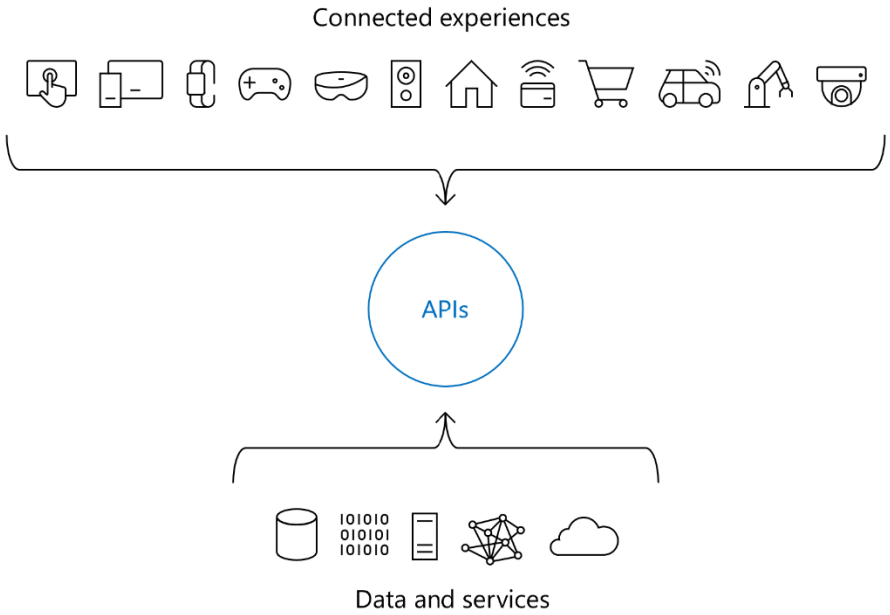
Azure API Management

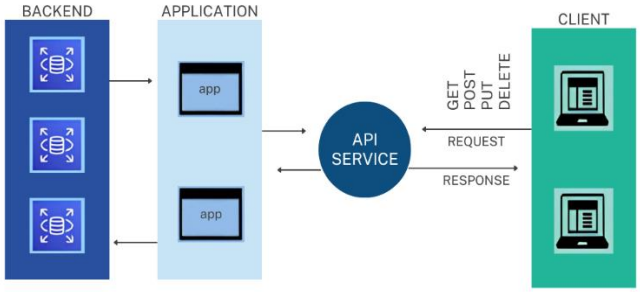
Azure API Management is a hybrid, multicloud management platform for APIs across all environments. As a platform-as-a-service, API Management supports the complete API lifecycle.

APIs enable digital experiences, simplify application integration, underpin new digital products, and make data and services reusable and universally accessible. With the proliferation and increasing dependency on APIs, organizations need to manage them as first-class assets throughout their lifecycle.



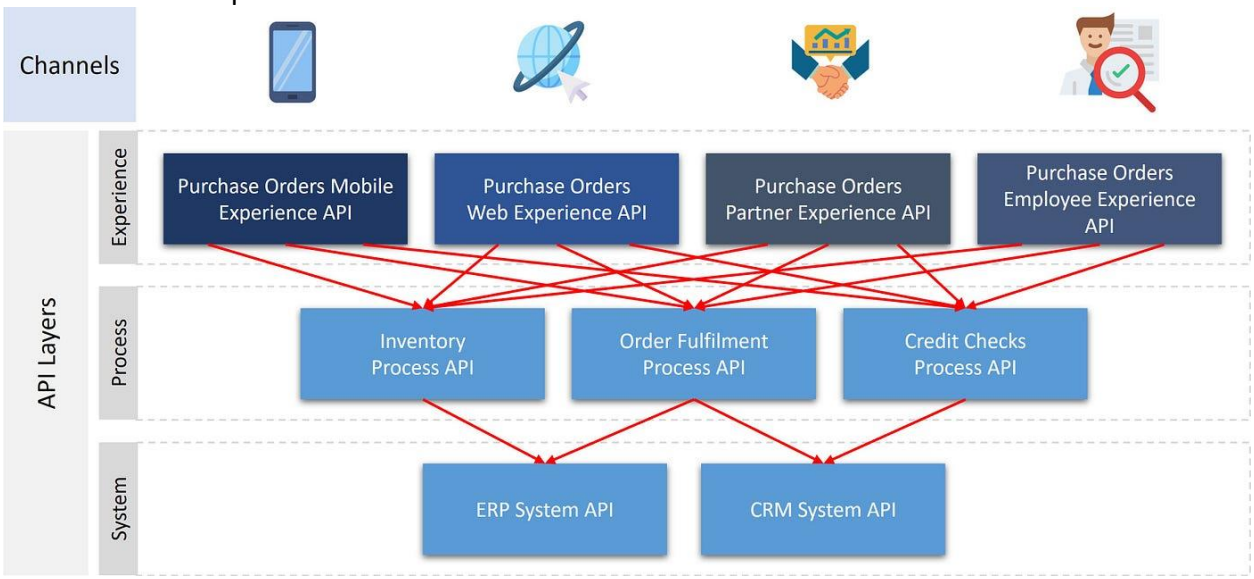
Common scenarios include:

- **Unlocking legacy assets** - APIs are used to abstract and modernize legacy backends and make them accessible from new cloud services and modern applications. APIs allow innovation without the risk, cost, and delays of migration.

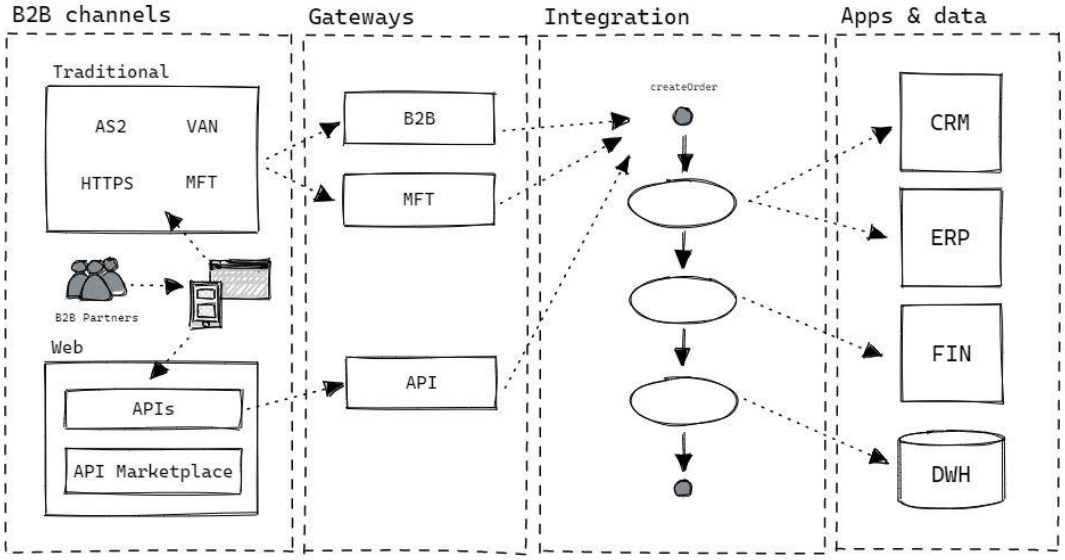


- **API-centric app integration** - APIs are easily consumable, standards-based, and self-describing mechanisms for exposing and accessing data, applications, and processes. They simplify and reduce the cost of app integration.

- **Multi-channel user experiences** - APIs are frequently used to enable user experiences such as web, mobile, wearable, or Internet of Things applications. Reuse APIs to accelerate development and ROI.



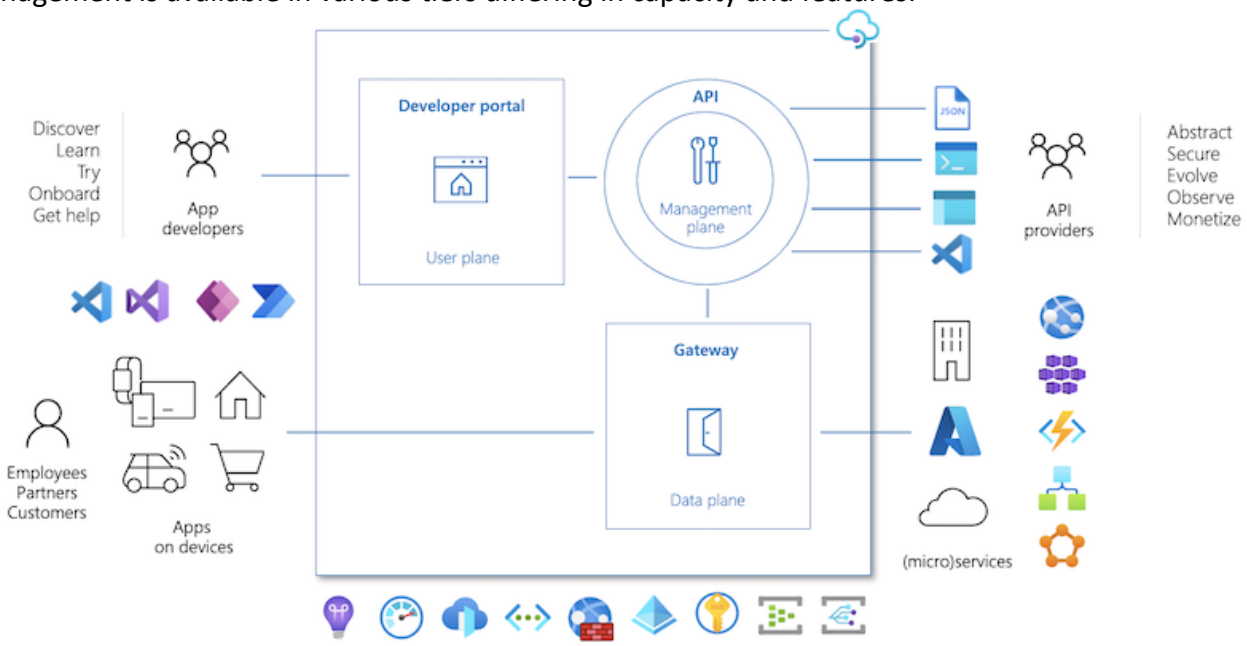
- **B2B integration** - APIs exposed to partners and customers lower the barrier to integrate business processes and exchange data between business entities. APIs eliminate the overhead inherent in point-to-point integration. Especially with self-service discovery and onboarding enabled, APIs are the primary tools for scaling B2B integration.



1.1 API Management components

Azure API Management is made up of an API *gateway*, a *management plane*, and a *developer portal*. These components are Azure-hosted and fully managed by default.

API Management is available in various tiers differing in capacity and features.



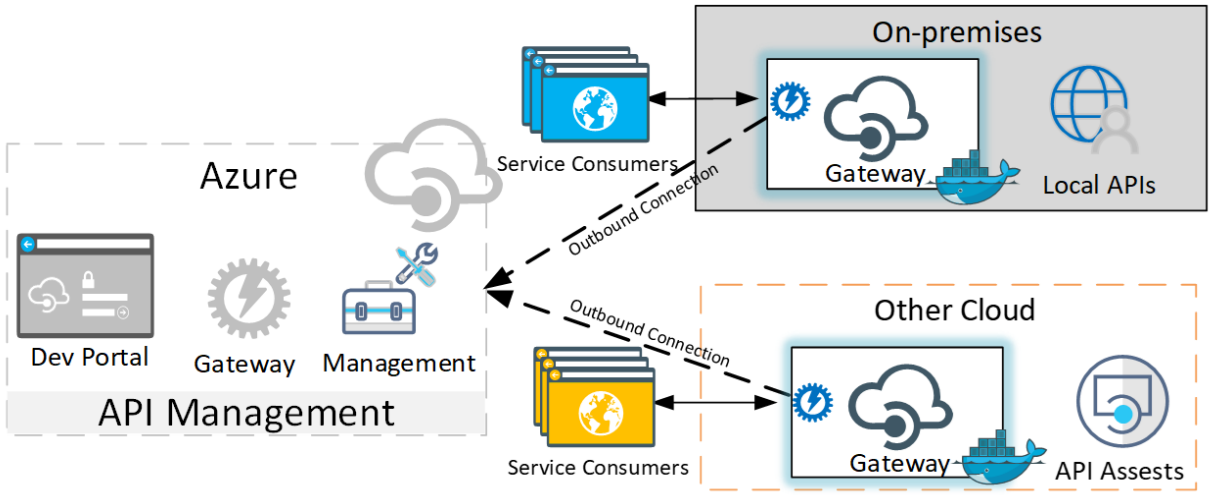
1.2 API gateway

All requests from client applications first reach the API gateway, which then forwards them to respective backend services. The API gateway acts as a facade to the backend services, allowing API providers to abstract API implementations and evolve backend architecture without impacting API consumers. The gateway enables consistent configuration of routing, security, throttling, caching, and observability.
Specifically, the gateway:

- Acts as a facade to backend services by accepting API calls and routing them to appropriate backends
- Verifies API keys and other credentials such as JWT tokens and certificates presented with requests
- Enforces usage quotas and rate limits
- Optionally transforms requests and responses as specified in policy statements
- If configured, caches responses to improve response latency and minimize the load on backend services
- Emits logs, metrics, and traces for monitoring, reporting, and troubleshooting

1.3 Self-hosted gateway

With the self-hosted gateway, customers can deploy the API gateway to the same environments where they host their APIs, to optimize API traffic and ensure compliance with local regulations and guidelines. The self-hosted gateway enables customers with hybrid IT infrastructure to manage APIs hosted on-premises and across clouds from a single API Management service in Azure.



