# Azure API Management

## What is Azure API Management?

Azure API Management is a fully managed cloud service that you can use to publish, secure, transform, maintain, and monitor APIs. It helps organizations publish APIs to external, partner, and internal developers to unlock the potential of their data and services. API Management handles all the tasks involved in mediating API calls, including request authentication and authorization, rate limit and quota enforcement, request and response transformation, logging and tracing, and API version management. API Management enables you to create and manage modern API gateways for existing backend services no matter where they're hosted.

### Microservices architecture challenges

Client apps are coupled to microservices. If you want to change the location or definition of the microservice, you may have to reconfigure or update the client app.

Each microservice may be presented under different domain names or IP addresses. This presentation can give an impression of inconsistency to users and can negatively affect your branding.

It can be difficult to enforce consistent API rules and standards across all microservices. For example, one team may prefer to respond with XML and another may prefer JSON.

You're reliant on individual teams to implement security in their microservice correctly. It's difficult to impose these requirements centrally.

### API Management also includes helpful tools

You can test each microservice and its operations to ensure that they behave in accordance with your requirements. You can also monitor the behavior and performance of deployed services.

Azure API Management supports importing Azure Function Apps as new APIs or appending them to existing APIs. The process automatically generates a host key in the Azure Function App, which is then assigned to a named value in Azure API Management.

Here are some examples of functionality that could be offloaded to a gateway:

* SSL termination
* Authentication
* IP allow/block list
* Client rate limiting (throttling)
* Logging and monitoring
* Response caching
* GZIP compression
* Servicing static content

## Components

The system is made up of the following components:

* The API gateway is the endpoint:
  + Accepts API calls and routes them to your backend(s).
  + Verifies API keys, JWT tokens, certificates, and other credentials.
  + Enforces usage quotas and rate limits.
  + Transforms your API on the fly without code modifications.
  + Caches backend responses when set up.
  + Logs call metadata for analytics purposes.
* The Azure portal
  + Is the administrative interface where you set up your API program
  + Define or import API schema.
  + Package APIs into products.
  + Set up policies like quotas or transformations on the APIs.
  + Get insights from analytics.
  + Manage users.
* The Developer portal

(Consumption tier services don't come with the developer portal.)

* + Serves as the main web presence for developers
  + Read API documentation.
  + Try out an API via the interactive console.
  + Create an account and subscribe to get API keys.
  + Access analytics on their own usage.

## Products

Products are how APIs are surfaced to developers. Products in API Management have one or more APIs, and are configured with a title, description, and terms of use.

Products can be **Open** or **Protected**. **Protected products must be subscribed** to before they can be used, while open products can be used without a subscription. Subscription approval is configured at the product level and can either require administrator approval, or be auto-approved.

## Groups

Groups are used to manage the visibility of products to developers. API Management has the following immutable system groups:

* Administrators –

Azure subscription administrators are members of this group. Administrators manage API Management service instances, creating the APIs, operations, and products that are used by developers.

* Developers –

Authenticated developer portal users fall into this group. Developers are the customers that build applications using your APIs. Developers are granted access to the developer portal and build applications that call the operations of an API.

* Guests –

Unauthenticated developer portal users, such as prospective customers visiting the developer portal of an API Management instance fall into this group. They can be granted certain read-only access, such as the ability to view APIs but not call them.

In addition to these system groups, administrators can create custom groups or leverage external groups in associated Azure Active Directory tenants.

## Developers

Developers represent the user accounts in an API Management service instance. Developers can be created or invited to join by administrators, or they can sign up from the Developer portal. Each developer is a member of one or more groups, and can subscribe to the products that grant visibility to those groups.

## Policies

Policies are a powerful capability of API Management that allow the Azure portal to change the behavior of the API through configuration. Policies are a collection of statements that are executed sequentially on the request or response of an API. Popular statements include format conversion from XML to JSON and call rate limiting to restrict the number of incoming calls from a developer, and many other policies are available.

## Create a new API in API Management from a function app

### Create functions

[Create an Azure Function App - ProductFunction](#_Create_a_function)

[Create ProductDetails function in your ProductFunction app](#_Create_a_function_1)

[Test ProductDetails function](#_Test_your_Azure)

### Expose function app as an API using Azure API Management

Azure Portal -> All resources -> ProductFunction function app

-> API -> API Management -> Create new

-> Install API Management gateway pane appears

“Subscription” = Concierge Subscription,

“Resource group” = [sandbox resource group name]

“Region” = select any that supports the Consumption Plan

“Administrator email” = Enter an email address.

“Pricing tier” = Consumption (99.95% SLA)

-> Next: Monitoring and clear the Application Insights option

-> Review + Create -> Create -> Link API

-> Import Azure Functions API Management service pane appears

-> Select “ProductDetails” function and Select to continue

-> “API URL suffix” = “products”

-> Create.

### Test the OnlineStore products endpoint

API Management pane of your function app -> Test -> GET ProductDetails

-> Under Query parameters, select Add parameter

-> Enter name in the NAME field and value and in the VALUE field

-> Send

-> The **HTTP response** section of the console has two tabs, Message and Trace. The Message tab is populated with the HTTP response. The product details appear in JSON format at the end of the response.

-> Scroll up to the HTTP request section and notice the format of the request. The request was sent to a destination in the azure-api.net domain. This location is different from the azurewebsites.net domain where the function app is hosted.

