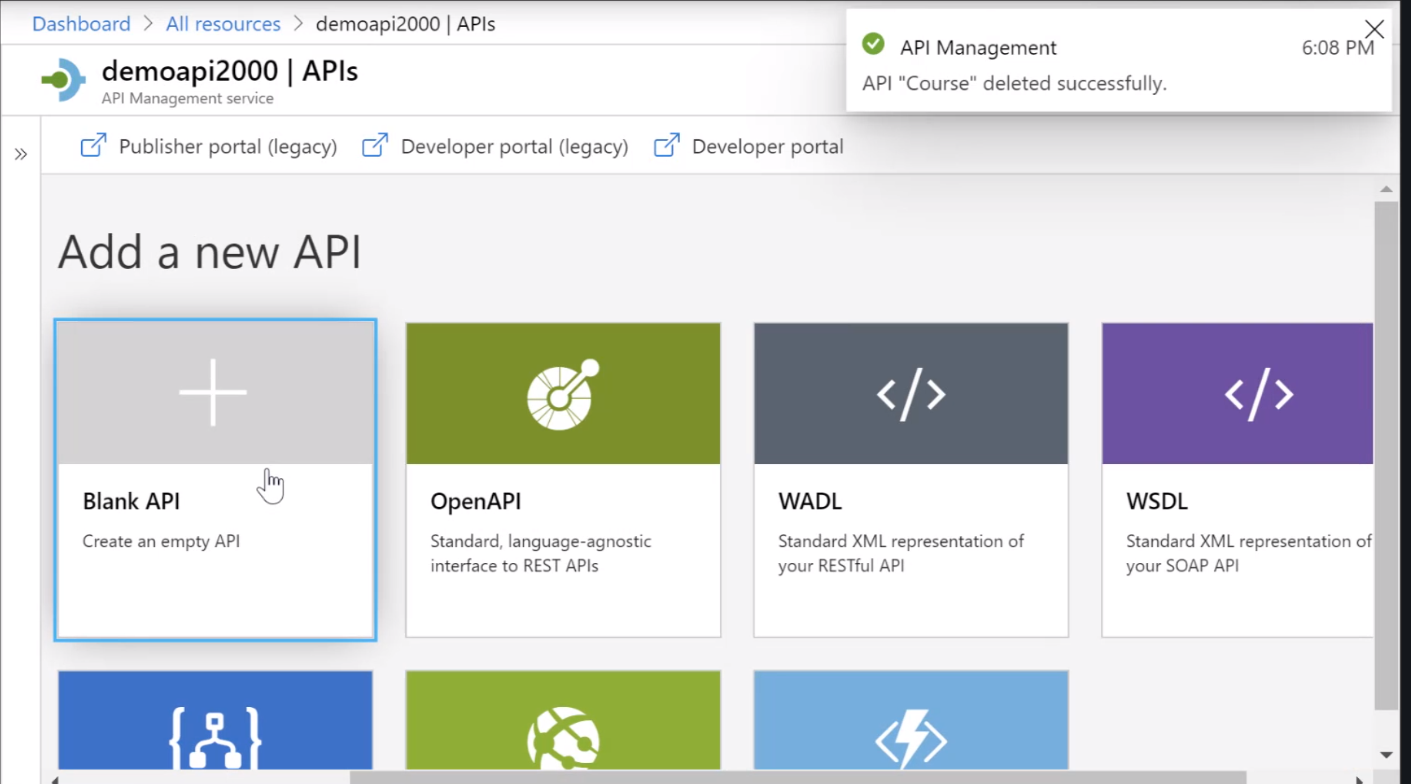
**<on-error>**

**<base />**

**</on-error>**

**</policies>**

* **Importing API definition (swagger/open api standards)**



Azure API Management Overview

#### ****Azure API Management****

This is a management service for your backend API's. This service helps to publish your API's to external partners , to your consultants or even to your developers.

Your API Management instance can carry out the following tasks

* Accept API calls from users and route them on to your backend API's
* Have the ability to verify API keys, JWT tokens and certificates
* It can also enforce usage quotas and rate limits
* It can also cache responses for better performance of responses to users

You can policies in Azure API Management to control the way requests and responses are handled

You can use the following link to refer to all of the policies

<https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-policies>

To work along with the lab to restrict access based on an IP address, you can use the following reference policy

1. <policies>
2. <inbound>
3. <ip-filter action="forbid">
4. <address>92.98.36.48</address>
5. </ip-filter>
6. </inbound>
7. <backend>
8. <base />
9. </backend>
10. <outbound>
11. <base />
12. </outbound>
13. <on-error>
14. <base />
15. </on-error>
16. </policies>

As an extra note , please refer to the following PowerShell commands that can be used to work with the API Management instance

**// 1. Here we are setting PowerShell variables.**

**// This is the name we want to give to the API**

$appservicename = "core-management"

**// Here we mention an existing resource group**

$resourceGroupName = "demogrp1"

**// Here we specify the location in which we want to create our API Management Instance**

$location = "East US"

**// Here we specify the organization name**

$organisation = "CompanyA"

**// Here we specify the organization email**

$adminEmail = "admin@company.com"

**// Here we are assuming that our API is using the Open API Specification**

$Url="https://corewebapi4000.azurewebsites.net/swagger/v1/swagger.json"

**// Here we are giving the location of an Azure Web App that is hosting our API**

$webapi="https://corewebapi4000.azurewebsites.net"

**// The below PowerShell command can be used to create a new API Management Instance**

New-AzApiManagement -ResourceGroupName $resourceGroupName -Name $appservicename -Location $location -Organization $organisation -AdminEmail $adminEmail