References :

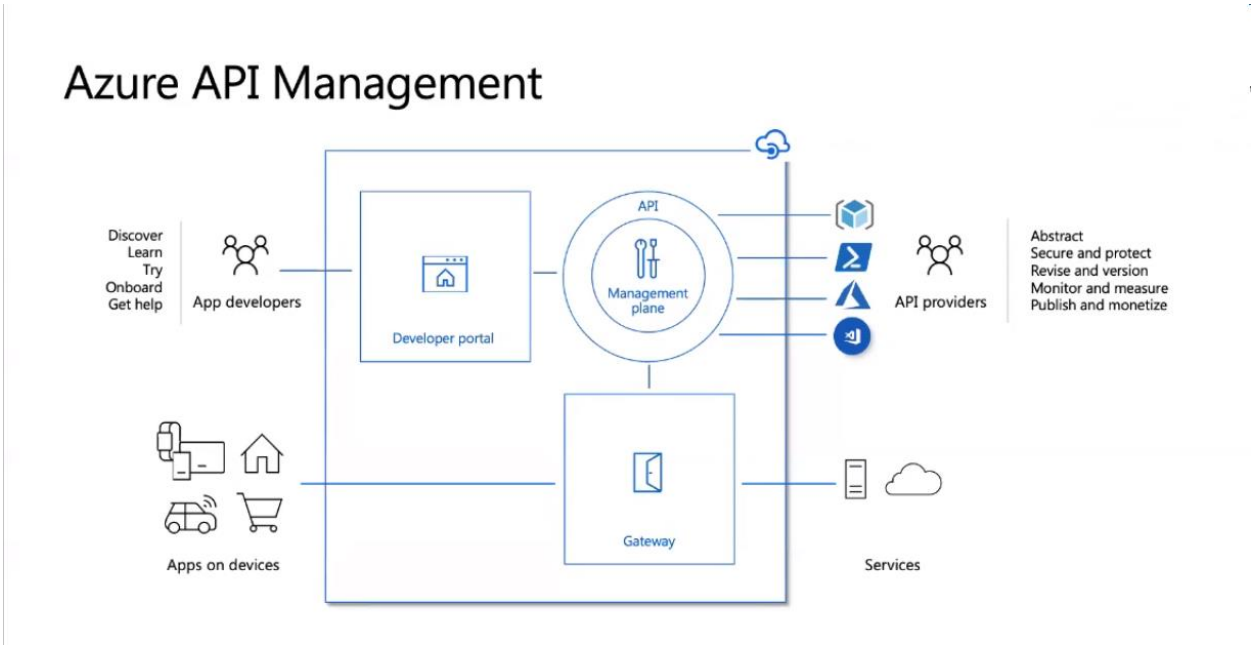
1. https://tointegrationandbeyond.com/blogs/index.php/2021/08/12/azure-apim-self-hosted-gateway-misconceptions/
2. https://soltisweb.com/blog/detail/2021-04-22-deployinganazureapimself-hostedgateway

The self-hosted gateway is packaged as a Linux-based Docker container and is commonly deployed to Kubernetes, including to Azure Kubernetes Service and Azure Arc-enabled Kubernetes.

1.4 Management plane

API providers interact with the service through the management plane, which provides full access to the API Management service capabilities.
Use the management plane to:

- Provision and configure API Management service settings
- Define or import API schemas from a wide range of sources, including OpenAPI, WSDL, and OData definitions, Azure compute services, and WebSocket, GraphQL, and gRPC backends
- Package APIs into products
- Set up policies like quotas or transformations on the APIs
- Get insights from analytics
- Manage users



APIM instances can be updated or altered using the Management plan which can be accessed from different tools like VS Code extension, Azure portal, PowerShell, ARM templates.

Observability can be achieved by integrating API with Azure Monitor, Azure application insight, and Azure Event Hubs.

Observability options

Tech	Reporting	Monitoring	Debugging	Data lag	Retention	Sampling	Data schema	Data kind	Enabled
API inspector	-	-	Good	Instant	Last 100 traces	Turned on per request	Fixed can be extended	Request trace	Always
Built-in reports	Good	-	-	Minutes	Unspecified	100%	Fixed	Reports Logs via API	Always
Azure Monitor Metrics	Basic	Good	-	Minutes	93 days export to extend	100%	Fixed	Metrics	Always
Azure Monitor Logs	Good	Good	Good	Minutes	31 day (5GB) upgrade to extend	100% adjustable	Fixed can be extended	Logs	Optional
Application Insights	Good	Good	Good	Seconds	90 days (5GB) upgrade to extend	Custom	Choice of presets can be extended	Logs, metrics	Optional
Log to Event Hub	Custom	Custom	Custom	Seconds	User managed	Custom	Custom	Logs	Optional

References: - https://turbo360.com/blog/azure-apim-developer-portal

1.5 Developer portal

The open-source developer portal is an automatically generated, fully customizable website with documentation of your APIs.
API providers can customize the look and feel of the developer portal by adding custom content, customizing styles, and adding their branding.

Using the developer portal, developers can:

- Read API documentation
- Call an API via the interactive console
- Create an account and subscribe to get API keys
- Access analytics on their own usage
- Download API definitions
- Manage API keys

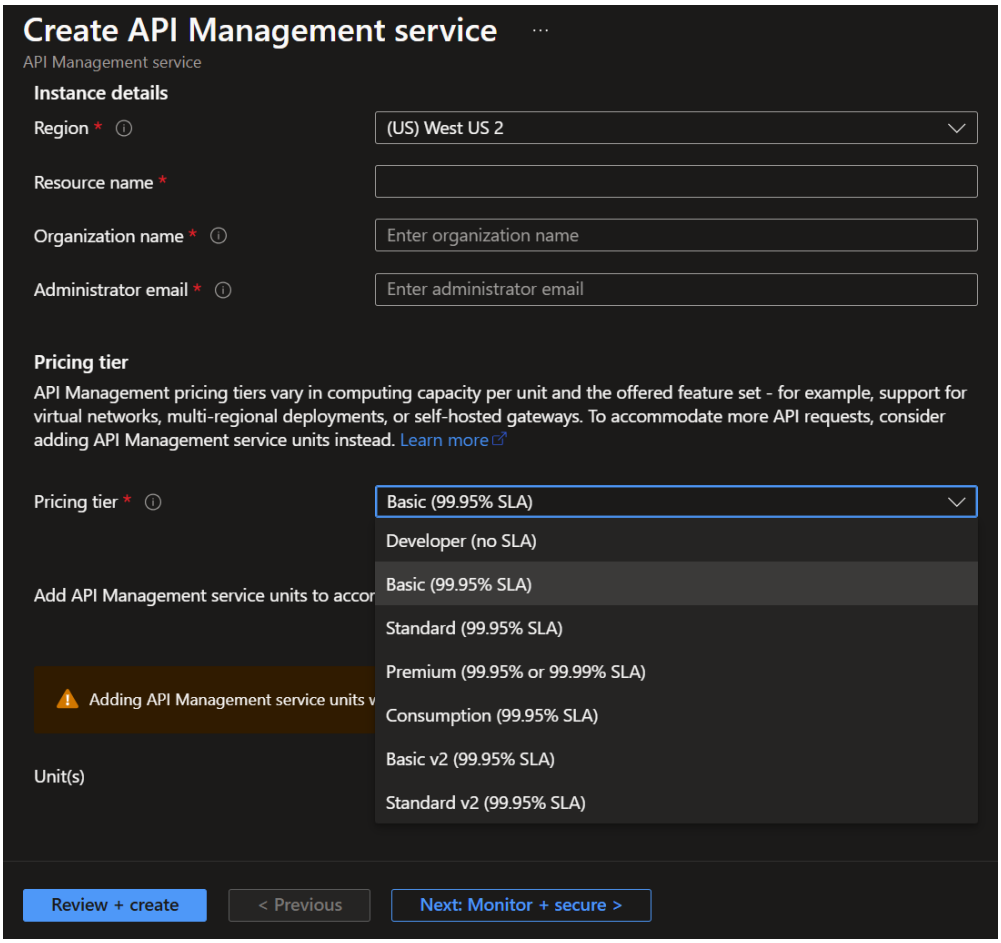
1.6 API Management tiers

Classic - The original API Management offering, including the Developer, Basic, Standard, and Premium tiers.

- The *Premium tier* is designed for enterprises requiring access to private backends, enhanced security features, multi-region deployments, availability zones, and high scalability.
- The *Developer tier* is an economical option for non-production use,
  while the *Basic, Standard, and Premium tiers* are production-ready tiers.

V2 - A new set of tiers that offer fast provisioning and scaling, including Basic v2 for development and testing, and Standard v2 for production workloads.
Standard v2 supports simplified connection to network-isolated backends.

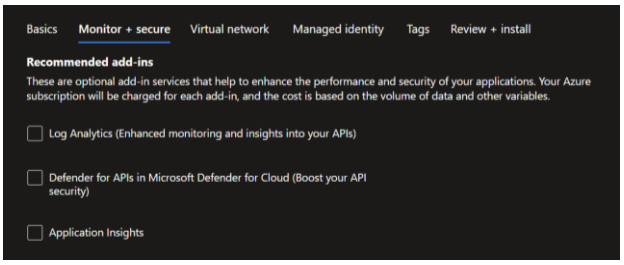
Consumption - The Consumption tier is a serverless gateway for managing APIs that scales based on demand and billed per execution.
It is designed for applications with serverless compute, microservices-based architectures, and those with variable traffic patterns.



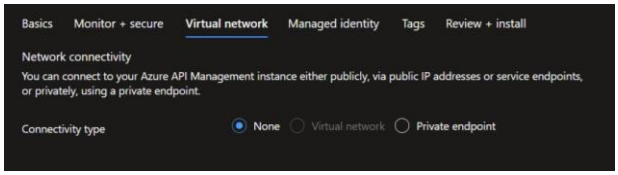
1.7 Integration with Azure services

API Management integrates with many complementary Azure services to create enterprise solutions, including:

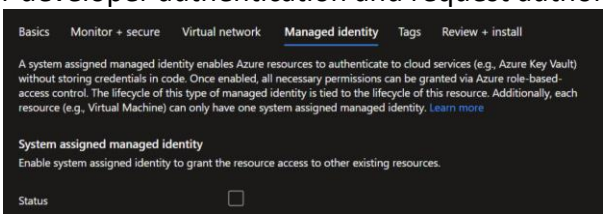
- **Azure API Center** to build a complete inventory of APIs in the organization - regardless of their type, lifecycle stage, or deployment location - for API discovery, reuse, and governance
- **Copilot** in Azure to help author API Management policies or explain already configured policies
- **Azure Key Vault** for secure safekeeping and management of client certificates and secrets
- **Azure Monitor** for logging, reporting, and alerting on management operations, systems events, and API requests
- **Application Insights** for live metrics, end-to-end tracing, and troubleshooting



- **Virtual networks, private endpoints, Application Gateway, and Azure Front Door** for network-level protection



- **Azure Defender** for APIs and Azure DDoS Protection for runtime protection against malicious attacks
- **Microsoft Entra ID** for developer authentication and request authorization



- **Event Hubs** for streaming events
- Several Azure compute offerings commonly used to build and host APIs on Azure, including Functions, Logic Apps, Web Apps, Service Fabric, and others including Azure OpenAI service.

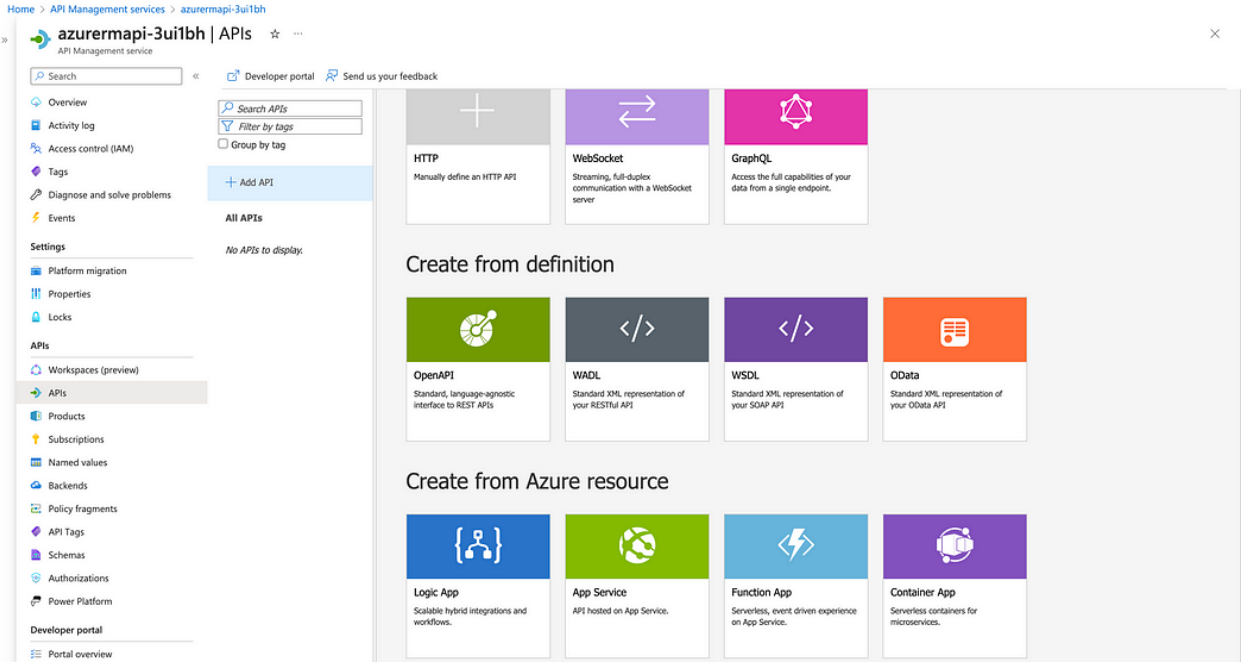
Add an API manually to the API Management instance. When you want to mock the API, you can create a blank API or define it manually.