## Add another Azure Functions app to an existing API

### Create another function

[Create OrderDetails function in your ProductFunction app](#_Create_a_function_1)

[Test the OrderDetails function](#_Test_your_Azure)

### Add a function to an existing API

Azure Portal -> All resources -> OnlineStore API Management service

-> APIs -> APIs -> Create from Azure resource -> Function App

-> Browse -> Import Azure Functions pane appears

-> Select -> Select Azure Function App pane appears

-> Select OrderFunction\*\*\*\*\*\*\* App -> Select -> Select OrderDetails function -> Select

-> “API URL suffix” = “orders” -> Create

### Test the OnlineStore orders endpoint in the portal

[Test orders endpoint](#_Test_the_OnlineStore_1)

Notice that both the functions can now be called through endpoints within the **azure-api.net** domain (as defined by the GATEWAY\_URL), which is the domain used by Azure API Management.

In other Learn modules**, you can learn how to**

* + - **apply policies,**
    - **security settings,**
    - **external caches,**
    - **and other features to the functions in an API Management Gateway.**

**APIM gateway provides you with a central control point, where you can manage multiple microservices without altering their code.**

## API Management policies

Policies are a collection of Statements that are executed sequentially on the request or response of an API.

A policy can apply changes to both the inbound request and outbound response. Policy expressions can be used as attribute values or text values in any of the API Management policies, unless the policy specifies otherwise.

Example:

Filter response content:

<policies>

**<inbound>**

<base />

**</inbound>**

**<backend>**

<base />

**</backend>**

**<outbound>**

<base />

<choose>

<when condition="@(context.Response.StatusCode == 200 && context.Product.Name.Equals("Starter"))">

<set-body>

@{

var response = context.Response.Body.As<JObject>();

foreach (var key in new [] {"minutely", "hourly", "daily", "flags"})

{ response.Property (key).Remove (); }

return response.ToString();

}

</set-body>

</when>

</choose>

**</outbound>**

**<on-error>**

<base />

**</on-error>**

</policies>

<!-- NOTE that we are not using preserveContent=true when deserializing response body stream into a JSON object since we don't intend to access it again. See details on https://docs.microsoft.com/azure/api-management/api-management-transformation-policies#SetBody -->

## Create advanced policies

### Control flow

<choose>

<when condition="Boolean expression | Boolean constant">

<!— one or more policy statements to be applied if the above condition is true -->

</when>

<when condition="Boolean expression | Boolean constant">

<!— one or more policy statements to be applied if the above condition is true -->

</when>

<otherwise>

<!— one or more policy statements to be applied if none of the above conditions are true -->

</otherwise>

</choose>

### Forward request

<forward-request timeout="time in seconds" follow-redirects="true | false"/>

Removing this policy results in the request not being forwarded to the backend service and the policies in the outbound section are evaluated immediately upon the successful completion of the policies in the inbound section.



### Limit concurrency

The **limit-concurrency** policy prevents enclosed policies from executing by more than the specified number of requests at any time. Upon exceeding that number, new requests will fail immediately with a 429 Too Many Requests status code.

<limit-concurrency key="expression" max-count="number">

<!— nested policy statements -->

</limit-concurrency>

### Log to Event Hub

The **log-to-eventhub** policy sends messages in the specified format to an Event Hub defined by a Logger entity. As its name implies, the policy is used for saving selected request or response context information for online or offline analysis.

<log-to-eventhub logger-id="id of the logger entity" partition-id="index of the partition where messages are sent" partition-key="value used for partition assignment">

Expression returning a string to be logged

</log-to-eventhub>

### Mock response

The **mock-response**, as the name implies, is used to mock APIs and operations. It aborts normal pipeline execution and returns a mocked response to the caller. The policy always tries to return responses of highest fidelity.

It prefers response content examples, whenever available. It generates sample responses from schemas, when schemas are provided and examples are not. If neither examples or schemas are found, responses with no content are returned.

<mock-response status-code="code" content-type="media type"/>

### Retry

The **retry** policy executes its child policies once and then retries their execution until the retry condition becomes false or retry count is exhausted.

<retry

condition="boolean expression or literal"

count="number of retry attempts"

interval="retry interval in seconds"

max-interval="maximum retry interval in seconds"

delta="retry interval delta in seconds"

first-fast-retry="boolean expression or literal">

<!-- One or more child policies. No restrictions -->

</retry>

### Return response

The **return-response** policy aborts pipeline execution and returns either a default or custom response to the caller. Default response is 200 OK with no body. Custom response can be specified via a context variable or policy statements. When both are provided, the response contained within the context variable is modified by the policy statements before being returned to the caller.

<return-response response-variable-name="existing context variable">

<set-header/>

<set-body/>

<set-status/>

</return-response>

**Additional resources**

* Visit [API Management policies](https://docs.microsoft.com/en-us/azure/api-management/api-management-policies) for more policy examples.
* [Error handling in API Management policies](https://docs.microsoft.com/en-us/azure/api-management/api-management-error-handling-policies).

# Call an API

Applications must include a valid key in all HTTP requests when they make calls to API endpoints that are protected by a subscription. Keys can be passed in the request header, or as a query string in the URL.

The default header name is Ocp-Apim-Subscription-Key, and the default query string parameter name is subscription-key.

curl --header "Ocp-Apim-Subscription-Key: <key string>" https://<apim gateway>.azure-api.net/api/path

**OR**

curl https://<apim gateway>.azure-api.net/api/path?subscription-key=<key string>

If the key is not passed in the header, or as a query string in the URL, you'll get a 401 Access Denied response from the API gateway.

**ALSO**

[Secure APIs by using certificates](https://docs.microsoft.com/en-us/learn/modules/explore-api-management/7-secure-access-api-certificates)