Go to <https://portal.azure.com>

Samples: <https://github.com/Azure-samples> type in filters dropdown e.g. for dot net core <https://github.com/Azure-Samples/dotnet-core-api>

**Azure API Management:** Azure API Management is a solution for publishing, managing, securing and analyse APIs to external and internal consumers. Can publish multiple versions of your API and integrate Azure API Management with Azure Application Insights. APIM is often used together with Azure service bus, logic apps and event grid. Create API gateway for your micro services. We can set policies like cache for operations. Can set read/use/rate limit like only 5 in one minute.

Breaking changes to API need versioning, for all other changing we call it revision without accidently to live APIs (use Revisions tab).

APIM = Developer Portal + Gateway + Azure portal

Gateway:

Developer porta is for the consumer of the API to test etc.

<https://docs.microsoft.com/en-au/azure/api-management/import-and-publish>

videos: <https://azure.microsoft.com/en-us/resources/videos/index/?services=api-management>

intro: <https://azure.microsoft.com/en-us/resources/videos/azure-friday-introducing-new-azure-api-management-capabilities/>

<https://azure.microsoft.com/en-us/resources/videos/ignite-2018-expose-apis-with-peace-of-mind-when-using-azure-api-management/> API Demo from 9 minutes

publish your api: go to APIM=> APIs (Select APIs from under API MANAGEMENT) => Add a new API => Swagger/ open API specification => select Open API file.

You can also use “API hosted on App Service”.

Select the API you created in the previous step => Test

Developer portal: Navigate to the Developer portal tab => click APIs => try it

To get access to the API, developers must first subscribe to a product

When they subscribe, they get a subscription key that is good for any API in that product.

If you created the APIM instance, you are an administrator already, so you are subscribed to every product.

<https://docs.microsoft.com/en-au/azure/api-management/import-and-publish>

<https://app.pluralsight.com/library/courses/microsoft-azure-api-management-essentials/table-of-contents>

<https://channel9.msdn.com/blogs/Cloud-and-Enterprise-Premium/Publish-and-Manage-APIs-with-Azure-API-Management> Demo at 20 minutes (may not good)

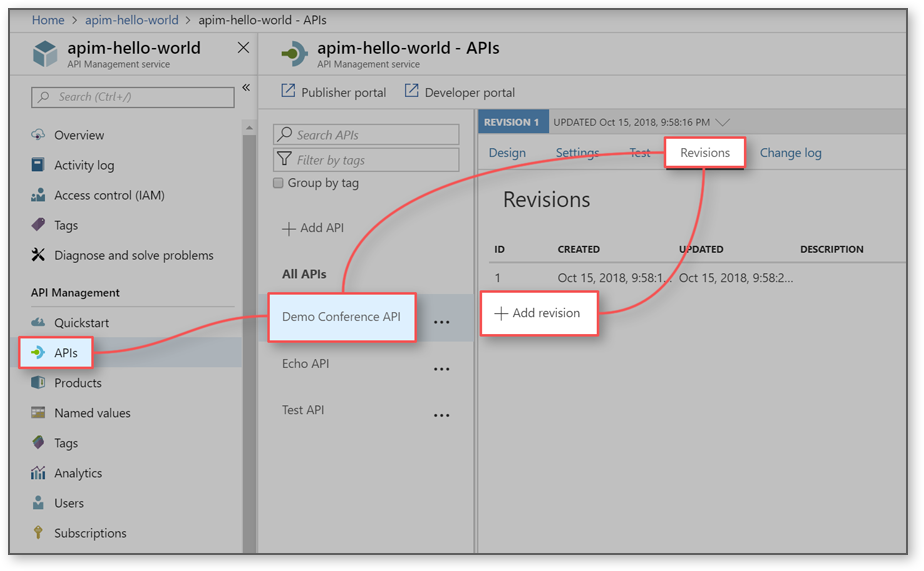
once swagger spec is imported, APIM API becomes a facade for the backend API. enables you to customize the facade according to your needs without touching the backend API.