2.1 Create an API

1. Navigate to your API Management service in the Azure portal and select **APIs** from the menu.
2. From the left menu, select **+ Add API**.

3. Select **HTTP** from the list.

4. Enter the backend **Web service URL** (for example, https://httpbin.org) and other settings for the API. The settings are explained in the Import and publish your first API tutorial.
5. Select **Create**.



Add and test an operation

Add a "/get" operation to map it to the back end "http://httpbin.org/get" operation.

Add and test a parameterized operation

This section shows how to add an operation that takes a parameter. In this case, we map the operation to "http://httpbin.org/status/200".

Add and test a wildcard operation

This section shows how to add a wildcard operation. A wildcard operation lets you pass an arbitrary value with an API request. Instead of creating separate GET operations as shown in the previous sections, you could create a wildcard GET operation.

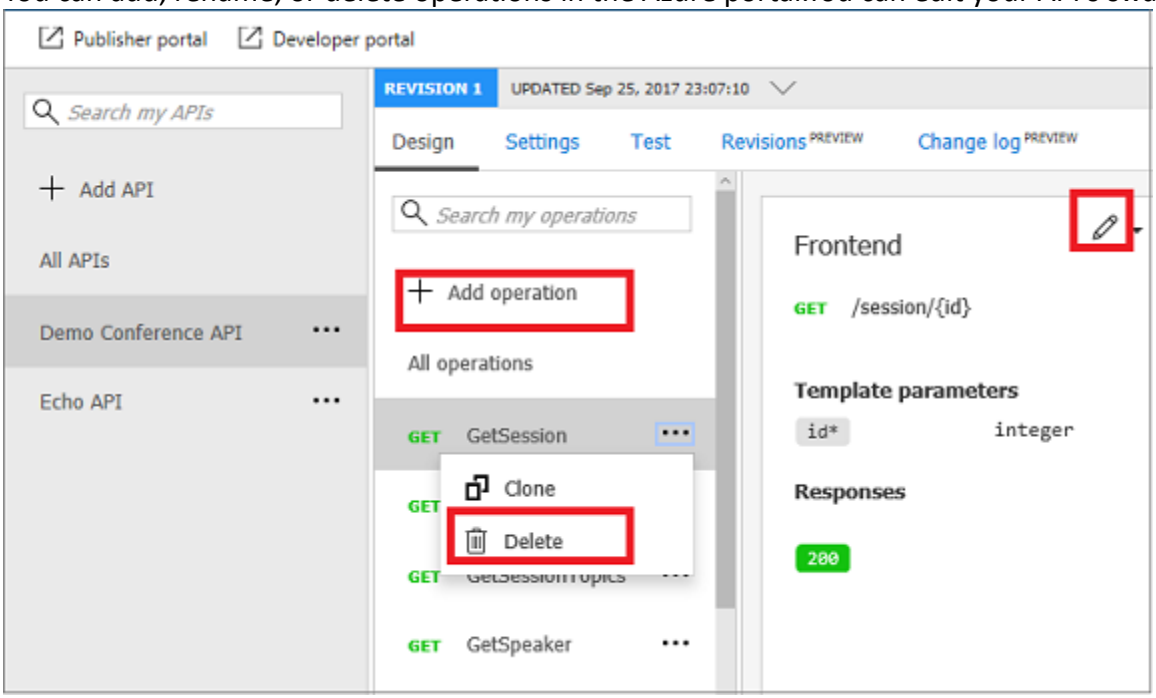
References : - https://medium.com/@abhimanyubajaj98/creating-your-first-azure-api-with-python-azure-web-app-and-terraform-using-azure-api-management-433490d78c79

Append other APIs

You can compose an API of APIs exposed by different services, including:

- An OpenAPI specification
- A SOAP API
- A GraphQL API
- A Web App hosted in Azure App Service
- Azure Function App
- Azure Logic Apps
- Azure Service Fabric

You can add, rename, or delete operations in the Azure portal.You can edit your API's swagger.

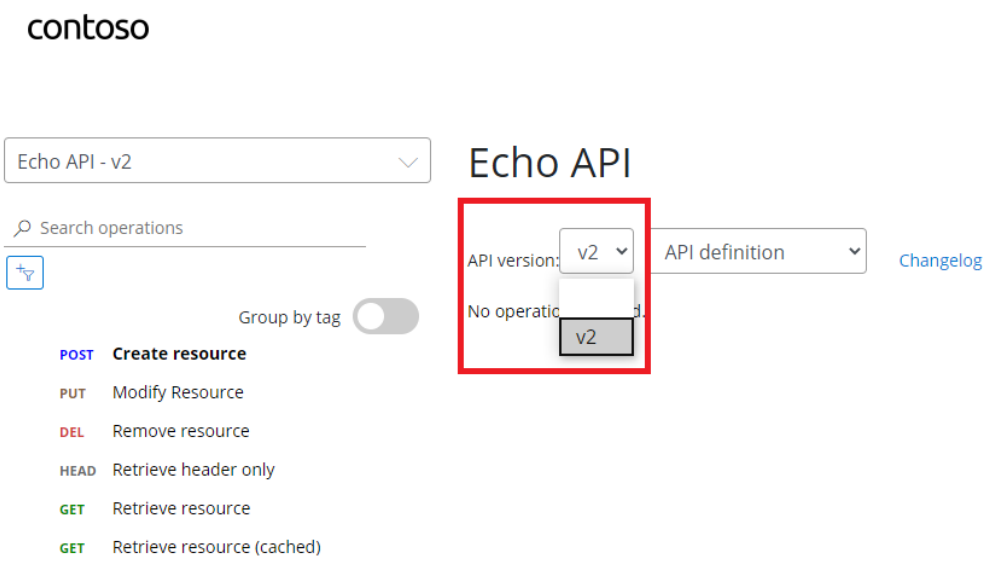


Clients can choose to use your new API version when they're ready, while existing clients continue to use an older version.

Publish multiple versions of your API at the same time.

Use a path, query string, or header to differentiate between versions.

Use any string value you wish to identify your version, which could be a number, a date, or a name.



Show your API versions grouped together on the developer portal.
Take an existing (non-versioned) API, and create a new version of it without breaking existing clients.

2.2 Backends in API Management

A *backend* (or *API backend*) in API Management is an HTTP service that implements your front-end API and its operations.
Configure and manage backend entities in the Azure portal or using Azure APIs or tools.

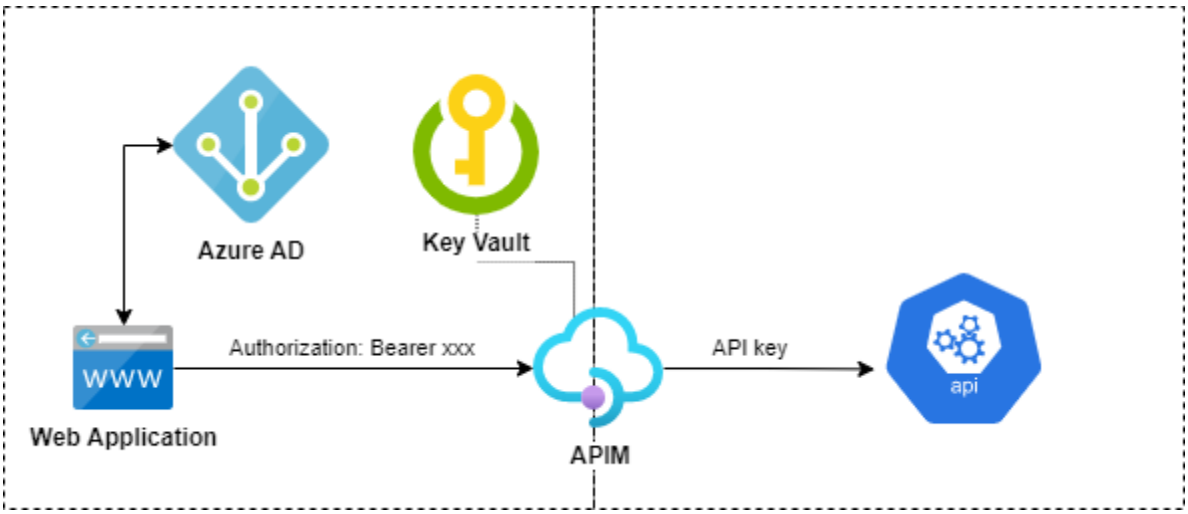
When importing certain APIs, API Management configures the API backend automatically. For example, API Management configures the backend web service when importing:

- An OpenAPI specification.
- A SOAP API.
- Azure resources, such as an HTTP-triggered Azure Function App or Logic App.

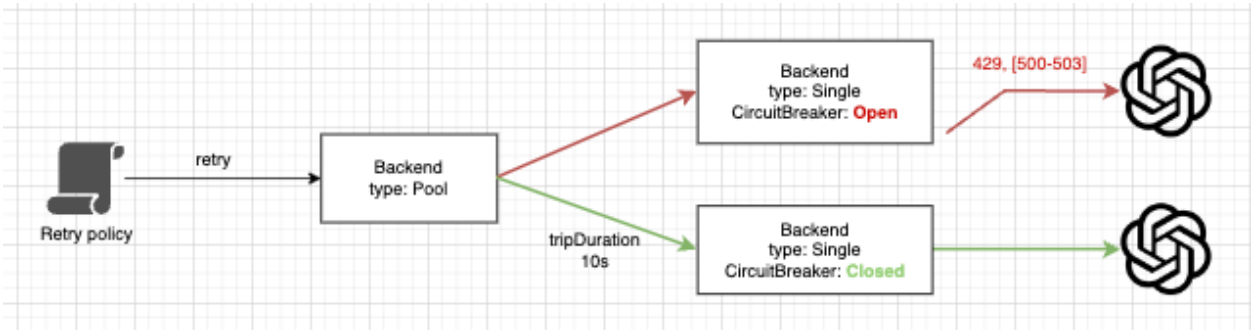
API Management also supports using other Azure resources as an API backend, such as:

- A Service Fabric cluster.
- A custom service.

Authorize the credentials of requests to the backend service



Take advantage of API Management functionality to maintain secrets in Azure Key Vault if named values are configured for header or query parameter authentication.

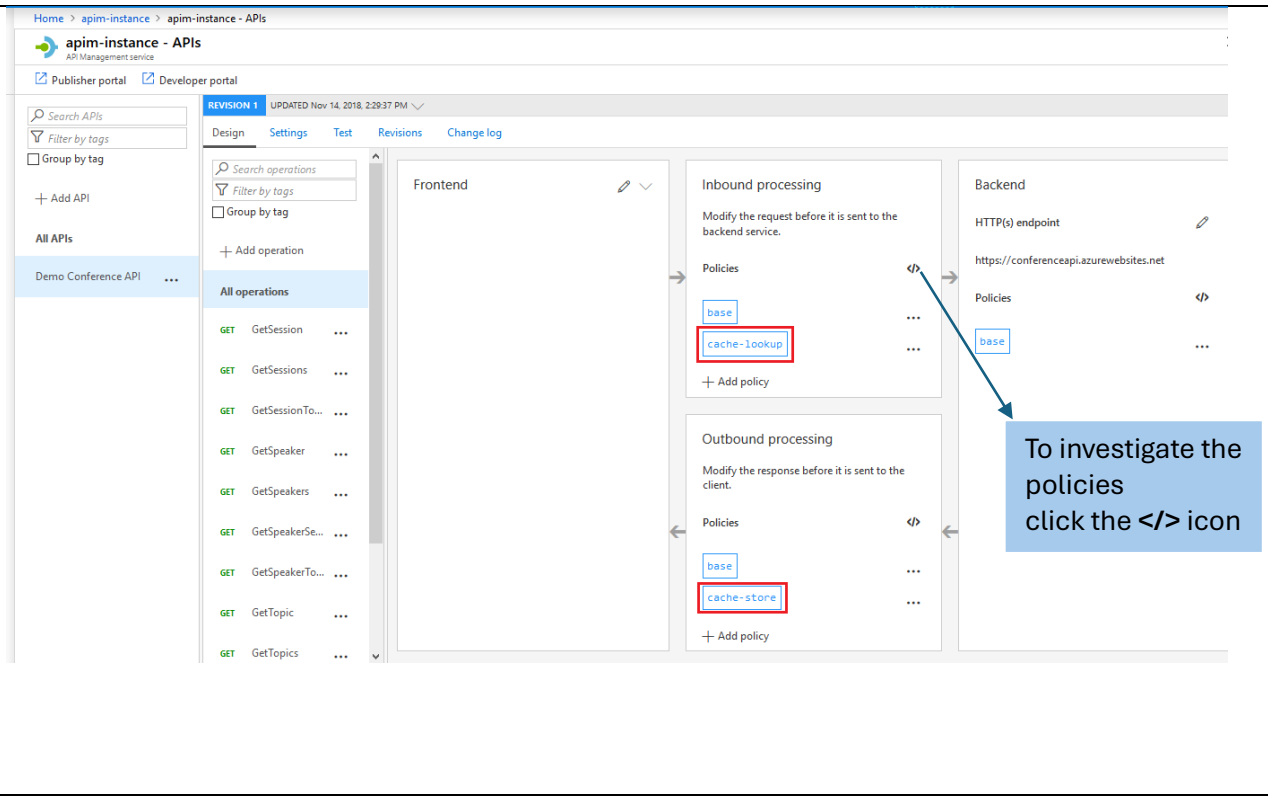


References :- https://techcommunity.microsoft.com/t5/apps-on-azure-blog/openai-at-scale-maximizing-api-management-through-effective/ba-p/4240317

Define circuit breaker rules to protect your backend from too many requests. Route or load-balance requests to multiple backends.