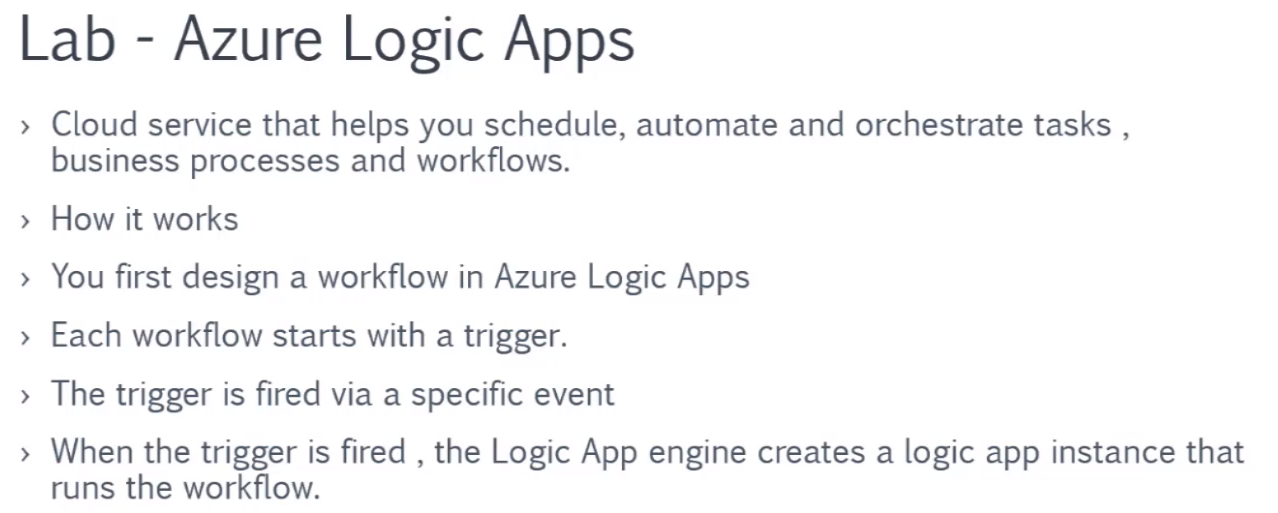
**// After we create the Azure API Management Instance, we then need to set the context**

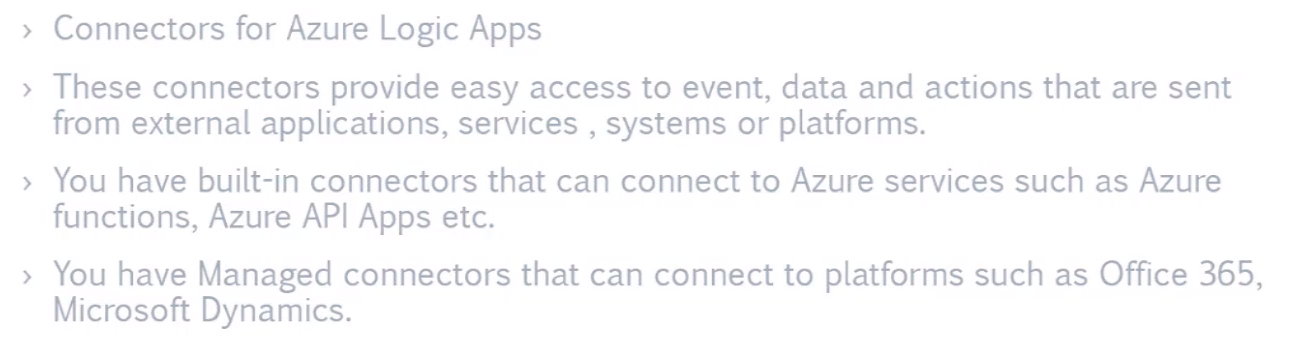
$context = New-AzApiManagementContext -ResourceGroupName $resourceGroupName -ServiceName $appservicename

**// We can then import an API using our web api**

$api = Import-AzApiManagementApi -Context $context -SpecificationUrl $Url -SpecificationFormat "OpenApi" -ServiceUrl $webapi