Add a new revision/update:

<https://docs.microsoft.com/en-us/azure/api-management/api-management-get-started-revise-api>

go to APIs=> select api => in right panel go to Revisions tab => click + Add revision button => save => go to Design tab.

To approve the revision same step but => in the list => select the created Revision => click … Create Version from this Revision



We can create API operations and mocked/sample responses until developed. We can disable mock response. We can test API inside the portal.

**Approve revision/update**: Once tested we can publish the new revision by **going to Revisions tab** (go to APIs => select the version => Revisions tab (in right panel) ) click make current. Go to developer portal and refresh to see the new operation.

**APIM Integration with Azure Application Insights**: to monitor and troubleshoot your API.

Go to Application Insights tab => click Add to create a connection between APIM instance and Insight instance. Just select from dropdown and save.

You can also go to All APIM tab and settings to enable error log, sample response log etc.

Go to application map or live dash board, Metrics explorer, and keep clicking to view telemetry.

Go to Log Analytics tab to run queries like give me result where resultCode != 200 in last 24 hours.

**Swagger AND Swashbuckle**: Smart bear makes testing software adopted Swagger. It’s like WSDL for RESTful Web Services. It uses yaml (a human readable language) or Json.

Use Autorest to generate code from swagger. Swagger is like a contract (path, parameters, input, responses). It acts as documentation for API.

**Azure Resource Manager ARM:**

for beginner <https://www.youtube.com/watch?v=VQ_rixthPCI>

<https://www.youtube.com/watch?v=VQ_rixthPCI>

To download existing ARM go to Resource groups => your resource group => Automation script => Download => unzip

To clean downloaded ARM template <https://www.youtube.com/watch?v=myYTGsONrn0>

Use git while cleaning and validate before every commit. Reduce number of parameters instead use variables.

Create ARM project => Cloude => AzureResourceGroup => Blank Template

Add new item => DeploymentProject => Azure Resource Manager Deployment Template

Copy past or replace (only from **template.json to azuredeploy.json**) the downloaded ARM template. Right click the project => Validate => put/past default values in parameters dialog if required.

Note: do not replace power shell file.

If get SecurityError than sign the PowerShell script by going to folder => press shift => click open power shell window here => .\sign.ps1

Note: copy sign.ps1 file in the same folder. After every change sign again by .\sign.ps1

**Queue**:

<https://docs.microsoft.com/en-us/azure/storage/queues/storage-dotnet-how-to-use-queues>

<https://docs.microsoft.com/en-us/azure/storage/queues/storage-queues-introduction>

Common uses of Queue storage include:

* Creating a backlog of work to process asynchronously
* Passing messages from an Azure web role to an Azure worker role

The maximum time that a message can remain in the queue is seven days. A message, in any format, of up to 64 KB.

you can also use the **Azure storage emulator** to run and test your code in a local environment.

We use Storage Accounts for Blobs, Files, Tables, and Queues.

Create a storage account <https://docs.microsoft.com/en-gb/azure/storage/common/storage-quickstart-create-account?toc=%2Fazure%2Fstorage%2Fqueues%2Ftoc.json&tabs=azure-portal#create-a-storage-account-1>

select All services => type Storage Accounts => Select Storage Accounts => Add

we just need storage account and can create a queue in code before using it.

Azure NuGet packages are not yet updated for **.NET Core**. <https://ericflemingblog.wordpress.com/2017/01/26/getting-started-with-net-core-and-azure-queue-storage>

* NuGet "WindowsAzure.Storage" & "WindowsAzure.ConfigurationManager"

Use the Azure storage emulator for development and testing. <https://docs.microsoft.com/en-us/azure/storage/common/storage-use-emulator>

Regenerate your key by using the Azure portal if you believe it may have been compromised.