2.3 Add caching

APIs and operations in API Management can be configured with response caching. Response caching can significantly reduce latency for API callers and backend load for API providers.



8. In the **inbound** element, add the following policy:

Copy

```
<cache-lookup vary-by-developer="false" vary-by-developer-groups="false">
    <vary-by-header>Accept</vary-by-header>
    <vary-by-header>Accept-Charset</vary-by-header>
    <vary-by-header>Authorization</vary-by-header>
</cache-lookup>
```

9. In the **outbound** element, add the following policy:

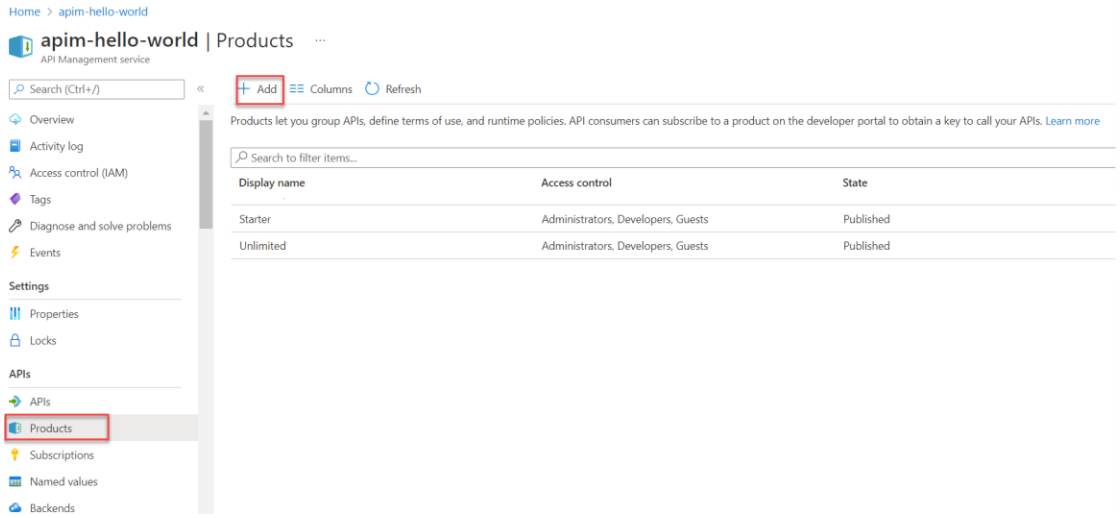
Copy

```
<cache-store duration="20" />
```

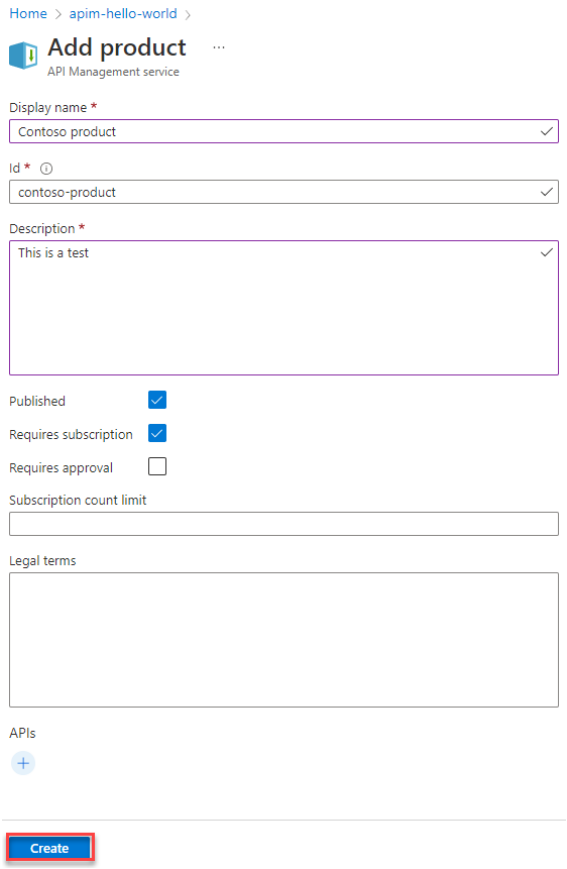
Duration specifies the expiration interval of the cached responses. In this example, the interval is **20** seconds.

In Azure API Management, a *product* contains one or more APIs, a usage quota, and the terms of use. After a product is published, *developers* can subscribe to the product and begin to use the product's APIs.

- Create and publish a product



- Add an API to the product



- Access product APIs

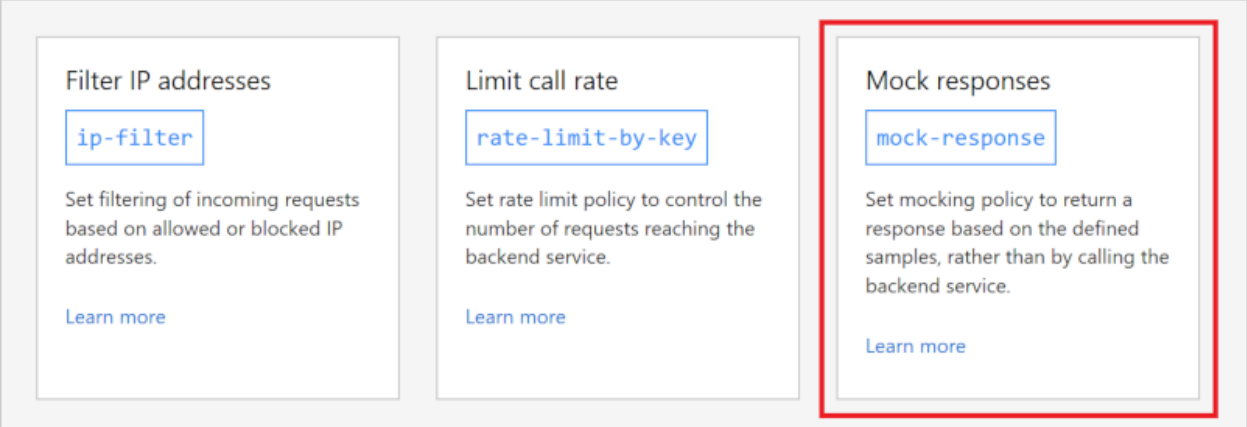
Mock API responses

The ability to mock up responses is useful in many scenarios:

- When the API façade is designed first and the backend implementation comes later. Or, the backend is being developed in parallel.
- When the backend is temporarily not operational or not able to scale.

Create a test API
Add an operation to the test API
Enable response mocking
Test the mocked API

Enable response mocking In the Inbound processing window, select **+ Add policy**.



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

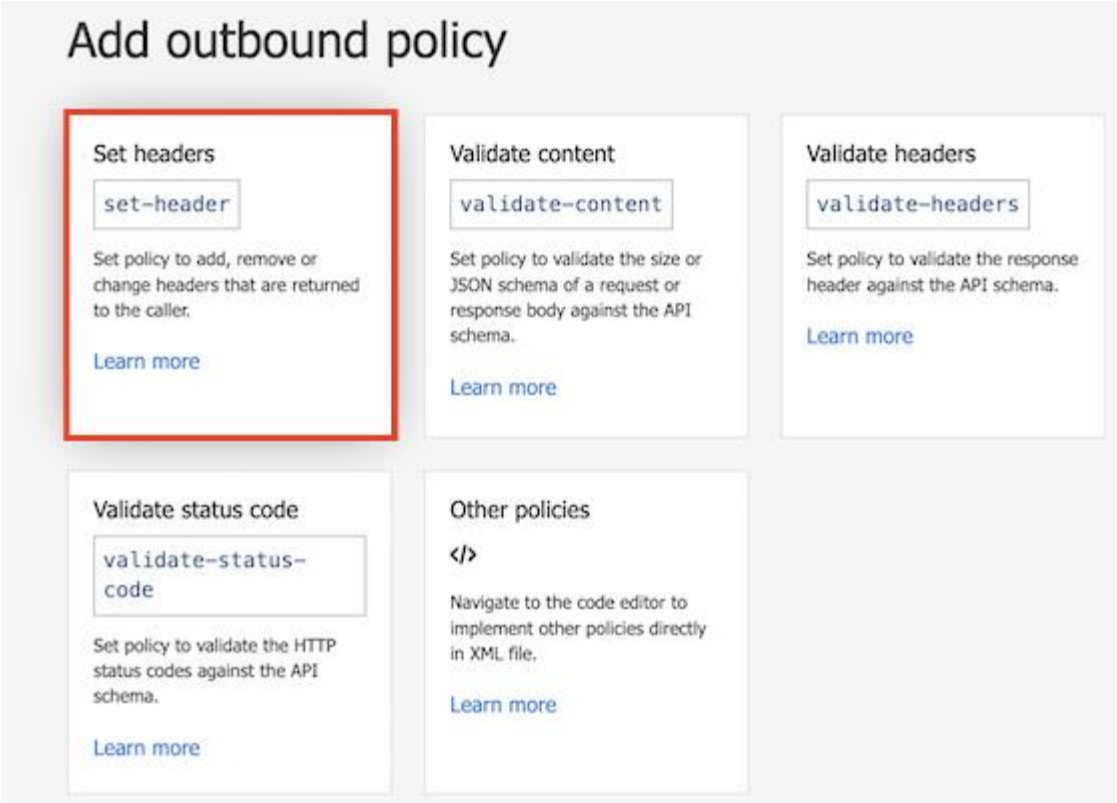
It prefers response content examples, when available. It generates sample responses from schemas, when schemas are provided, and examples aren't. If neither example or schemas are found, responses with no content are returned.
Usage

- Policy sections: inbound, outbound, on-error
- Policy scopes: global, workspace, product, API, operation
- Gateways: classic, v2, consumption, self-hosted, workspac

Transform and protect your API

You might want to transform your API so it doesn't reveal private backend info. Transforming an API can help you hide the technology stack info that's running in the backend, or hide the original URLs that appear in the body of the API's HTTP response.

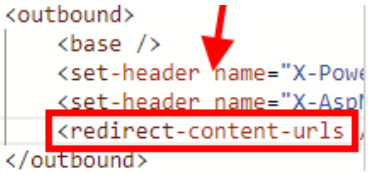
- Transform an API to strip response headers - For example, delete the following headers in the HTTP response:- X-Powered-By & X-AspNet-Version



The set-header policy assigns a value to an existing HTTP response and/or request header or adds a new response and/or request header.


```
<!-- Customize the responses -->
<outbound>
    <base />
    <set-header name="X-Powered-By" exists-action="delete" />
    <set-header name="X-AspNet-Version" exists-action="delete" />
</outbound>
<!-- Handle exceptions and customize error responses  -->
```

The redirect-content-urls policy rewrites (masks) links in the response body. Use in the outbound section to rewrite response body links to the backend service to make them point to the gateway instead.



Usage

- Policy sections: inbound, outbound
- Policy scopes: global, workspace, product, API, operation
- Gateways: classic, v2, consumption, self-hosted, workspace

For example, you might do this to hide URLs of the original backend service when they appear in the response.

- Replace original URLs in the body of the API response with API Management gateway URLs
- Protect an API by adding a rate limit policy (throttling) – Inbound Processing

```
<rate-limit-by-key calls="3" renewal-period="15" counter-key="@(context.Subscription.Id)" />
```

- Test the transformations

3.1 Authentication and authorization to APIs in Azure API Management

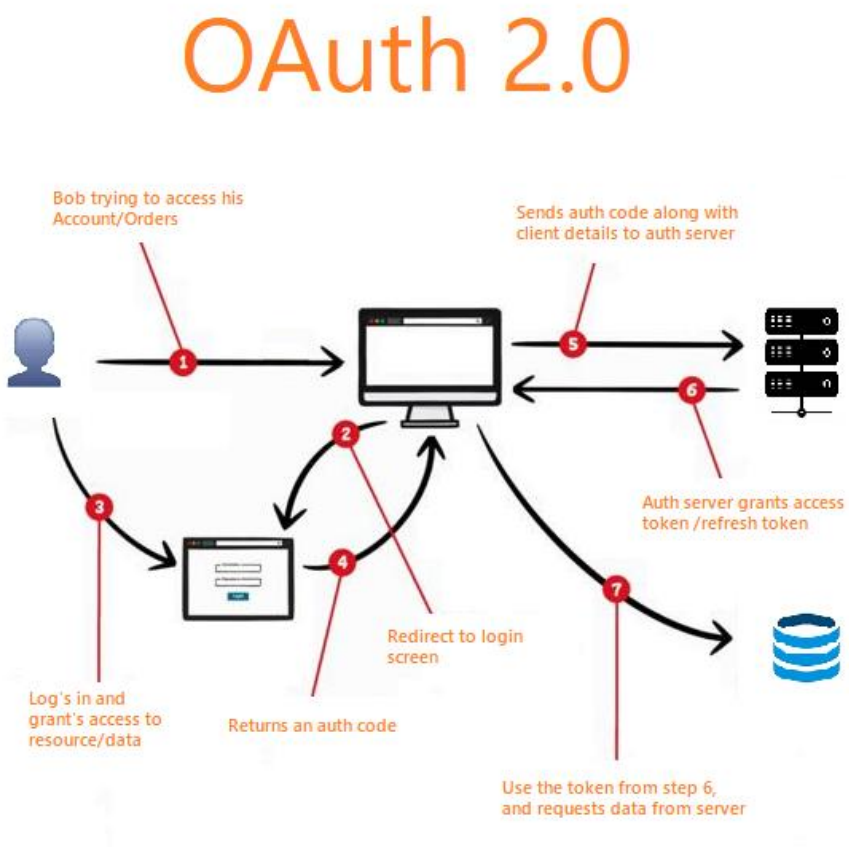
API authentication and authorization in API Management involve securing the end-to-end communication of client apps to the API Management gateway and through to backend APIs.