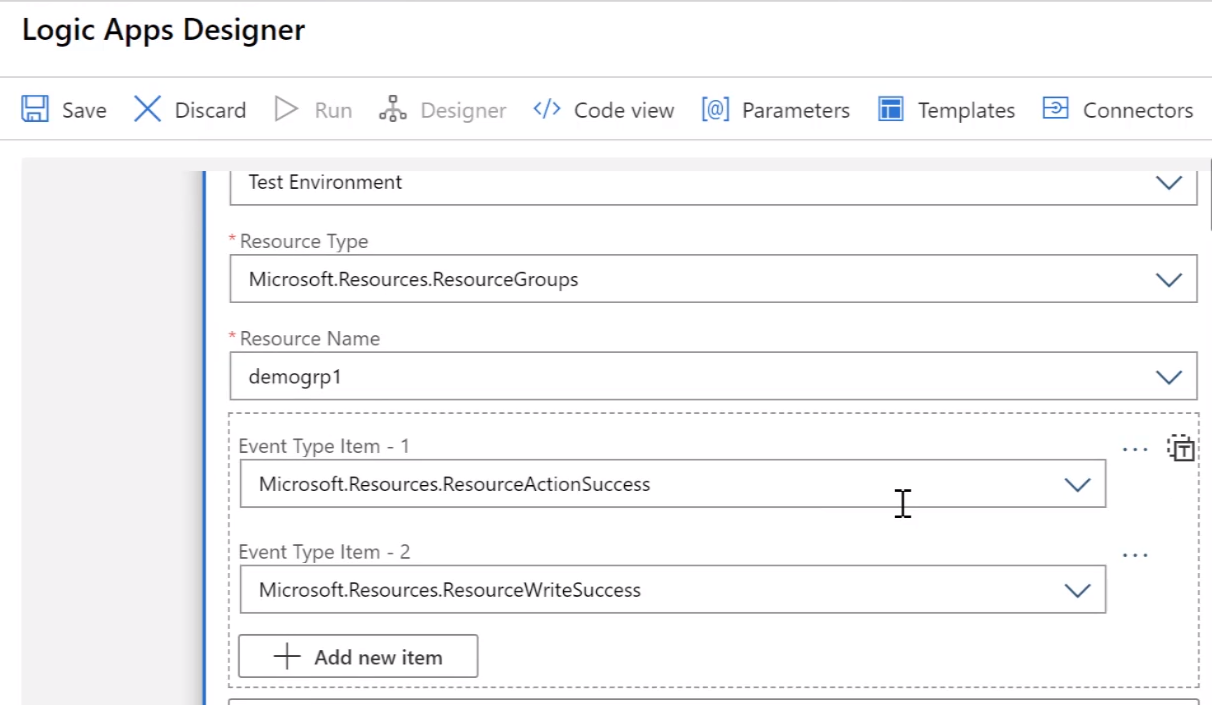
* **LOGIC APPS**

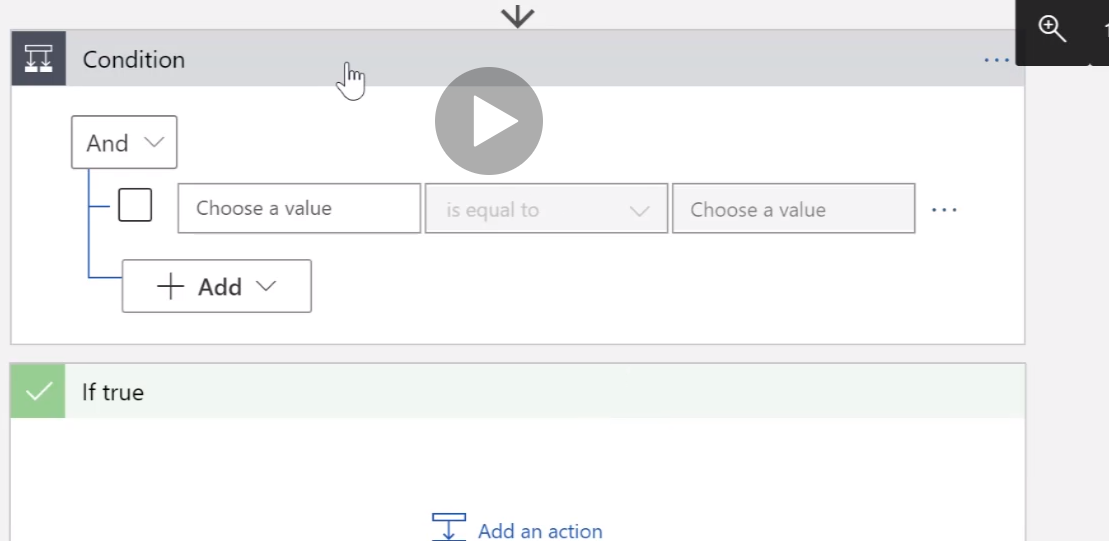


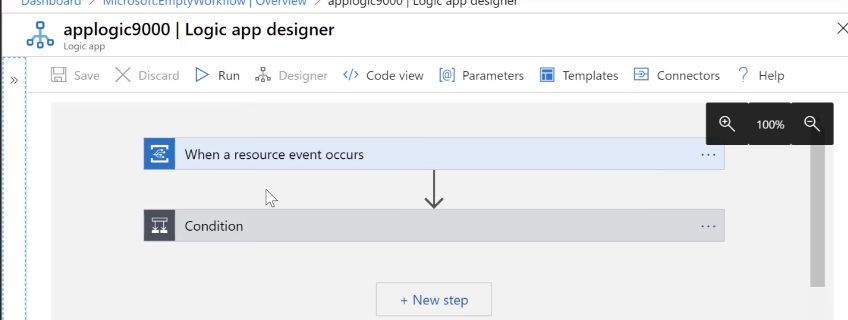


You can choose triggers , there are also templates available to start with, or you can create empty logic app

After trigger you can select action, there are external connectors also available







Azure Logic Apps - Overview and Debugging

#### ****Azure Logic Apps****

**What is Azure Logic Apps**

This is a loud service that helps you schedule, automate and orchestrate tasks , business processes and workflows.

**How it works**

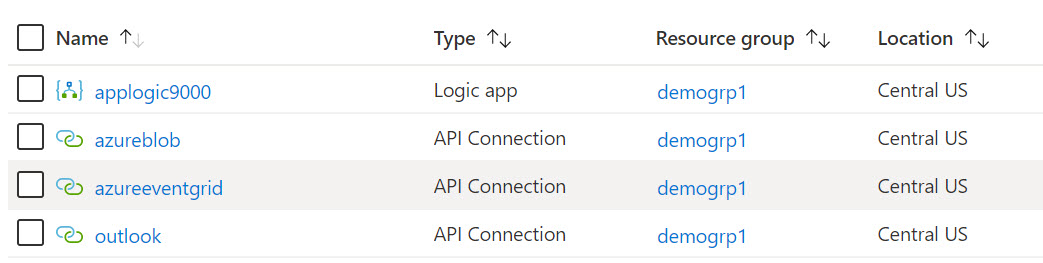
* You first design a workflow in Azure Logic Apps
* Each workflow starts with a trigger.
* The trigger is fired via a specific event
* When the trigger is fired , the Logic App engine creates a logic app instance that runs the workflow.

