



API Management

What is Azure API Management?



Azure API Management

Agenda

- | | | | |
|-----------|----------------------------------|-----------|----------------------------------|
| 01 | Introduction to API | 02 | API Management Overview |
| 03 | API Components | 04 | Backends in API Management |
| 05 | Products and Groups | 06 | Versions in Azure API Management |
| 07 | Revision in Azure API Management | 08 | Policies |
| 09 | Securing API | 10 | Monitoring API |
| 11 | Azure Automation | | |

What is an API ?

Application Programming Interface

API is the acronym for **Application Programming Interface**, which is a software intermediary that allows two applications to talk to each other. Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you're using an API.

An application programming interface is a connection between computers or between computer programs. It is a type of software interface, offering a service to other pieces of software.

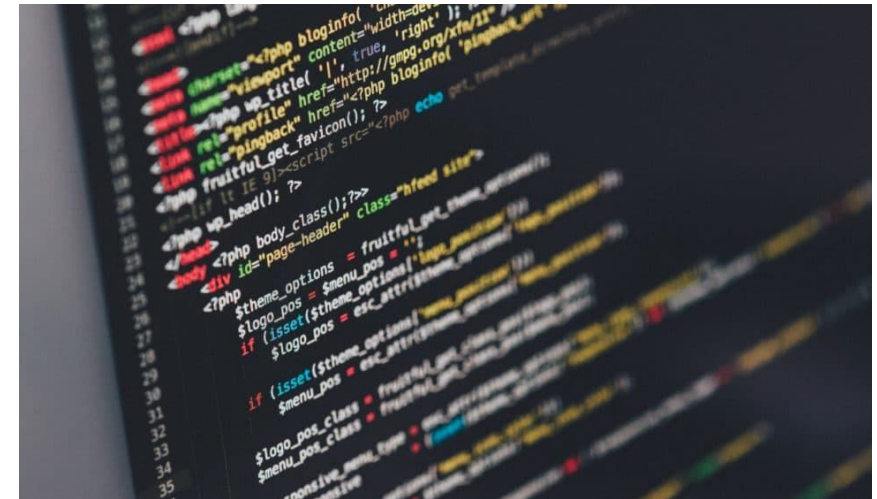


API Management Overview

API Specification

A document or standard that describes how to build or use such a connection or interface is called an **API specification**. A computer system that meets this standard is said to *implement* or *expose* an API.

An API specification provides a broad understanding of how an API behaves and how the API links with other APIs. It explains how the API functions and the results to expect when using the API.



API Management Overview

API Management Overview

Azure API Management is a fully managed service that enables customers to **publish, secure, transform** and **monitor APIs**.

API Management provides the core competencies to ensure a successful API program through developer engagement, business insights, analytics, security, and protection.

Azure API Management



API Management helps organizations publish APIs to external, partner, and internal developers to unlock the potential of their data and services.

API Management Overview

API Components

1. The **API gateway** is the endpoint that:

- Accepts API calls and routes them to your backends.
- 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 where set up.
- Logs call metadata for analytics purposes.