API Management supports OAuth 2.0 authorization between the client and the API Management gateway, between the gateway and the backend API, or both independently.

API Management supports other client-side and service-side authentication and authorization mechanisms that supplement OAuth 2.0 or that are useful when OAuth 2.0 authorization for APIs isn't possible.

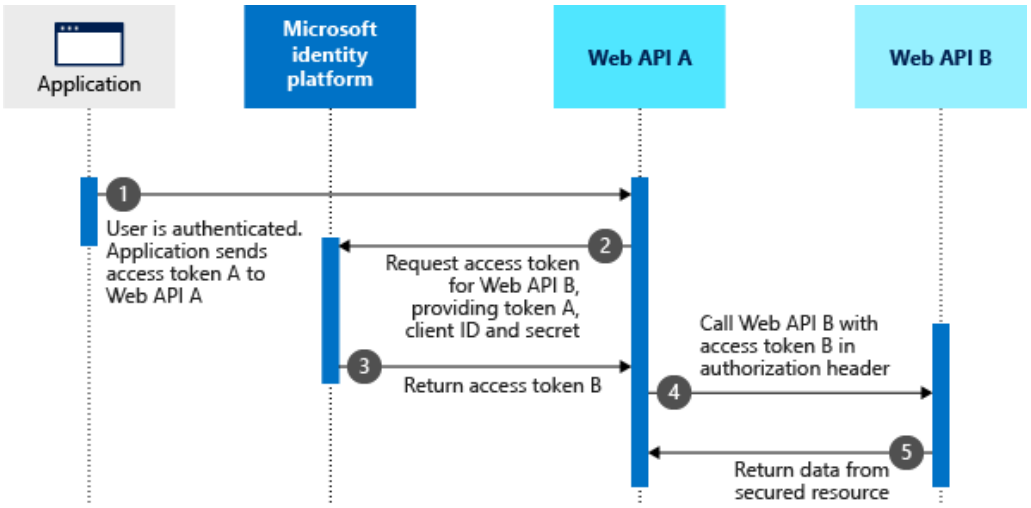
Authentication - The process of verifying the identity of a user or app that accesses the API. Authentication may be done through credentials such as username and password, a certificate, or through single sign-on (SSO) or other methods.

Authorization - The process of determining whether a user or app has permission to access a particular API, often through a token-based protocol such as OAuth 2.0.



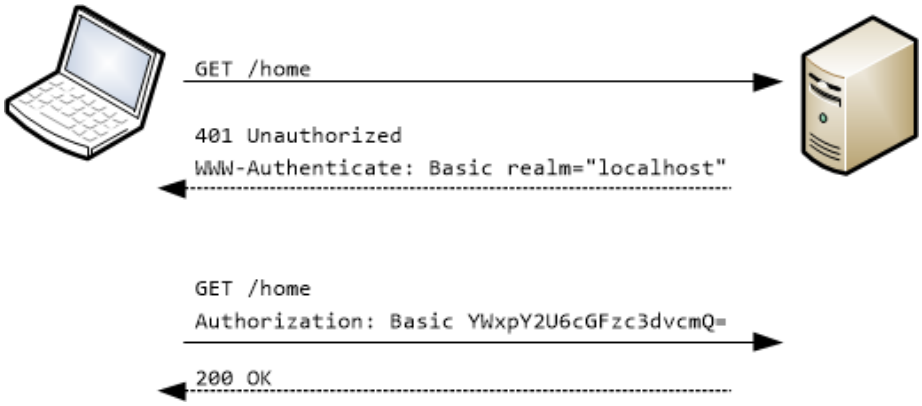
The client (the calling app, or bearer) authenticates using credentials to an identity provider.

The client obtains a time-limited access token (a JSON web token, or JWT) from the identity provider's authorization server.



The identity provider (for example, Microsoft Entra ID) is the issuer of the token, and the token includes an audience claim that authorizes access to a resource server (for example, to a backend API, or to the API Management gateway itself).

The client calls the API and presents the access token - for example, in an Authorization header.



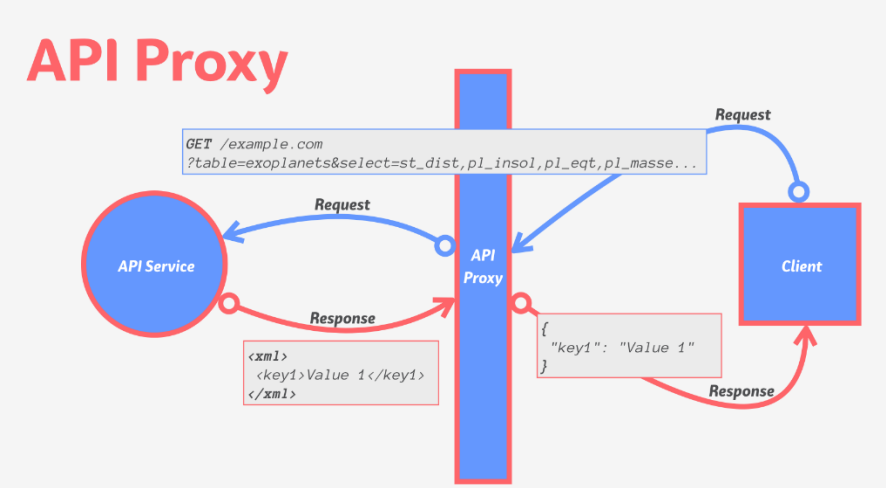
The resource server validates the access token. Validation is a complex process that includes a check that the issuer and audience claims contain expected values.

Based on token validation criteria, access to resources of the backend API is then granted.

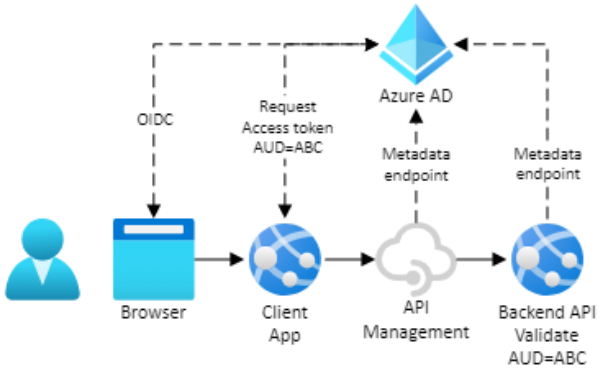
Scenario 1 - Client app authorizes directly to backend

A common authorization scenario is when the calling application requests access to the backend API directly and presents an OAuth 2.0 token in an authorization header to the gateway.

Azure API Management then acts as a "transparent" proxy between the caller and backend API and passes the token through unchanged to the backend. The scope of the access token is between the calling application and backend API.



An *API proxy* is a thin API server that exposes a stable interface for an existing service or services. The following image shows an example where Microsoft Entra ID is the authorization provider. The client app might be a single-page application (SPA).



Although the access token sent along with the HTTP request is intended for the backend API, API Management still allows for a defense in depth approach.

For example, configure policies to validate the JWT, rejecting requests that arrive without a token, or a token that's not valid for the intended backend API. You can also configure API Management to check other claims of interest extracted from the token.

Protect an API in Azure API Management using OAuth 2.0 authorization with Microsoft Entra ID

Register an application (called *backend-app* in this article) in Microsoft Entra ID to protect access to the API.
To access the API, users or applications will acquire and present a valid OAuth token granting access to this app with each API request.
Configure the validate-jwt policy in API Management to validate the OAuth token presented in each incoming API request. Valid requests can be passed to the API.

```
<validate-jwt header-name="Authorization" failed-validation-httpcode="401" failed-validation-error-message="Unauthorized. Access token is missing or invalid.">
  <openid-config url="https://login.microsoftonline.com/{aad-tenant}/v2.0/.well-known/openid-configuration" />
  <audiences>
    <audience>{audience-value - (ex:api://guid)}</audience>
  </audiences>
  <issuers>
    <issuer>{issuer-value - (ex: https://sts.windows.net/{tenant id}/)}</issuer>
  </issuers>
  <required-claims>
    <claim name="aud">
      <value>{backend-app-client-id}</value>
    </claim>
  </required-claims>
</validate-jwt>
```

The validate-jwt policy enforces existence and validity of a supported JSON web token (JWT) extracted from a specified HTTP header, extracted from a specified query parameter, or matching a specific value.

Microsoft Entra ID single tenant token validation

XML Copy

```
<validate-jwt header-name="Authorization" failed-validation-httpcode="401" failed-valid
    <openid-config url="https://login.microsoftonline.com/contoso.onmicrosoft.com/.well
    <audiences>
        <audience>00001111-aaaa-2222-bbbb-3333cccc4444</audience>
    </audiences>
    <required-claims>
        <claim name="id" match="all">
            <value>insert claim here</value>
        </claim>
    </required-claims>
</validate-jwt>
```

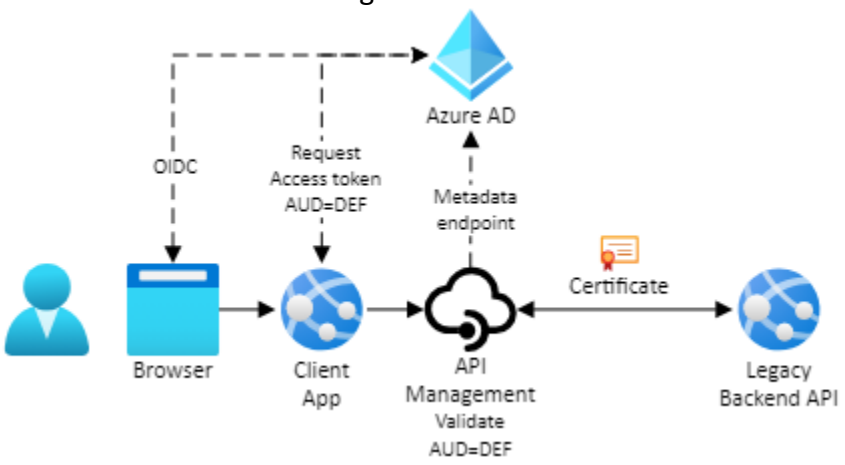
Microsoft Entra ID customer tenant token validation

XML Copy

```
<validate-jwt header-name="Authorization" failed-validation-httpcode="401" failed-valid
    <openid-config url="https://<tenant-name>.ciamlogin.com/<tenant-id>/v2.0/.well-know
    <required-claims>
        <claim name="azp" match="all">
            <value>insert claim here</value>
        </claim>
    </required-claims>
</validate-jwt>
```

Scenario 2 - Client app authorizes to API Management

In this scenario, the API Management service acts on behalf of the API, and the calling application requests access to the API Management instance.



The scope of the access token is between the calling application and the API Management gateway. In API Management, configure a policy (validate-jwt or validate-azure-ad-token) to validate the token before the gateway passes the request to the backend.

Usage of both validate-jwt and validate-azure-ad-token

- Policy sections: inbound
- Policy scopes: global, workspace, product, API, operation
- Gateways: classic, v2, consumption, self-hosted, workspace

A separate mechanism typically secures the connection between the gateway and the backend API.

The validate-azure-ad-token policy enforces the existence and validity of a JSON web token (JWT) that was provided by the Microsoft Entra (formerly called Azure Active Directory) service for a specified set of principals in the directory. The JWT can be extracted from a specified HTTP header, query parameter, or value provided using a policy expression or context variable.

In the following example, Microsoft Entra ID is again the authorization provider, and mutual TLS (mTLS) authentication secures the connection between the gateway and the backend.

There are different reasons for doing this. For example:

- **The backend is a legacy API that can't be updated to support OAuth**

API Management should first be configured to validate the token (checking the issuer and audience claims at a minimum). After validation, use one of several options available to secure onward connections from API Management, such as mutual TLS (mTLS) authentication.

- **The context required by the backend isn't possible to establish from the caller**

After API Management has successfully validated the token received from the caller, it then needs to obtain an access token for the backend API using its own context, or context derived from the calling application. This scenario can be accomplished using either:

  - A custom policy such as send-request to obtain an onward access token valid for the backend API from a configured identity provider.
  - The API Management instance's own identity – passing the token from the API Management resource's system-assigned or user-assigned managed identity to the backend API.