In Azure Logic Apps, you have connectors that help connect to Azure based services and other third party services as well.

* These connectors provide easy access to event, data and actions that are sent from external applications, services , systems or platforms.
* You have built-in connectors that can connect to Azure services such as Azure functions, Azure API Apps etc.
* You have Managed connectors that can connect to platforms such as Office 365, Microsoft Dynamics.

Now after completing the labs on Azure Logic Apps, when you look at your resources , you will notice that there are API Connection resources in place.

These are the resources created for the various connectors when working with Azure Logic Apps.



Here we have a connector for Azure Blob storage, for the Azure Event Grid service and for connecting to Outlook.

Please refer to the following URL for connectors for Azure Logic Apps

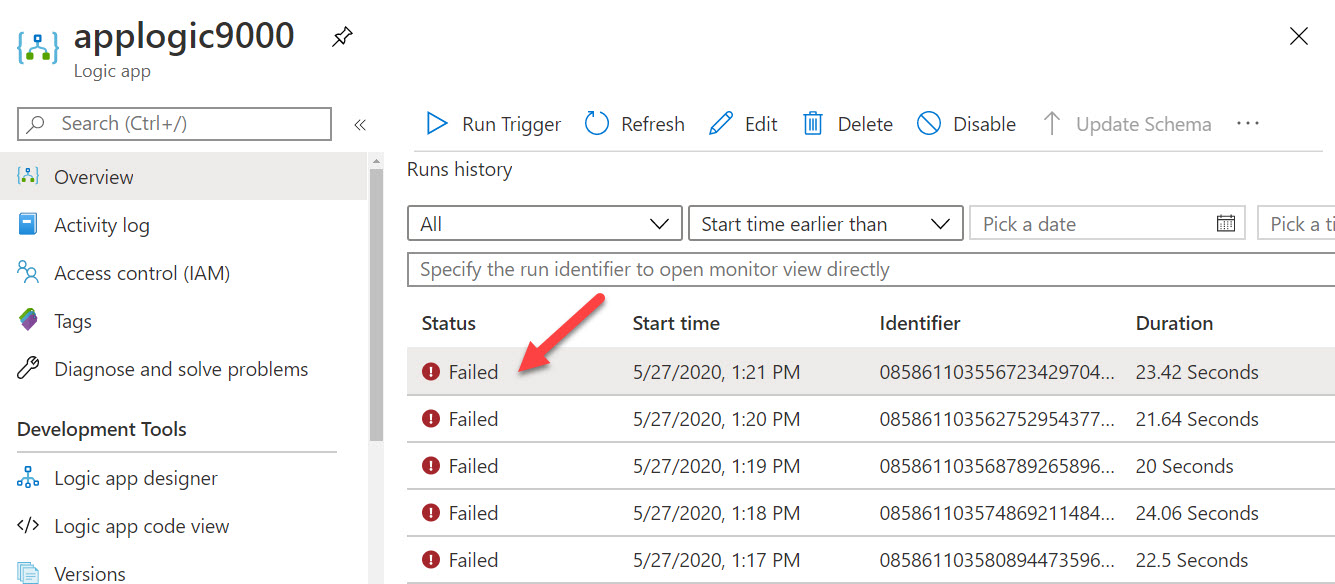
<https://docs.microsoft.com/en-us/azure/connectors/apis-list>

If you want to use your own connector, please refer to this link

**Debugging your Logic Apps**

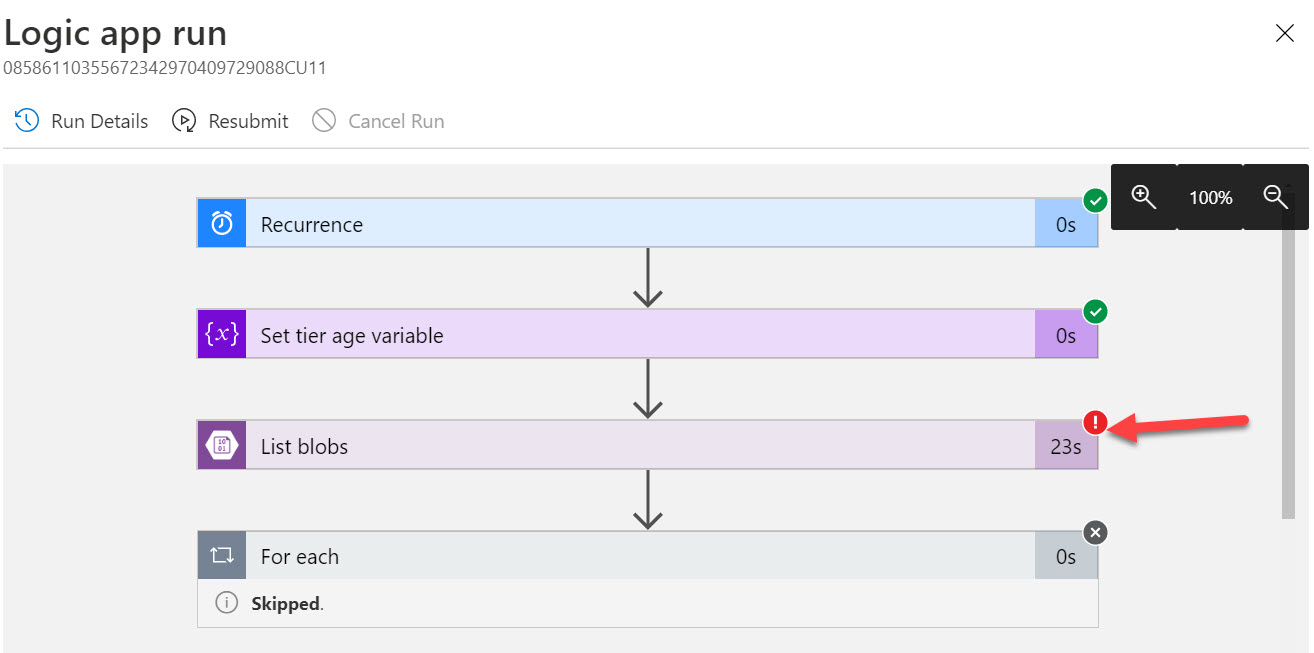
If your Logic App is not working for some reason, you can use the following Debugging procedure