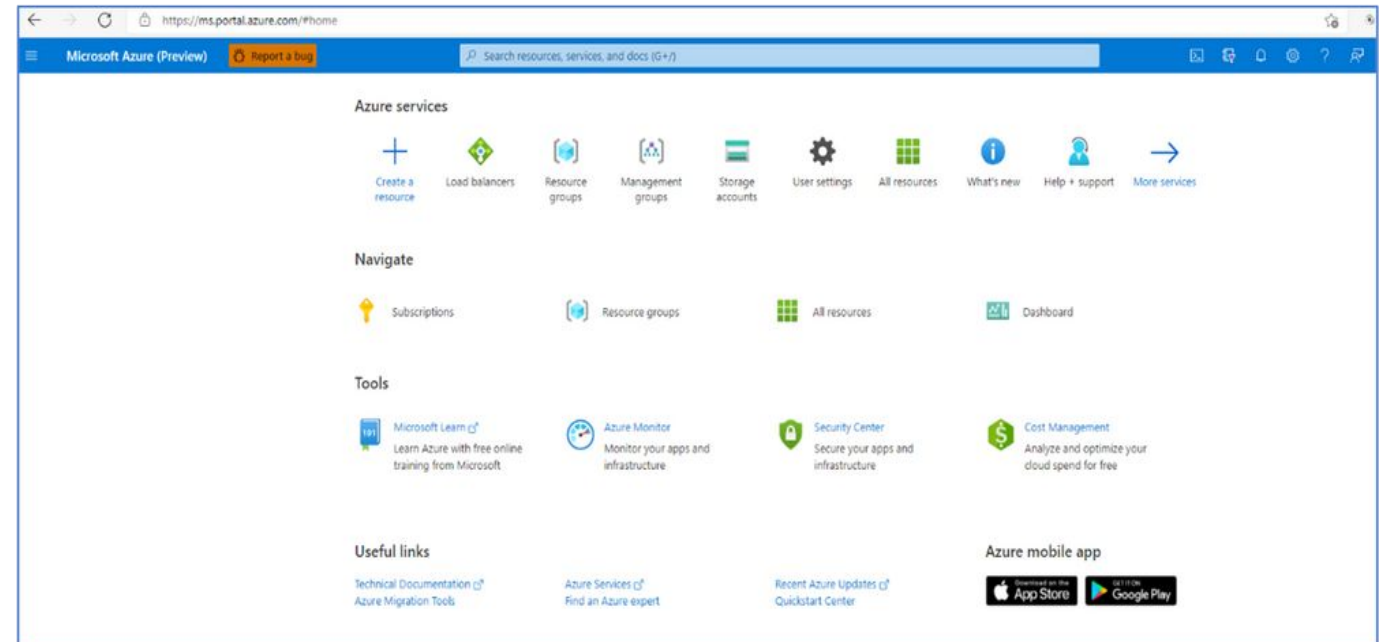


API Management Overview

API Components

2. The **Azure portal** is the administrative interface to set up your API program. It is used to:

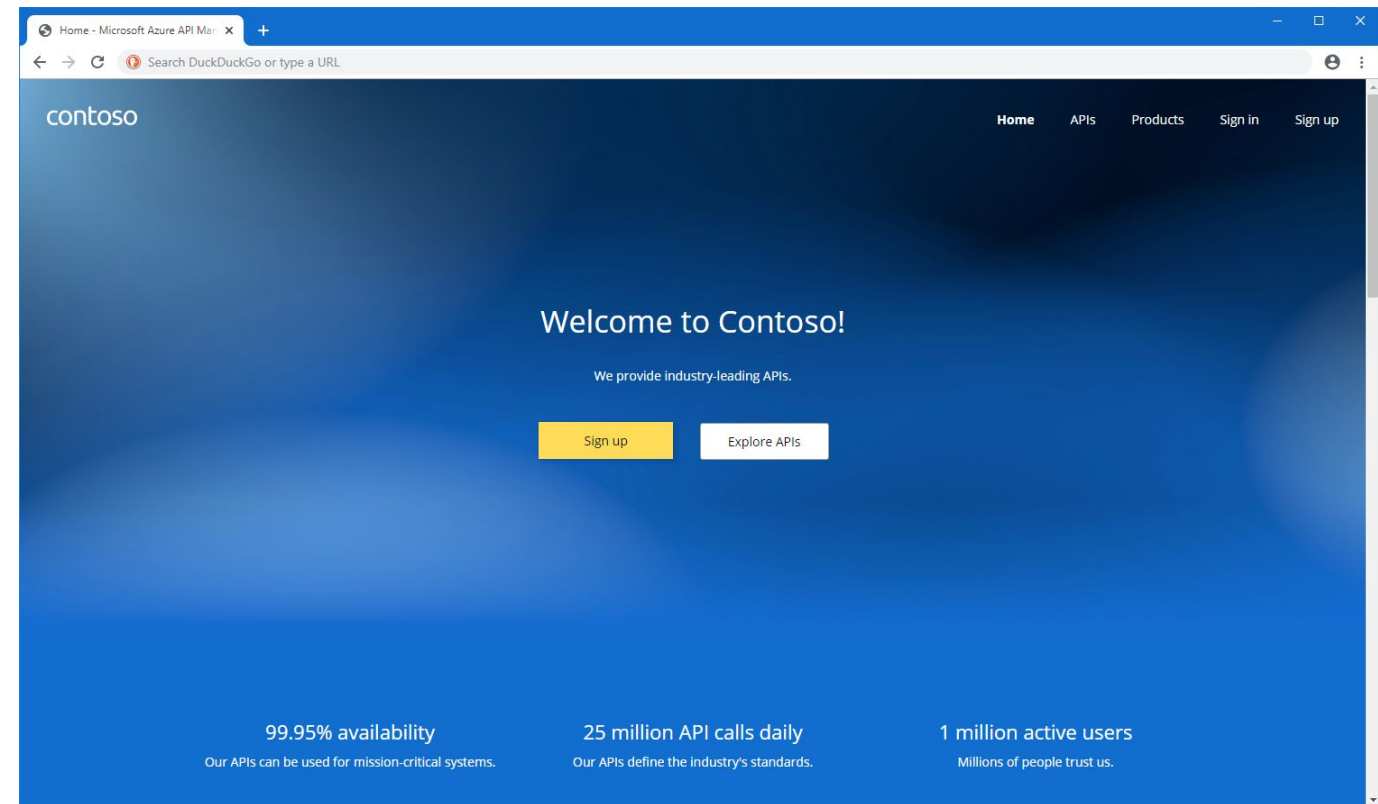
- Define or import API schema.
- Package APIs into products.
- Set up policies like quotas or transformations on the APIs.
- Get insights from analytics.
- Manage users.