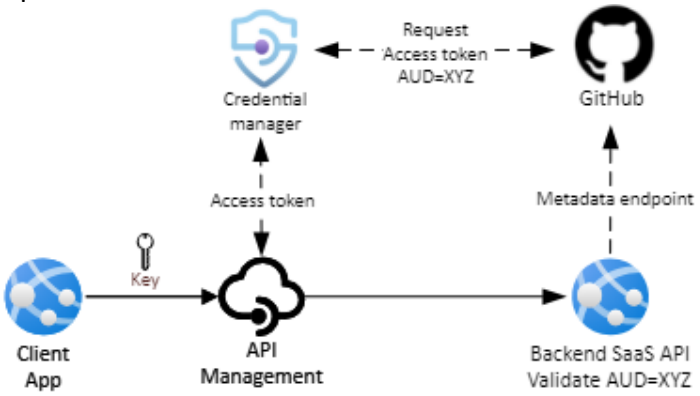
- **The organization wants to adopt a standardized authorization approach**

Regardless of the authentication and authorization mechanisms on their API backends, organizations may choose to converge on OAuth 2.0 for a standardized authorization approach on the front end. API Management's gateway can enable consistent authorization configuration and a common experience for API consumers as the organization's backends evolve.

Scenario 3: API Management authorizes to backend
With managed connections (formerly called *authorizations*), you use credential manager in API Management to authorize access to one or more backend or SaaS services, such as LinkedIn, GitHub, or other OAuth 2.0-compatible backends.

In this scenario, a user or client app makes a request to the API Management gateway, with gateway access controlled using an identity provider or other client side options.

Then, through policy configuration, the user or client app delegates backend authentication and authorization to API Management.
In the following example, a subscription key is used between the client and the gateway, and GitHub is the credential provider for the backend API.



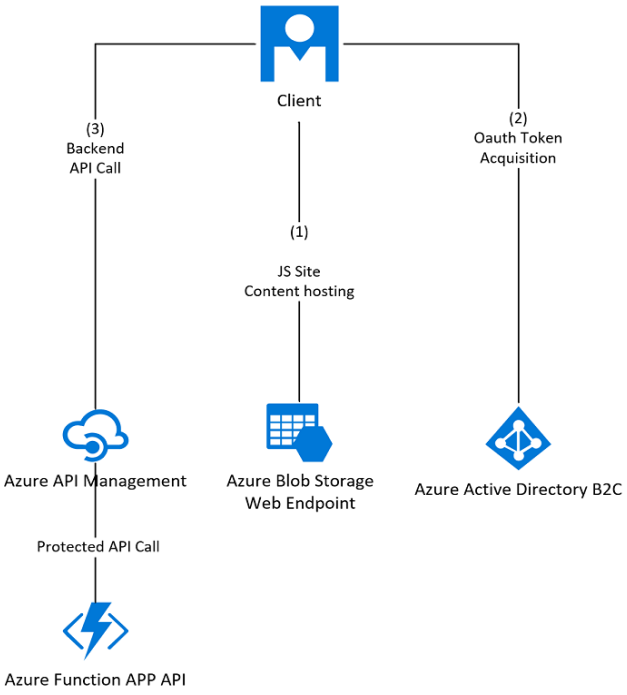
With a connection to a credential provider, API Management acquires and refreshes the tokens for API access in the OAuth 2.0 flow. Connections simplify token management in multiple scenarios, such as:

- A client app might need to authorize to multiple SaaS backends to resolve multiple fields using GraphQL resolvers.
- Users authenticate to API Management by SSO from their identity provider, but authorize to a backend SaaS provider (such as LinkedIn) using a common organizational account.
- A client app (or bot) needs to access backend secured online resources on behalf of an authenticated user (for example, checking emails or placing an order).

Examples:

- Configure credential manager - Microsoft Graph API
- Configure credential manager - GitHub API
- Configure credential manager - user delegated access to backend APIs

Protect serverless APIs with Azure API Management and Azure AD B2C for consumption from a SPA



You'll create a JavaScript (JS) app calling an API that signs in users with Azure AD B2C. Then you'll use API Management's validate-jwt, CORS, and Rate Limit By Key policy features to protect the Backend API.

The rate-limit-by-key policy prevents API usage spikes on a per key basis by limiting the call rate to a specified number per specified time period. When this call rate is exceeded, the caller receives a 429 Too Many Requests response status code.

```
<rate-limit-by-key calls="number"
                   renewal-period="seconds"
                   increment-condition="condition"
                   increment-count="number"
                   counter-key="key value"
                   retry-after-header-name="custom header name, replaces default 'Retry-After'"
                   retry-after-variable-name="policy expression variable name"
                   remaining-calls-header-name="header name"
                   remaining-calls-variable-name="policy expression variable name"
                   total-calls-header-name="header name"/>
```

The cors policy adds cross-origin resource sharing (CORS) support to an operation or an API to allow cross-domain calls from browser-based clients.

```
<cors allow-credentials="false | true" terminate-unmatched-request="true | false">
    <allowed-origins>
        <origin>origin uri</origin>
    </allowed-origins>
    <allowed-methods preflight-result-max-age="number of seconds">
        <method>HTTP verb</method>
    </allowed-methods>
    <allowed-headers>
        <header>header name</header>
    </allowed-headers>
    <expose-headers>
        <header>header name</header>
    </expose-headers>
</cors>
```

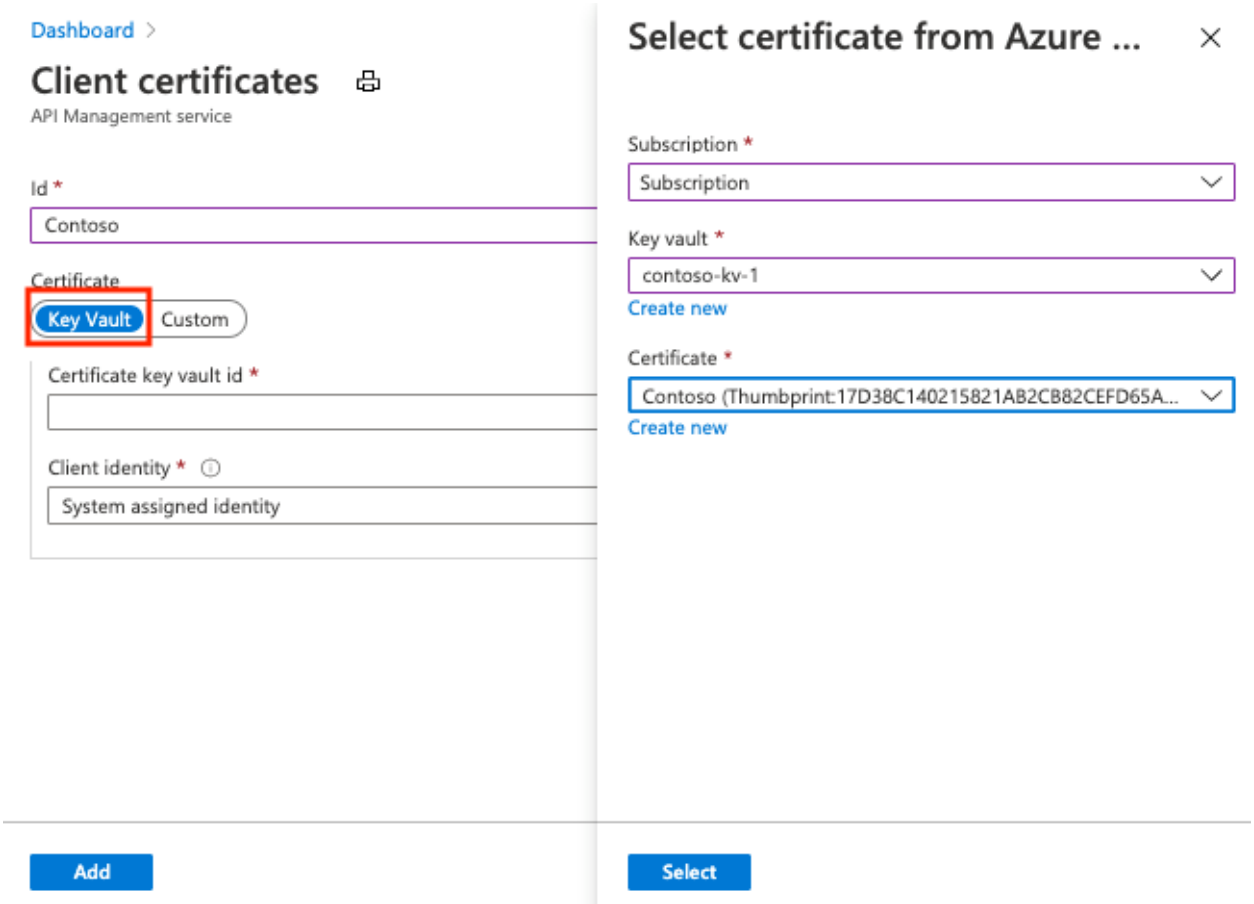
How to secure APIs using client certificate authentication in API Management

API Management provides the capability to secure access to APIs (that is, client to API Management) using client certificates and mutual TLS authentication.
You can validate certificates presented by the connecting client and check certificate properties against desired values using policy expressions.

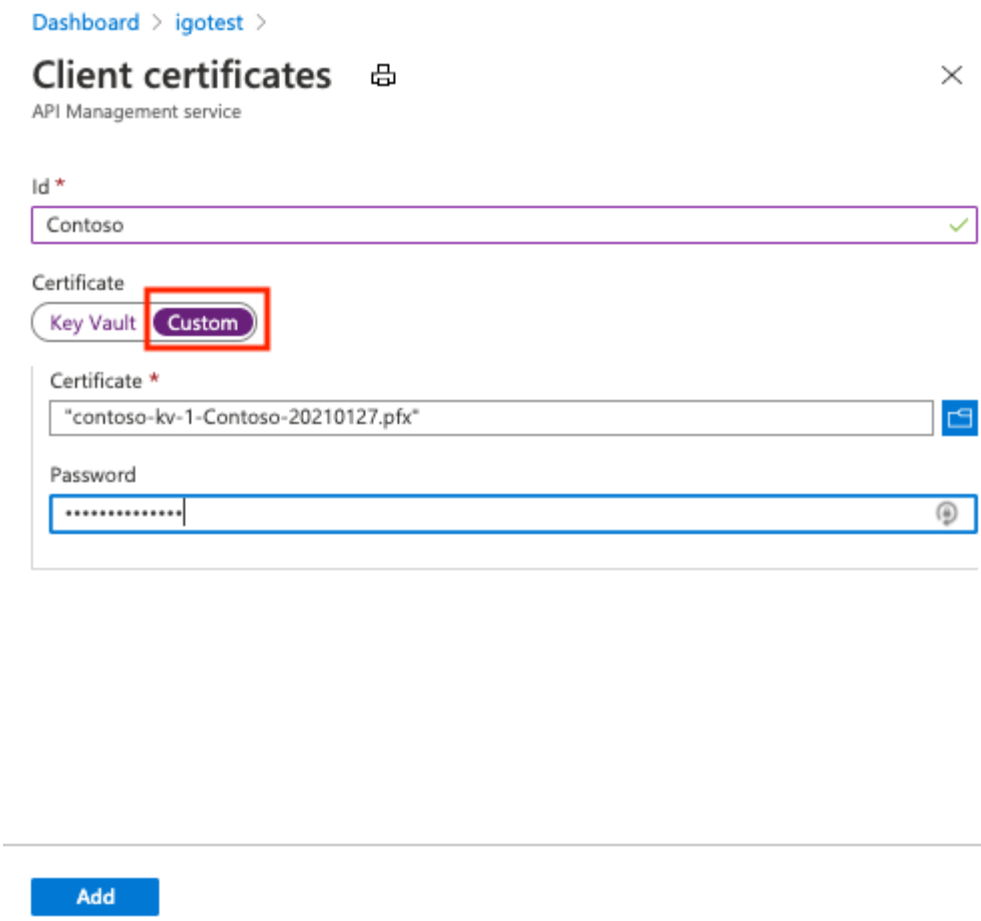
Using key vault certificates is recommended because it helps improve API Management security:

- Certificates stored in key vaults can be reused across services
- Granular access policies can be applied to certificates stored in key vaults
- Certificates updated in the key vault are automatically rotated in API Management. After update in the key vault, a certificate in API Management is updated within 4 hours. You can also manually refresh the certificate using the Azure portal or via the management REST API.

Add a key vault certificate



Upload a certificate


Dashboard > igotest >
Client certificates
API Management service
Id *
Contoso
Certificate
Key Vault
Custom
Certificate *
"contoso-kv-1-Contoso-20210127.pfx"
Password
Add

Policies in Azure API Management

Policies are a collection of statements that are run sequentially on the request or response of an API.
API Management provides more than 50 policies out of the box that you can configure to address common API scenarios such as authentication, rate limiting, caching, and transformation of requests or responses.