1) Go to the Run history in the Overview of the Azure Logic App instance. This is at the bottom of the page.

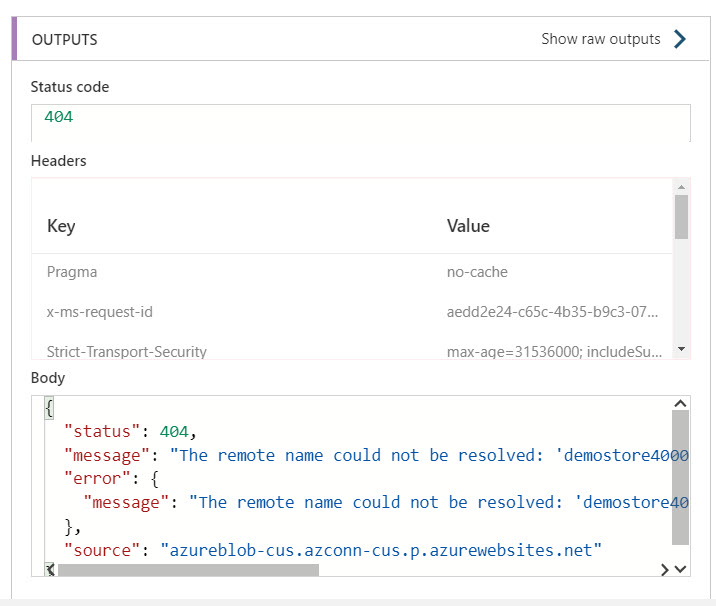


Click on any Failed run

You will then see the details of the run. You can also see the step which failed in the Run



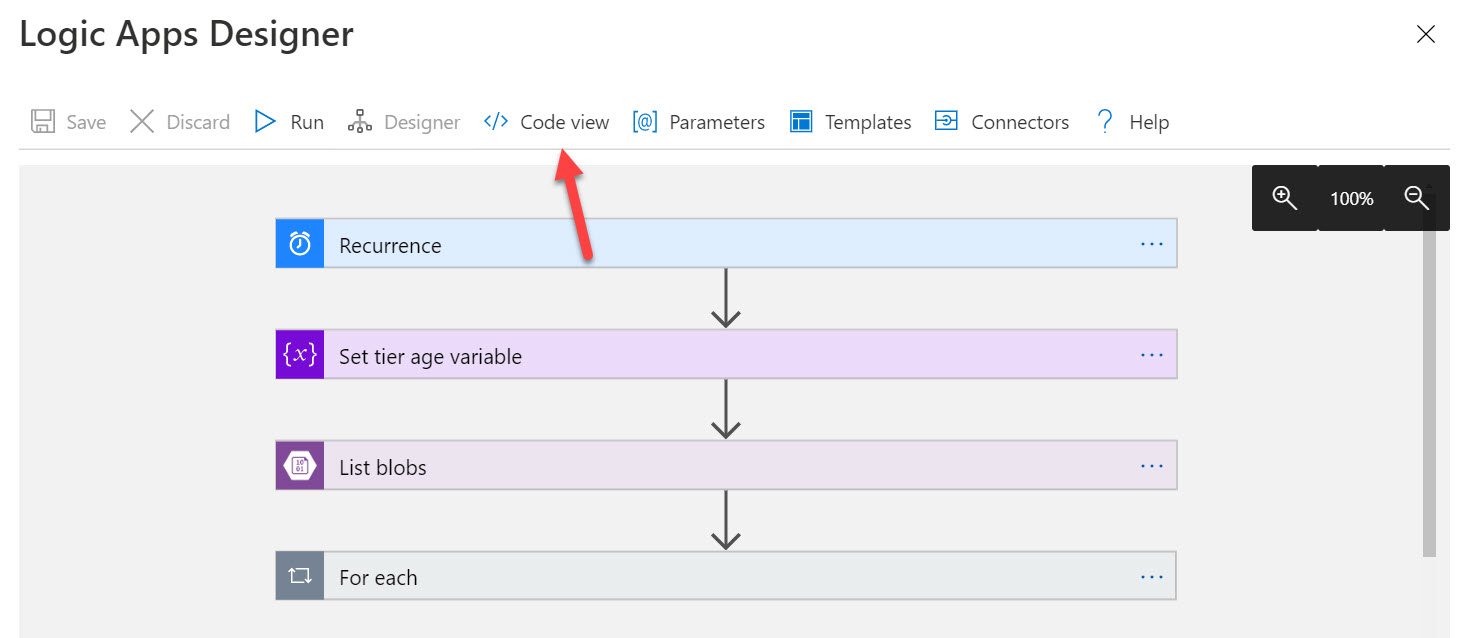
If you just expand the failing step, you will see the actual error for that step