API Components

3. The **Developer portal** serves as the main web presence for developers, where they can:

- 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.



Hands-on :
Creating API Management Instance using
portal and importing first API

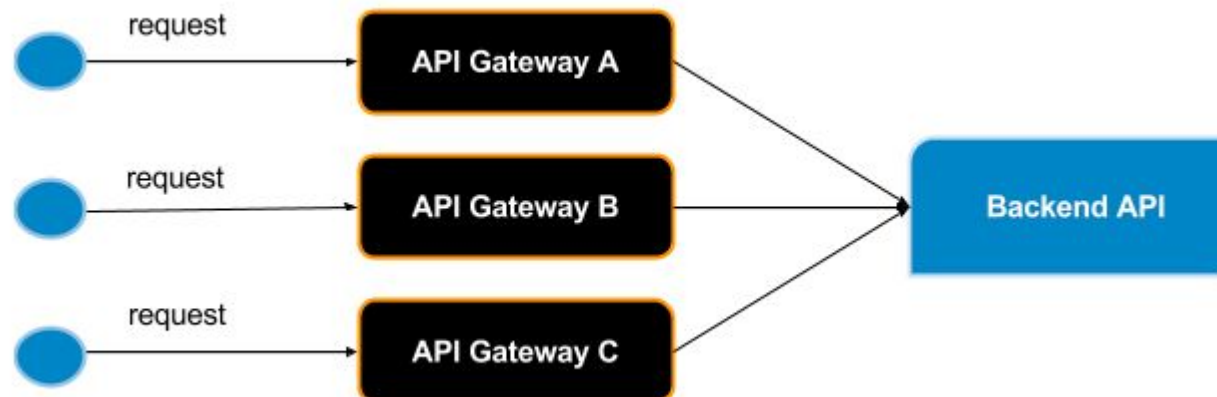


Backends in API Management

Backend API

A *backend* (or *API backend*) in API Management is an HTTP service that implements your front-end API and its operations.

When importing certain APIs, API Management configures the API backend automatically.



API Management Overview

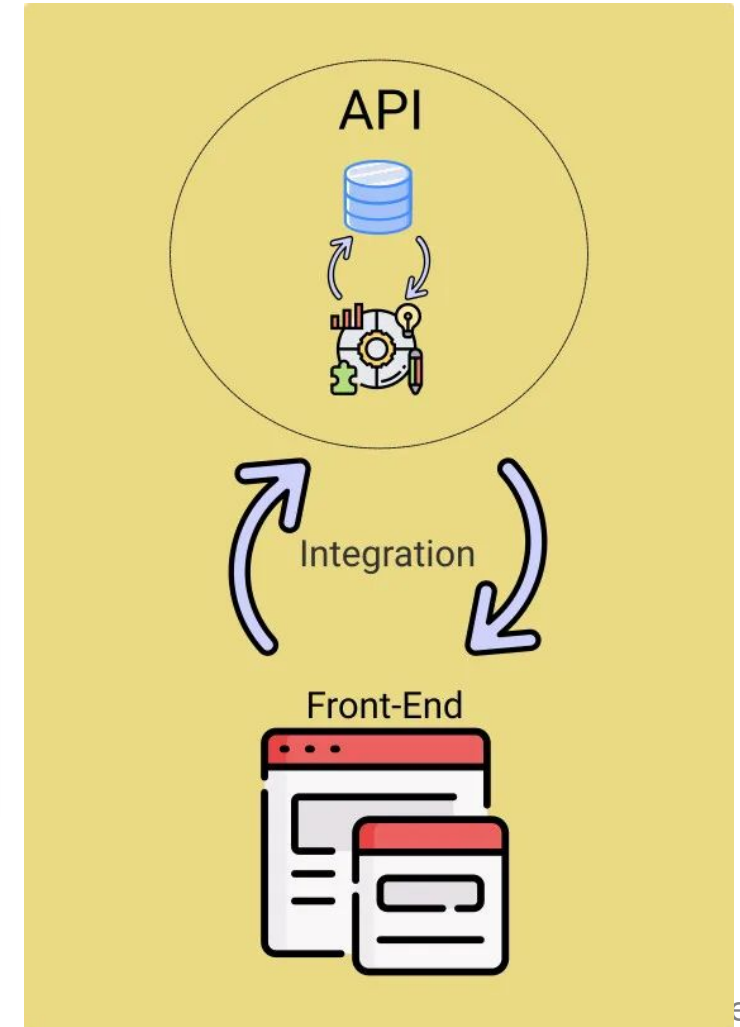
Backend API

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

- A [Service Fabric cluster](#).
- A custom service.