Understanding policy configuration

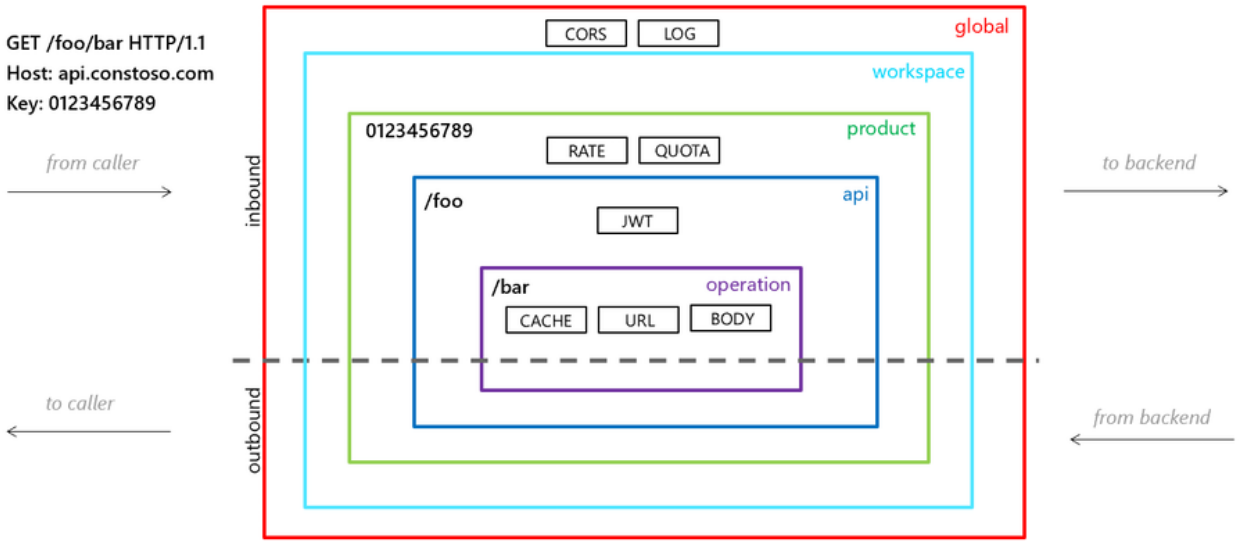
```
<policies>
  <inbound>
    <!-- statements to be applied to the request go here -->
  </inbound>
  <backend>
    <!-- statements to be applied before the request is forwarded to
         the backend service go here -->
  </backend>
  <outbound>
    <!-- statements to be applied to the response go here -->
  </outbound>
  <on-error>
    <!-- statements to be applied if there is an error condition go here -->
  </on-error>
</policies>
```

Scopes
API Management allows you to define policies at the following *scopes*, from most broad to most narrow:

- Global (all APIs)
- Workspace (all APIs associated with a selected workspace)
- Product (all APIs associated with a selected product)
- API (all operations in an API)
- Operation (single operation in an API)

When configuring a policy, you must first select the scope at which the policy applies.

Policy scopes



Policy expressions

Unless the policy specifies otherwise, policy expressions can be used as attribute values or text values in any of the API Management policies. A policy expression is either:

- a single C# statement enclosed in @(expression), or
- a multi-statement C# code block, enclosed in @{expression}, that returns a value

Each expression has access to the implicitly provided context variable and an allowed subset of .NET Framework types.

Policy expressions provide a sophisticated means to control traffic and modify API behavior without requiring you to write specialized code or modify backend services. Some policies are based on policy expressions, such as Control flow and Set variable.

Error handling

By providing a ProxyError object, Azure API Management allows publishers to respond to error conditions, which may occur during processing of requests.

The ProxyError object is accessed through the context.LastError property and can be used by policies in the on-error policy section.

By placing policy statements in the on-error section, you can:

- Review the error using the context.LastError property.
- Inspect and customize the error response using the set-body policy.
- Configure what happens if an error occur

API Management policy reference

The API Management gateways that support each policy are indicated.

Limit call rate by subscription and Set usage quota by subscription have a dependency on the subscription key. A subscription key isn't required when other policies are applied.

Policy	Description	Classic	V2	Consumption	Self-hosted
Limit call rate by subscription	Prevents API usage spikes by limiting call rate, on a per subscription basis.	Yes	Yes	Yes	Yes
Limit call rate by key	Prevents API usage spikes by limiting call rate, on a per key basis.	Yes	Yes	No	Yes
Set usage quota by subscription	Allows you to enforce a renewable or lifetime call volume and/or bandwidth quota, on a per subscription basis.	Yes	Yes	Yes	Yes
Set usage quota by key	Allows you to enforce a renewable or lifetime call volume and/or bandwidth quota, on a per key basis.	Yes	No	No	Yes
Limit concurrency	Prevents enclosed policies from executing by more than the specified number of requests at a time.	Yes	Yes	Yes	Yes
Limit Azure OpenAI Service token usage	Prevents Azure OpenAI API usage spikes by limiting large language model tokens per calculated key.	Yes	Yes	No	No
Limit large language model API token usage	Prevents large language model (LLM) API usage spikes by limiting LLM tokens per calculated key.	Yes	Yes	No	No

Authentication and authorization

Policy	Description	Classic	V2	Consumption	Self-hosted
Check HTTP header	Enforces existence and/or value of an HTTP header.	Yes	Yes	Yes	Yes
Get authorization context	Gets the authorization context of a specified connection to a credential provider configured in the API Management instance.	Yes	Yes	Yes	No
Restrict caller IPs	Filters (allows/denies) calls from specific IP addresses and/or address ranges.	Yes	Yes	Yes	Yes
Validate Microsoft Entra token	Enforces existence and validity of a Microsoft Entra (formerly called Azure Active Directory) JWT extracted from either a specified HTTP header, query parameter, or token value.	Yes	Yes	Yes	Yes
Validate JWT	Enforces existence and validity of a JWT extracted from either a specified HTTP header, query parameter, or token value.	Yes	Yes	Yes	Yes
Validate client certificate	Enforces that a certificate presented by a client to an API Management instance matches specified validation rules and claims.	Yes	Yes	Yes	Yes
Authenticate with Basic	Authenticates with a backend service using Basic authentication.	Yes	Yes	Yes	Yes
Authenticate with client certificate	Authenticates with a backend service using client certificates.	Yes	Yes	Yes	Yes
Authenticate with managed identity	Authenticates with a backend service using a managed identity.	Yes	Yes	Yes	Yes

Content validation

Policy	Description	Classic	V2	Consumption	Self-hosted
Validate content	Validates the size or content of a request or response body against one or more API schemas. The supported schema formats are JSON and XML.	Yes	Yes	Yes	Yes
Validate GraphQL request	Validates and authorizes a request to a GraphQL API.	Yes	Yes	Yes	Yes
Validate OData request	Validates a request to an OData API to ensure conformance with the OData specification.	Yes	Yes	Yes	Yes
Validate parameters	Validates the request header, query, or path parameters against the API schema.	Yes	Yes	Yes	Yes
Validate headers	Validates the response headers against the API schema.	Yes	Yes	Yes	Yes
Validate status code	Validates the HTTP status codes in responses against the API schema.	Yes	Yes	Yes	Yes

Routing

Policy	Description	Classic	V2	Consumption	Self-hosted
Forward request	Forwards the request to the backend service.	Yes	Yes	Yes	Yes
Set backend service	Changes the backend service base URL of an incoming request to a URL or a backend. Referencing a backend resource allows you to manage the backend service base URL and other settings in a single place. Also implement load balancing of traffic across a pool of backend services and circuit breaker rules to protect the backend from too many requests.	Yes	Yes	Yes	Yes
Set HTTP proxy	Allows you to route forwarded requests via an HTTP proxy.	Yes	Yes	Yes	Yes

Caching

Policy	Description	Classic	V2	Consumption	Self-hosted
Get from cache	Performs cache lookup and return a valid cached response when available.	Yes	Yes	Yes	Yes
Store to cache	Caches response according to the specified cache control configuration.	Yes	Yes	Yes	Yes
Get value from cache	Retrieves a cached item by key.	Yes	Yes	Yes	Yes
Store value in cache	Stores an item in the cache by key.	Yes	Yes	Yes	Yes
Remove value from cache	Removes an item in the cache by key.	Yes	Yes	Yes	Yes
Get cached responses of Azure OpenAI API requests	Performs lookup in Azure OpenAI API cache using semantic search and returns a valid cached response when available.	Yes	Yes	Yes	Yes
Store responses of Azure OpenAI API requests to cache	Caches response according to the Azure OpenAI API cache configuration.	Yes	Yes	Yes	Yes
Get cached responses of large language model API requests	Performs lookup in large language model API cache using semantic search and returns a valid cached response when available.	Yes	Yes	Yes	Yes
Store responses of large language model API requests to cache	Caches response according to the large language model API cache configuration.	Yes	Yes	Yes	Yes

Transformation

Policy	Description	Classic	V2	Consumption	Self-hosted
Set request method	Allows you to change the HTTP method for a request.	Yes	Yes	Yes	Yes
Set status code	Changes the HTTP status code to the specified value.	Yes	Yes	Yes	Yes
Set variable	Persists a value in a named context variable for later access.	Yes	Yes	Yes	Yes
Set body	Sets the message body for a request or response.	Yes	Yes	Yes	Yes
Set HTTP header	Assigns a value to an existing response and/or request header or adds a new response and/or request header.	Yes	Yes	Yes	Yes
Set query string parameter	Adds, replaces value of, or deletes request query string parameter.	Yes	Yes	Yes	Yes
Rewrite URL	Converts a request URL from its public form to the form expected by the web service. Policy sections**: inbound**	Yes	Yes	Yes	Yes
Convert JSON to XML	Converts request or response body from JSON to XML.	Yes	Yes	Yes	Yes
Convert XML to JSON	Converts request or response body from XML to JSON.	Yes	Yes	Yes	Yes
Find and replace string in body	Finds a request or response substring and replaces it with a different substring.	Yes	Yes	Yes	Yes
Mask URLs in content	Rewrites (masks) links in the response body so that they point to the equivalent link via the gateway.	Yes	Yes	Yes	Yes
Transform XML using an XSLT	Applies an XSL transformation to XML in the request or response body.	Yes	Yes	Yes	Yes
Return response	Aborts pipeline execution and returns the specified response directly to the caller.	Yes	Yes	Yes	Yes
Mock response	Aborts pipeline execution and returns a mocked response directly to the caller.	Yes	Yes	Yes	Yes

Cross-domain

Policy	Description	Classic	V2	Consumption	Self-hosted
Allow cross-domain calls	Makes the API accessible from Adobe Flash and Microsoft Silverlight browser-based clients.	Yes	Yes	Yes	Yes
CORS	Adds cross-origin resource sharing (CORS) support to an operation or an API to allow cross-domain calls from browser-based clients.	Yes	Yes	Yes	Yes
JSONP	Adds JSON with padding (JSONP) support to an operation or an API to allow cross-domain calls from JavaScript browser-based clients.	Yes	Yes	Yes	Yes

Integration and external communication

Policy	Description	Classic	V2	Consumption	Self-hosted
Send request	Sends a request to the specified URL. Policy sections: **inbound, outbound, backend, on-error**	Yes	Yes	Yes	Yes
Send one way request	Sends a request to the specified URL without waiting for a response. Policy sections: **inbound, outbound, backend, on-error**	Yes	Yes	Yes	Yes
Log to event hub	Sends messages in the specified format to an event hub defined by a Logger entity.	Yes	Yes	Yes	Yes
Send request to a service (Dapr)	Uses Dapr runtime to locate and reliably communicate with a Dapr microservice. To learn more about service invocation in Dapr, see the description in this README file. Policy sections: **inbound**	No	No	No	Yes
Send message to Pub/Sub topic (Dapr)	Uses Dapr runtime to publish a message to a Publish/Subscribe topic. To learn more about Publish/Subscribe messaging	No	No	No	Yes

	in Dapr, see the description in this README file.				
Trigger output binding (Dapr)	Uses Dapr runtime to invoke an external system via output binding. To learn more about bindings in Dapr, see the description in this README file.	No	No	No	Yes

Logging

Policy	Description	Classic	V2	Consumption	Self-hosted
Trace	Adds custom traces into the request tracing output in the test console, Application Insights telemetries, and resource logs.	Yes	Yes[1]	Yes	Yes
Emit metrics	Sends custom metrics to Application Insights at execution.	Yes	Yes	Yes	Yes
Emit Azure OpenAI token metrics	Sends metrics to Application Insights for consumption of large language model tokens through Azure OpenAI service APIs.	Yes	Yes	No	Yes
Emit large language model API token metrics	Sends metrics to Application Insights for consumption of large language model (LLM) tokens through LLM APIs.	Yes	Yes	No	Yes

[1] In the V2 gateway, the trace policy currently does not add tracing output in the test console.

GraphQL resolvers

Policy	Description	Classic	V2	Consumption	Self-hosted
Azure SQL data source for resolver	Configures the Azure SQL request and optional response to resolve data for an object type and field in a GraphQL schema.	Yes	Yes	No	No
Cosmos DB data source for resolver	Configures the Cosmos DB request and optional response to resolve data for an object type and field in a GraphQL schema.	Yes	Yes	No	No
HTTP data source for resolver	Configures the HTTP request and optionally the HTTP response to resolve data for an object type and field in a GraphQL schema.	Yes	Yes	Yes	No

Publish event to GraphQL subscription	Publishes an event to one or more subscriptions specified in a GraphQL API schema. Configure the policy in a GraphQL resolver for a related field in the schema for another operation type such as a mutation.	Yes	Yes	Yes	No

Policy control and flow

Policy	Description	Classic	V2	Consumption	Self-hosted
Control flow	Conditionally applies policy statements based on the results of the evaluation of Boolean expressions.	Yes	Yes	Yes	Yes
Include fragment	Inserts a policy fragment in the policy definition.	Yes	Yes	Yes	Yes
Retry	Retries execution of the enclosed policy statements, if and until the condition is met. Execution will repeat at the specified time intervals and up to the specified retry count.	Yes	Yes	Yes	Yes
Wait	Waits for enclosed Send request, Get value from cache, or Control flow policies to complete before proceeding.	Yes	Yes	Yes	Yes

Policy fragments are centrally managed, reusable XML snippets containing one or more API Management policy configurations. Policy fragments help you configure policies consistently and maintain policy definitions without needing to repeat or retype XML code.