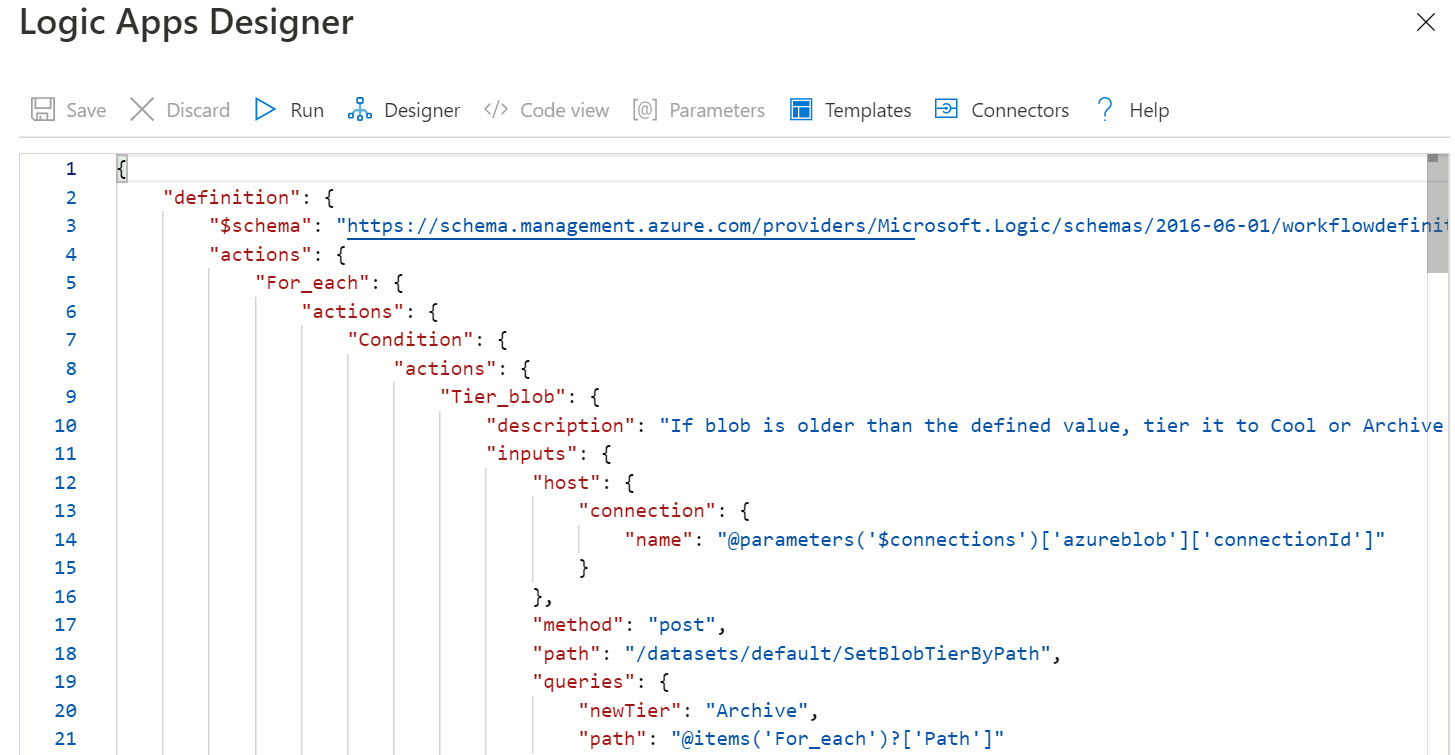
#### ****Azure Logic Apps - Extra Notes from exam perspective****

**Code view**

When you create the workflow for the Azure Logic App, you would normally first create the workflow in the Logic Apps Designer. But at any time, you can switch over to the Code view



This gives you the JSON representation of the entire workflow. Here you can edit the definition of the workflow if you are comfortable working with JSON.



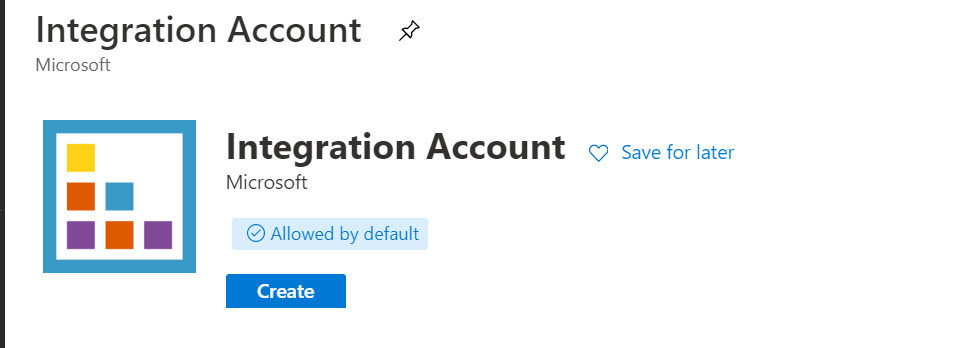
**B2B Enterprise Integration**

With Azure Logic Apps, you have multiple connectors in place. You can use these connectors to connect to Azure services or even other third party services as well.