Custom backends require extra configuration to authorize the credentials of requests to the backend service and define API operations. Configure and manage custom backends in the Azure portal, or using Azure APIs or tools.

After creating a backend, you can reference the backend URL in your APIs. Use the [set-backend-service](#) policy to redirect an incoming API request to the custom backend instead of the default backend for that API.





OpenAPI Specification

The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection. When properly defined, a consumer can understand and interact with the remote service with a minimal amount of implementation logic.

SOAP API

SOAP – Simple Object Access Protocol – is probably the better known of the two models.

SOAP relies heavily on XML, and together with schemas, defines a very strongly typed messaging framework.



Benefits of backends

A custom backend has several benefits, including:

- Abstracts information about the backend service, promoting reusability across APIs and improved governance.
- Easily used by configuring a transformation policy on an existing API.
- Takes advantage of API Management functionality to maintain secrets in Azure Key Vault if named values are configured for header or query parameter authentication.

Limitation of Backend

For Developer and Premium tiers, an API Management instance deployed in an [internal virtual network](#) can throw HTTP 500 “BackendConnectionFailure” errors when the gateway endpoint URL and backend URL are the same. If you encounter this limitation, follow the instructions in the [Self-Chained API Management request limitation in internal virtual network mode](#) article in the Tech Community blog.



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.

The visibility of products to developers is controlled through groups.

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.



GROUPS

- **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](#). Custom and external groups can be used alongside system groups in giving developers visibility and access to API products.



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.
- When developers subscribe to a product, they are granted the primary and secondary key for the product. This key is used when making calls into the product's APIs.

Hands-on : Creating and publishing a product



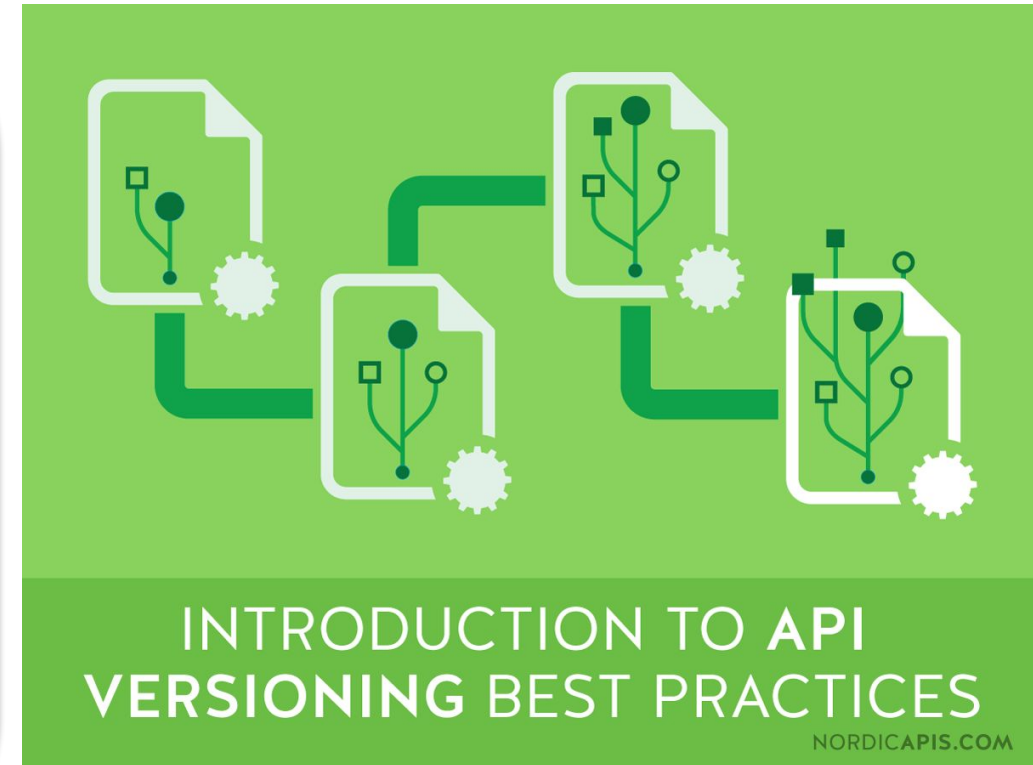
Versions in Azure API Management

API Management Overview

Version in Azure API Management

Versions allow you to present groups of related APIs to your developers. You can use versions to handle breaking changes in your API safely. Clients can choose to use your new API version when they're ready, while existing clients continue to use an older version. Versions are differentiated through a version identifier (which is any string value you choose), and a versioning scheme allows clients to identify which version of an API they want to use.