Your code de-queues a message from a queue in two steps. When you call GetMessage, you get the next message in a queue. A message returned from GetMessage becomes invisible to any other code reading messages from this queue. By default, this message stays invisible for 30 seconds. To finish removing the message from the queue, you must also call DeleteMessage. This two-step process of removing a message assures that if your code fails to process a message due to hardware or software failure, another instance of your code can get the same message and try again. Your code calls DeleteMessage right after the message has been processed.

**Service Bus**:

<https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-get-started-with-queues>

create a namespace: Create a resource => Integration => Service Bus => Create namespace

To create further rules with more constrained rights for regular senders and receivers See Service Bus authentication and authorization <https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-authentication-and-authorization>

Shared Access Signature (SAS): click the newly created namespace => click Shared access policies =>

click RootManageSharedAccessKey => click the copy button next to Primary Connection String

**Create a queue**: click the newly created namespace => click Queues => click + Queue => Enter the queue Name => click Create

Console application to send messages:

Add the Service Bus NuGet package “Microsoft.Azure.ServiceBus”

**Azure Key Vault**: Use Azure Key Vault to encrypt keys, tokens, certificates and small secrets like passwords. Applications have no direct access to keys. Monitor and audit your key use with Azure logging – pipe logs into Azure HDInsight or your security information and event management (SIEM).

<https://docs.microsoft.com/en-au/azure/key-vault/key-vault-get-started>

web app: <https://docs.microsoft.com/en-us/azure/key-vault/tutorial-net-create-vault-azure-web-app>

Azure CLI: open command

az login

az keyvault create --name "KeyVault1000" --resource-group "AzurePoc" --location "West US"

az keyvault secret set --vault-name "KeyVault1000" --name "AppKey" --value "password123"

to show/check: az keyvault secret show --name "AppKey" --vault-name "KeyVault1000"

Secret Identifier url: Go to KeyVault1000 => Secrets => AppKey => Current Version => Secret Identifier

Install these NuGet packages: “Microsoft.Azure.Services.AppAuthentication”, “Microsoft.Azure.KeyVault”

**Azure Storage Explorer**: with Azure Storage Explorer manage the contents of your storage account with Azure Storage Explorer. Upload, download, and manage blobs, files, queues, tables, and Cosmos DB entities.

**Secret Manager tool:** stores sensitive data for development work outside of your project tree. This approach helps prevent the accidental sharing of app secrets within source code.

<https://docs.microsoft.com/en-au/aspnet/core/security/app-secrets?view=aspnetcore-2.2&tabs=windows>

**Redis Caches**: <https://docs.microsoft.com/en-au/azure/azure-cache-for-redis/>

<https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-web-app-howto>

Videos: <https://www.youtube.com/results?search_query=azure+redis+cache>

ASP.NET Session State Provider for Azure Cache for Redis: <https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-aspnet-session-state-provider>

Azure Cache for Redis can cache both .NET objects and primitive data types (basic data types like int, string), but before a .NET object can be cached it must be serialized. Use JsonConvert.SerializeObject(obj) and JsonConvert.DeserializeObject<type>(obj).

to set or get a string value use StringSet or StringGet methods.

Locate your cache instance and your application in the same region.

Redis works best with smaller values.

Reuse connections - Creating new connections is expensive and increases latency, so reuse connections as much as possible.

Set an expiration value on your keys. See the Expire and ExpireAt commands for more details.

Install-Package StackExchange.Redis

**Redis Session State Provider**: Store ASP.NET session state in the cache.

<https://docs.microsoft.com/en-us/azure/azure-cache-for-redis/cache-aspnet-session-state-provider>

Azure Cache for Redis provides a session state provider that you can use to store your session state in-memory with Azure Cache for Redis instead of a SQL Server database.

Create Asp.Net Web App => Cloud => Asp.Net Web App (.Net Framework Standard) => MVC

Install-Package Microsoft.Web.RedisSessionStateProvider

**In Proc** – This is the default very basic Session mode which uses the Web Server itself to handle Session State. This is faster than other modes.