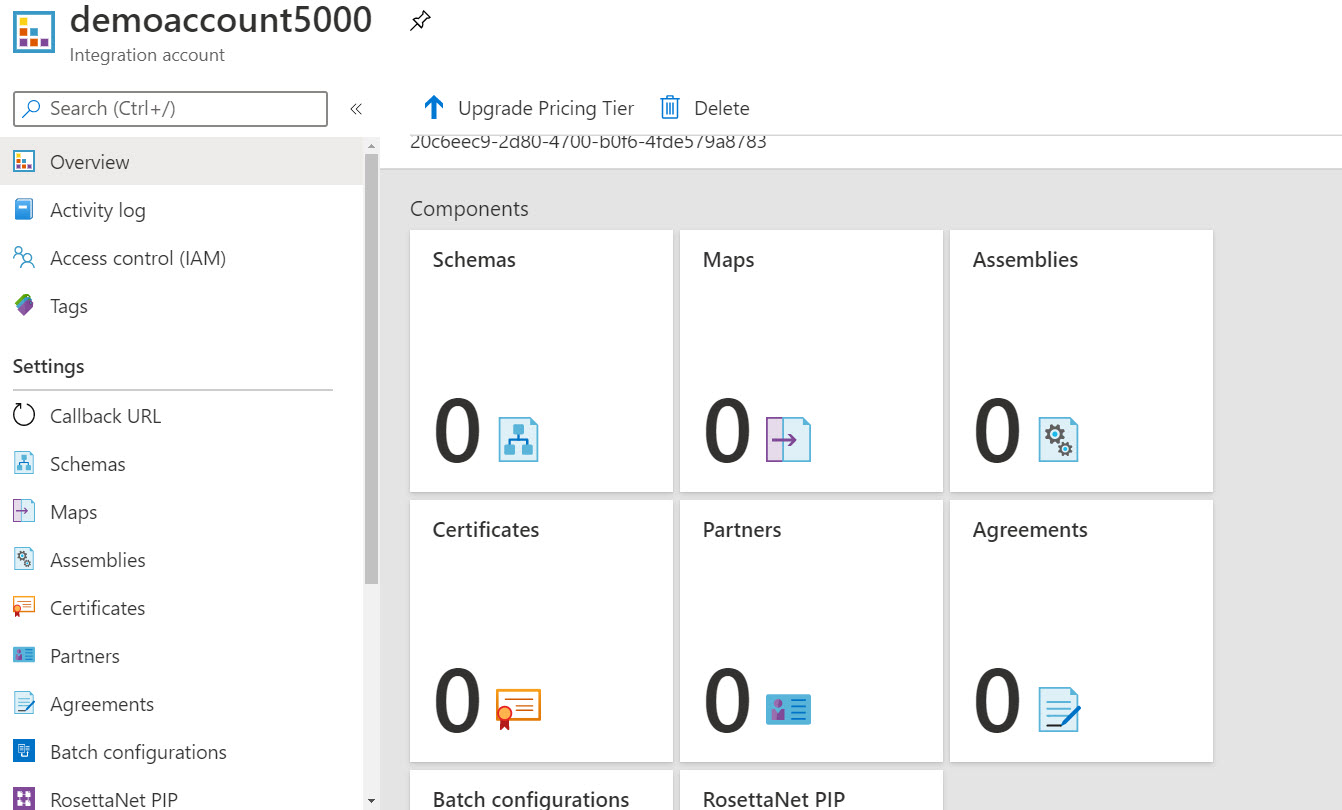
There may also be cases wherein you may want to connect Azure Logic Apps to your on-premises business services as well. Your business services could also be hosted in other environments as well.

These business services might not exchange data in the traditional way and may depend on protocols such as EDI ( Electronic Data Interchange) or EAI ( Enterprise Application Integration). In such cases , you can use the Enterprise Integration Pack with Azure Logic Apps to work with these business services.

To enable this, you first need to create a new resource known as an Integration Account in Azure.



Once you create the Integration Account , you can add artifacts such as Partners , agreements etc onto the Integration Account.



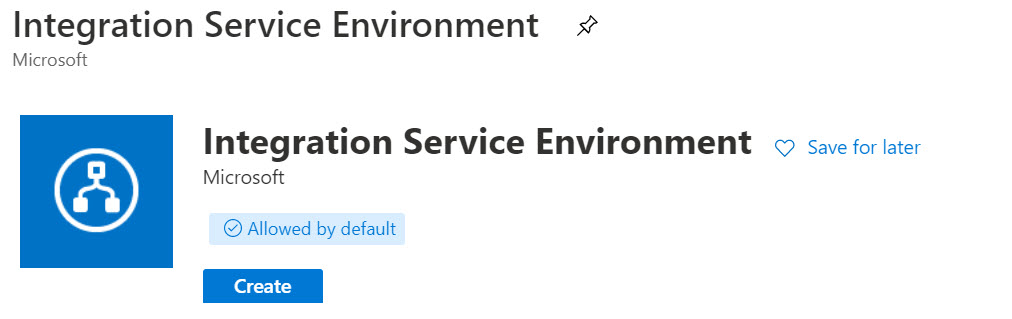
Once you add the artifacts, you can then link your Logic App to the Integration account.

You can then build workflows in Azure Logic Apps using the artifacts in the Integration account.