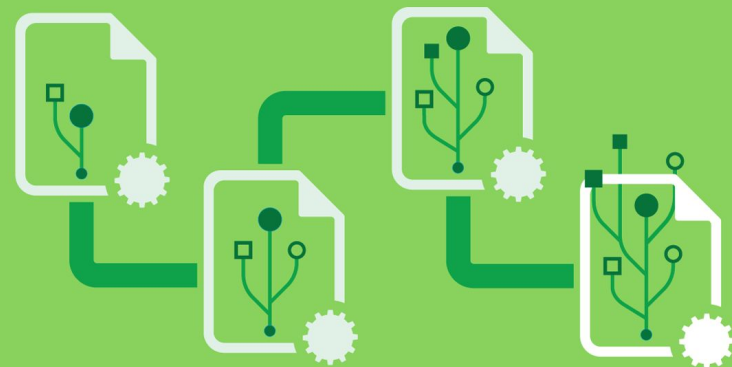
For most purposes, each API version can be considered its own independent API. Two different API versions might have different sets of operations and different policies.



Version in Azure API Management

With versions you can:

- 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.



INTRODUCTION TO API
VERSIONING BEST PRACTICES

NORDICAPIS.COM

Types of Versioning

Path-based versioning

When the path versioning scheme is used, the version identifier needs to be included in the URL path for any API requests.

For example, `https://apis.contoso.com/products/v1` and `https://apis.contoso.com/products/v2` could refer to the same `products` API but to versions `v1` and `v2` respectively.

The format of an API request URL when using path-based versioning is:

```
https://{yourDomain}/{apiName}/{versionIdentifier}/{operationId}.
```



Header-based versioning



When the header versioning scheme is used, the version identifier needs to be included in an HTTP request header for any API requests. You can specify the name of the HTTP request header.

For example, you might create a custom header named Api-Version, and clients could specify v1 or v2 in the value of this header.

Query string-based versioning

When the query string versioning scheme is used, the version identifier needs to be included in a query string parameter for any API requests. You can specify the name of the query string parameter.

The format of an API request URL when using query string-based versioning is:

`https://{yourDomain}/{apiName}/{operationId}?{queryStringParameterName}={versionIdentifier}`.

For example, `https://apis.contoso.com/products?api-version=v1` and `https://apis.contoso.com/products/api-version=v2` could refer to the same `products` API but to versions `v1` and `v2` respectively.



Migrating a non-versioned API to a versioned API

When you use the Azure portal to enable versioning on an existing API, the following changes are made to your API Management resources:

- A new version set is created.
- The existing version is maintained and **configured as the Original API version**. The API is linked to the version set but doesn't require a version identifier to be specified.
- The new version is created as a new API, and is linked to the version set. This new API must be accessed using the versioning scheme and identifier.





Revision in Azure API Management overview

Revisions in Azure API Management

Revisions allow you to make changes to your APIs in a controlled and safe way. When you want to make changes, create a new revision. You can then edit and test API without disturbing your API consumers. When you're ready, you then make your revision current. At the same time, you can optionally post an entry to the change log, to keep your API consumers up to date with what has changed. The change log is published to your developer portal.

Azure/azure-api-management-devops-resource-kit

#316 [Creator] Revisions with subsequent api arm template gives error

11 comments



kbulte opened on January 6, 2020



Revisions in Azure API Management

Azure/azure-api-management-devops-resource-kit

#316 [Creator] Revisions
with subsequent api
arm template gives error

11 comments

 kbulte opened on January 6, 2020



With revisions you can:

- Safely make changes to your API definitions and policies, without disturbing your production API.
- Try out changes before publishing them.
- Document the changes you make, so your developers can understand what is new.
- Roll back if you find issues.

Accessing specific revisions

Each revision to your API can be accessed using a specially formed URL. Append ;rev={revisionNumber} at the end of your API URL, but before the query string, to access a specific revision of that API. For example, you might use this URL to access revision 3 of the customers API:

<https://apis.contoso.com/customers;rev=3?customerId=123>

By default, each revision has the same security settings as the current revision. You can deliberately change the policies for a specific revision if you want to have different security applied for each revision. For example, you might want to add a [IP filtering policy](#) to prevent external callers from accessing a revision that is still under development.