A policy fragment:

- Must be valid XML containing one or more policy configurations
- May include policy expressions, if a referenced policy supports them
- Is inserted as-is in a policy definition by using the include-fragment policy

Limitations:

- A policy fragment can't include a policy section identifier (<inbound>, <outbound>, etc.) or the <base/> element.
- Currently, a policy fragment can't nest another policy fragment.
- The maximum size of a policy fragment is 32 KB.

Create a new policy fragment

Add your policy fragment properties.

Properties

Name *

ForwardContext

Description

Sets a custom header to forward context information to backend service

XML policy fragment

```
<!--
    IMPORTANT:
    - Policy fragment are included as-is whenever they are referenced.
    - If using variables. Ensure they are setup before use.
    - Copy and paste your code here or simply start coding
-->
<fragment>
    <set-header name="x-request-context-data" exists-action="override">
        <value>@(context.User.Id)</value>
        <value>@(context.Deployment.Region)</value>
    </set-header>
</fragment>
```

Create Discard

```
<policies>
  <inbound>
    <include-fragment fragment-id="ForwardContext" />
    <base />
  </inbound>
[...]
```

Secure backend services using client certificate authentication in Azure API Management

API Management allows you to secure access to the backend service of an API using client certificates and mutual TLS authentication.

Add a key vault certificate
See Prerequisites for key vault integration.
Important
When adding a key vault certificate to your API Management instance, you must have permissions to list secrets from the key vault.

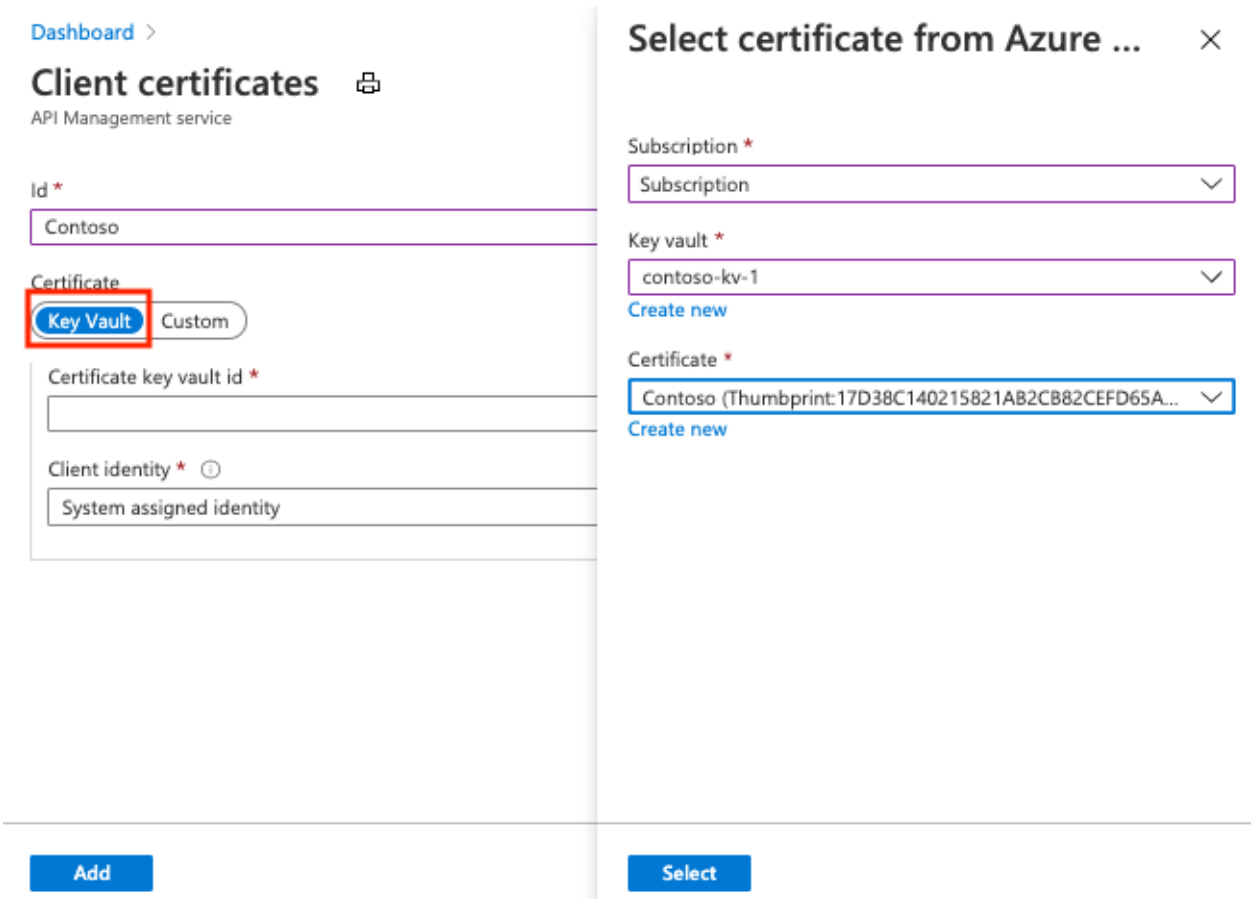
Caution
When using a key vault certificate in API Management, be careful not to delete the certificate, key vault, or managed identity used to access the key vault.

To add a key vault certificate to API Management:

1. In the Azure portal, navigate to your API Management instance.
2. Under **Security**, select **Certificates**.
3. Select **Certificates** > **+ Add**.
4. In **Id**, enter a name of your choice.
5. In **Certificate**, select **Key vault**.
6. Enter the identifier of a key vault certificate, or choose **Select** to select a certificate from a key vault.
7. In **Client identity**, select a system-assigned or an existing user-assigned managed identity. Learn how to add or modify managed identities in your API Management service.

Note
The identity needs permissions to get and list certificate from the key vault. If you haven't already configured access to the key vault, API Management prompts you so it can automatically configure the identity with the necessary permissions.

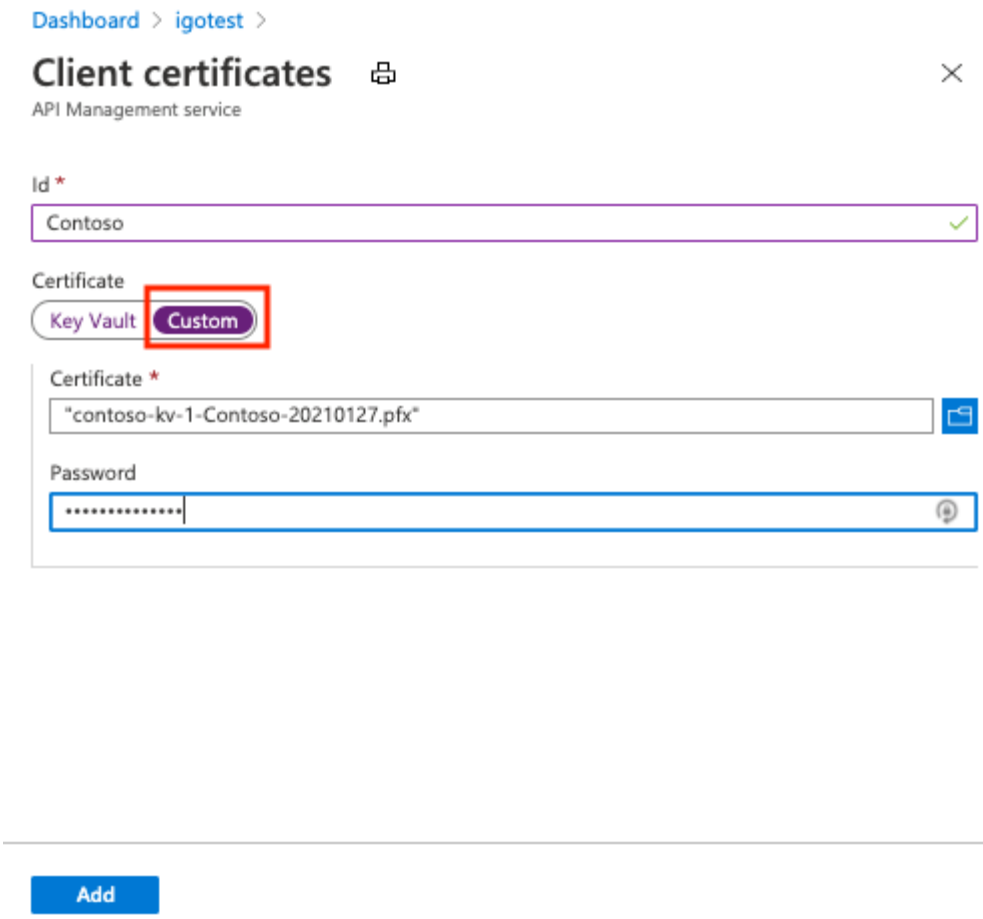
8. Select **Add**.



Upload a certificate

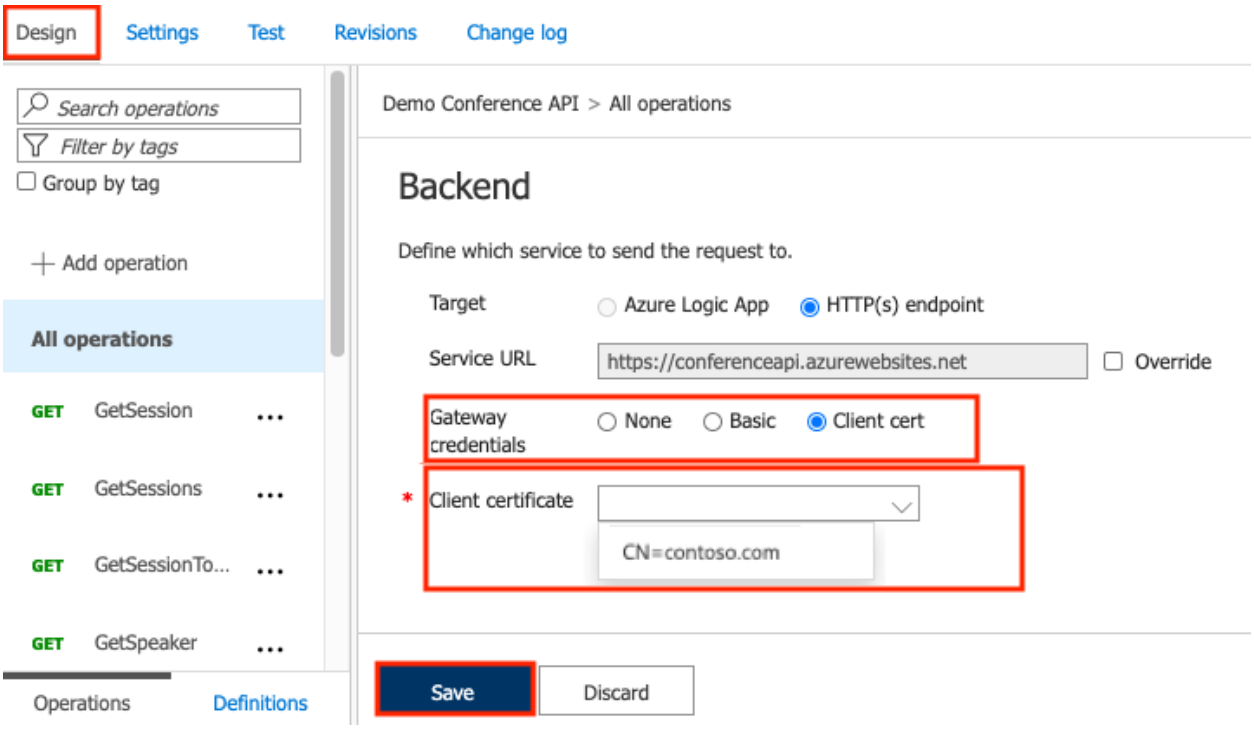
To upload a client certificate to API Management:

1. In the Azure portal, navigate to your API Management instance.
2. Under **Security**, select **Certificates**.
3. Select **Certificates** > **+ Add**.
4. In **Id**, enter a name of your choice.
5. In **Certificate**, select **Custom**.
6. Browse to select the certificate .pfx file, and enter its password.
7. Select **Add**.



Configure an API to use client certificate for gateway authentication

1. In the Azure portal, navigate to your API Management instance.
2. Under **APIs**, select **APIs**.
3. Select an API from the list.
4. In the **Design** tab, select the editor icon in the **Backend** section.
5. In **Gateway credentials**, select **Client cert** and select your certificate from the dropdown.
6. Select **Save**.



Disable certificate chain validation for self-signed certificates

If you are using self-signed certificates, you will need to disable certificate chain validation for API Management to communicate with the backend system.

Otherwise it will return a 500 error code. To configure this, you can use the New-AzApiManagementBackend (for new backend) or Set-AzApiManagementBackend (for existing backend) PowerShell cmdlets and set the -SkipCertificateChainValidation parameter to True.