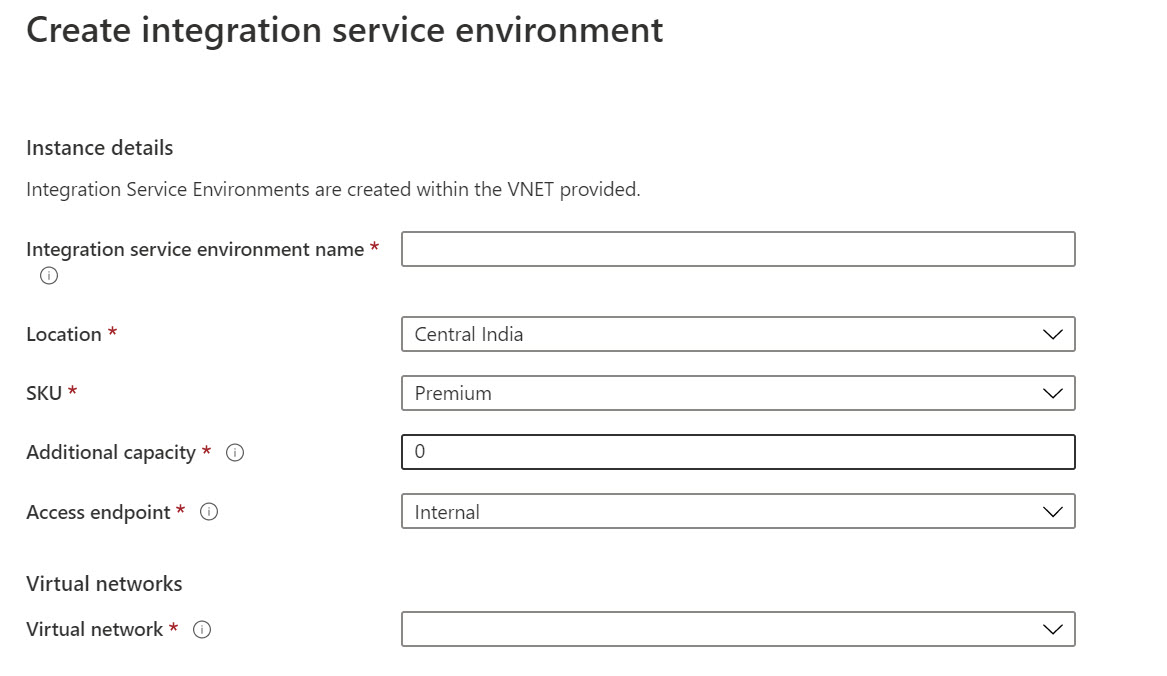
#### ****Integration Service Environment****

If you need your Azure Logic App and your Integration accounts to access an Azure virtual network, then you have to create an Integration service environment. This is a dedicated environment that uses dedicated storage and other resources as well. These resources are kept separate from "global" multi-tenant Logic Apps service.

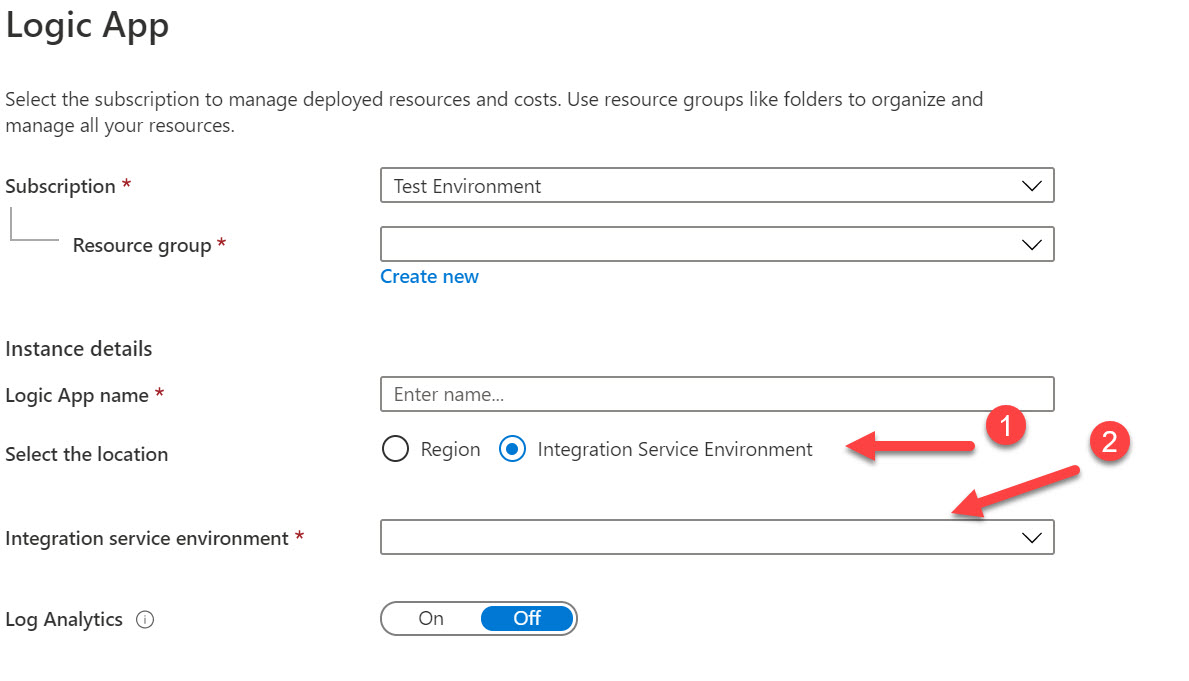
To use this feature, you first have to create an Integration Service Environment resource in Azure



Here you have to specify details such as the Location of the service environment, the Location, the Virtual network to associate the Integration Service Environment with.



Once you have the Integration Service Environment in place, then when you create the Azure Logic App, you can then associate the Azure Logic App with the Integration Service Environment



#### ****On-premise data gateway****

You can connect Azure Logic Apps to your on-premise data sources using the on-premise data gateway.The gateway works as a bridge that provides quick data transfer and encryption between data sources on premises and your logic apps

For more information on the on-premise data gateway, please refer to the following link

<https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-gateway-install>

**Azure Notification Hubs**

Azure Notification Hubs allows you to send notifications to any platform (iOS, Android, Windows etc) from any back-end platform.

Notifications allows users to get information , especially on their mobile devices , for any sort of desired information.