A revision can be taken offline, which makes it inaccessible to callers even if they try to access the revision through its URL. You can mark a revision as offline using the Azure portal. If you use PowerShell, you can use the `Set-AzApiManagementApiRevision` cmdlet and set the `Path` argument to `$null`.

Current Revision

A single revision can be set as the *current* revision. This revision will be the one used for all API requests that don't specify an explicit revision number in the URL. You can roll back to a previous revision by setting that revision as current.

You can set a revision as current using the Azure portal. If you use PowerShell, you can use the `New-AzApiManagementApiRelease` cmdlet.

Azure/azure-api-management-devops-resource-kit

#316 [Creator] Revisions with subsequent api arm template gives error



11 comments



kbulte opened on January 6, 2020



Revision descriptions

When you create a revision, you can set a description for your own tracking purposes. Descriptions aren't played to your API users.

When you set a revision as current you can also optionally specify a public change log note. The change log is included in the developer portal for your API users to view. You can modify your change log note using the Update-AzApiManagementApiRelease PowerShell cmdlet.

Azure/azure-api-management-devops-resource-kit

#316 [Creator] Revisions with subsequent api arm template gives error



11 comments



kbulte opened on January 6, 2020





Policies in Azure API Management overview

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.



- Policies are applied inside the gateway which sits between the API consumer and the managed API. The gateway receives all requests and usually forwards them unaltered to the underlying API.

However a policy can apply changes to both the inbound request and outbound response.



Policy Configuration

- The **policy definition is a XML document** that describes a sequence of inbound and outbound statements.
- The configuration is divided into **inbound, backend, outbound, and on-error**. The series of specified policy statements is executed in order for a request and a response.
- If there is an **error during the processing** of a request, any remaining steps in the inbound, backend, or outbound sections are skipped and execution jumps to the statements in the on-error section.
- By placing policy statements in the on-error section you can **review the error by using the context.LastError** property, inspect and customize the error response using the set-body policy, and configure what happens if an error occurs.

```
<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>
```

Apply policies specified at different scopes

If a policy exists at the global level and a policy configured for an API, then whenever that particular API is used **both policies will be applied. API Management allows for deterministic ordering of combined policy statements via the base element.**

```
<policies>
  <inbound>
    <cross-domain />
    <base />
    <find-and-replace from="xyz" to="abc" />
  </inbound>
</policies>
```

In the example policy definition above, the cross-domain statement would execute before any higher policies which would in turn, be followed by the find-and-replace policy.

Filter response content

- The policy defined in example below demonstrates how to filter data elements from the response payload based on the product associated with the request.
- The snippet assumes that response content is formatted as JSON and contains root-level properties named “minutely”, “hourly”, “daily”, “flags”.

```
<policies>
  <inbound>
    <base />
  </inbound>
  <backend>
    <base />
  </backend>
  <outbound>
    <base />
    <choose>
      <when condition="@context.Response.StatusCode == 200 && context.P
        <!-- NOTE that we are not using preserveContent=true when deseri
        <set-body>
          @{
            var response = context.Response.Body.As<JsonObject>();
            foreach (var key in new [] {"minutely", "hourly", "daily", "
              response.Property (key).Remove ();
            }
            return response.ToString();
          }
        </set-body>
      </when>
    </choose>
  </outbound>
  <on-error>
    <base />
  </on-error>
</policies>
```

API Management advanced policies

- **Control flow** - Conditionally applies policy statements based on the results of the evaluation of Boolean expressions.
- **Forward request** - Forwards the request to the backend service.
- **Limit concurrency** - Prevents enclosed policies from executing by more than the specified number of requests at a time.
- **Log to Event Hub** - Sends messages in the specified format to an Event Hub defined by a Logger entity.
- **Mock response** - Aborts pipeline execution and returns a mocked response directly to the caller.
- **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.



Choose Policy

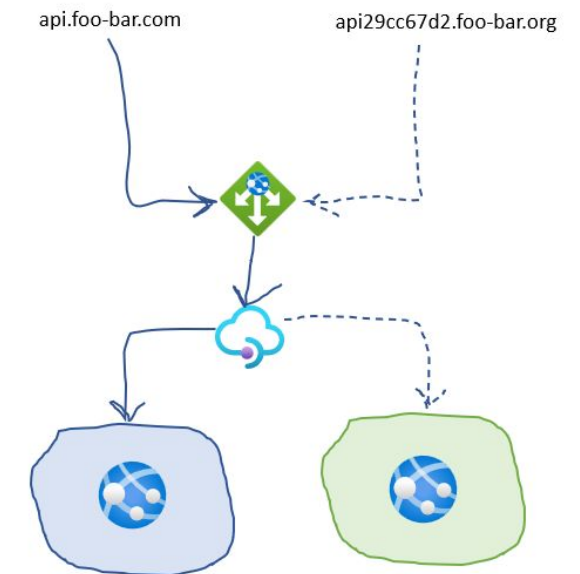
The choose policy applies enclosed policy statements based on the outcome of evaluation of Boolean expressions, similar to an if-then-else or a switch construct in a programming language.

```
<choose>
  <when condition="Boolean expression | Boolean constant">
    <!-- one or more policy statements to be applied if the above con
  </when>
  <when condition="Boolean expression | Boolean constant">
    <!-- one or more policy statements to be applied if the above con
  </when>
  <otherwise>
    <!-- one or more policy statements to be applied if none of the a
  </otherwise>
</choose>
```

Control flow Policy

The control flow policy must contain at least one `<when/>` element. The `<otherwise/>` element is optional. Conditions in `<when/>` elements are evaluated in order of their appearance within the policy.

Policy statement(s) enclosed within the first `<when/>` element with condition attribute equals true will be applied. Policies enclosed within the `<otherwise/>` element, if present, will be applied if all of the `<when/>` element condition attributes are false.



Forward Request Policy

The **forward-request** policy forwards the incoming request to the backend service specified in the request context. The backend service URL is specified in the API settings and can be changed using the set backend service policy.

```
<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 Policy

The **limit-concurrency** policy prevents enclosed policies from executing by more than the specified number of requests at any time.

```
<limit-concurrency key="expression" max-count="number">  
    <!-- nested policy statements -->  
</limit-concurrency>
```

Upon exceeding that number, new requests will fail immediately with a *429 Too Many Requests* status code.

Log to Eventhubs Policy

The log-to-eventhub policy sends messages in the specified format to an Event Hub defined by a Logger entity.

```
<log-to-eventhub logger-id="id of the logger entity" partition-id="index  
  Expression returning a string to be logged  
</log-to-eventhub>
```

As its name implies, the policy is used for saving selected request or response context information for online or offline analysis.

Retry Policy

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 Policy

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.

```
<return-response response-variable-name="existing context variable">  
  <set-header/>  
  <set-body/>  
  <set-status/>  
</return-response>
```

When both are provided, the response contained within the context variable is modified by the policy statements before being returned to the caller. 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

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.

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

It generates sample responses from schemas, when schemas are provided and examples are not. If neither examples-schemas are found, responses with no content are returned.

Securing APIs :

- using subscription keys and keys
 - using client certificates

Subscriptions in Azure API Management

- When APIs are published through API Management, it's easy and common to secure access to those APIs by using subscription keys.
- Developers who need to consume the published APIs must include a valid subscription key in HTTP requests when they make calls to those APIs.
- Otherwise, the calls are rejected immediately by the API Management gateway. They aren't forwarded to the back-end services.
- To get a subscription key for accessing APIs, a subscription is required.
- A subscription is essentially a named container for a pair of subscription keys. Developers who need to consume the published APIs can get subscriptions.
- And they don't need approval from API publishers. API publishers can also create subscriptions directly for API consumers.

Azure services



Subscription Key

- A subscription key is a unique auto-generated key that can be passed through in the headers of the client request or as a query string parameter.
- The key is directly related to a subscription, which can be scoped to different areas.
- Subscriptions give you granular control over permissions and policies.
- Applications that call a protected API must include the key in every request.
- You can regenerate these subscription keys at any time, for example, if you suspect that a key has been shared with unauthorized users.



Subscriptions and Keys

The three main subscription scopes are:

Scope	Details
All APIs	Applies to every API accessible from the gateway
Single API	This scope applies to a single imported API and all of its endpoints
Product	A product is a collection of one or more APIs that you configure in API Management. You can assign APIs to more than one product. Products can have different access rules, usage quotas, and terms of use.

API Management Overview

Every subscription has two keys, a primary and a secondary. Having two keys makes it easier when you do need to regenerate a key. For example, if you want to change the primary key and avoid downtime, use the secondary key in your apps.

For products where subscriptions are enabled, clients must supply a key when making calls to APIs in that product. Developers can obtain a key by submitting a subscription request. If you approve the request, you must send them the subscription key securely, for example, in an encrypted message. This step is a core part of the API Management workflow.



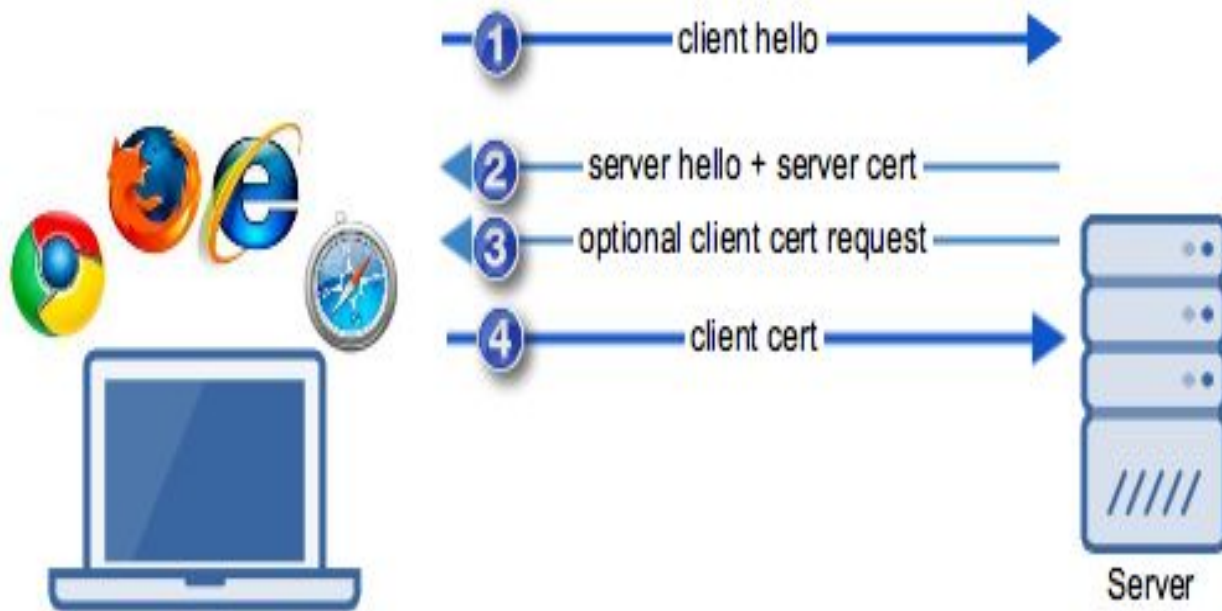
Calling an API with the subscription key

- 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 is subscription-key.
- To test out your API calls, you can use the developer portal, or command-line tools, such as curl.

Here's an example of a GET request using the developer portal, which shows the subscription key header:

- Here's how you can pass a key in the request header using curl:
- `curl --header "Ocp-Apim-Subscription-Key: <key string>" https://<apim gateway>.azure-api.net/api/path`
- Here's an example curl command that passes a key in the URL as a query string:
- `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.

Securing access to API using client certificates



Certificates can be used to provide TLS mutual authentication between the client and the API gateway. You can configure the API Management gateway to allow only requests with certificates containing a specific thumbprint. The authorization at the gateway level is handled through inbound policies.

TLS client authentication

With TLS client authentication, the API Management gateway can inspect the certificate contained within the client request and check for properties like:

Property	Reason
Certificate Authority (CA)	Only allow certificates signed by a particular CA
Thumbprint	Allow certificates containing a specified thumbprint
Subject	Only allow certificates with a specified subject
Expiration Date	Only allow certificates that have not expired

API Management Overview

These properties are not mutually exclusive and they can be mixed together to form your own policy requirements. For instance, you can specify that the certificate passed in the request is signed by a certain certificate authority and hasn't expired.


Client certificates are signed to ensure that they are not tampered with. When a partner sends you a certificate, verify that it comes from them and not an imposter.



API Management Overview

There are two common ways to verify a certificate:

- Check who issued the certificate. If the issuer was a certificate authority that you trust, you can use the certificate. You can configure the trusted certificate authorities in the Azure portal to automate this process.
- If the certificate is issued by the partner, verify that it came from them. For example, if they deliver the certificate in person, you can be sure of its authenticity. These are known as self-signed certificates.



Add certificate

Certificate name * ⓘ
myCert ✓

Certificate .pem or .cer file. ⓘ
"fullchain.pem" 📁

☒ Set certificate status to verified on upload ⓘ

ⓘ We'll verify this certificate automatically, with no manual verification steps required. [Learn more](#)

Accepting client certificates in the Consumption tier



The Consumption tier in API Management is designed to conform with serverless design principals.

If you build your APIs from serverless technologies, such as Azure Functions, this tier is a good fit. In the Consumption tier, you must explicitly enable the use of client certificates, which you can do on the Custom domains page. This step is not necessary in other tiers.

API Management Overview

Certificate Authorization Policies

Create these policies in the inbound processing policy file within the API Management gateway:

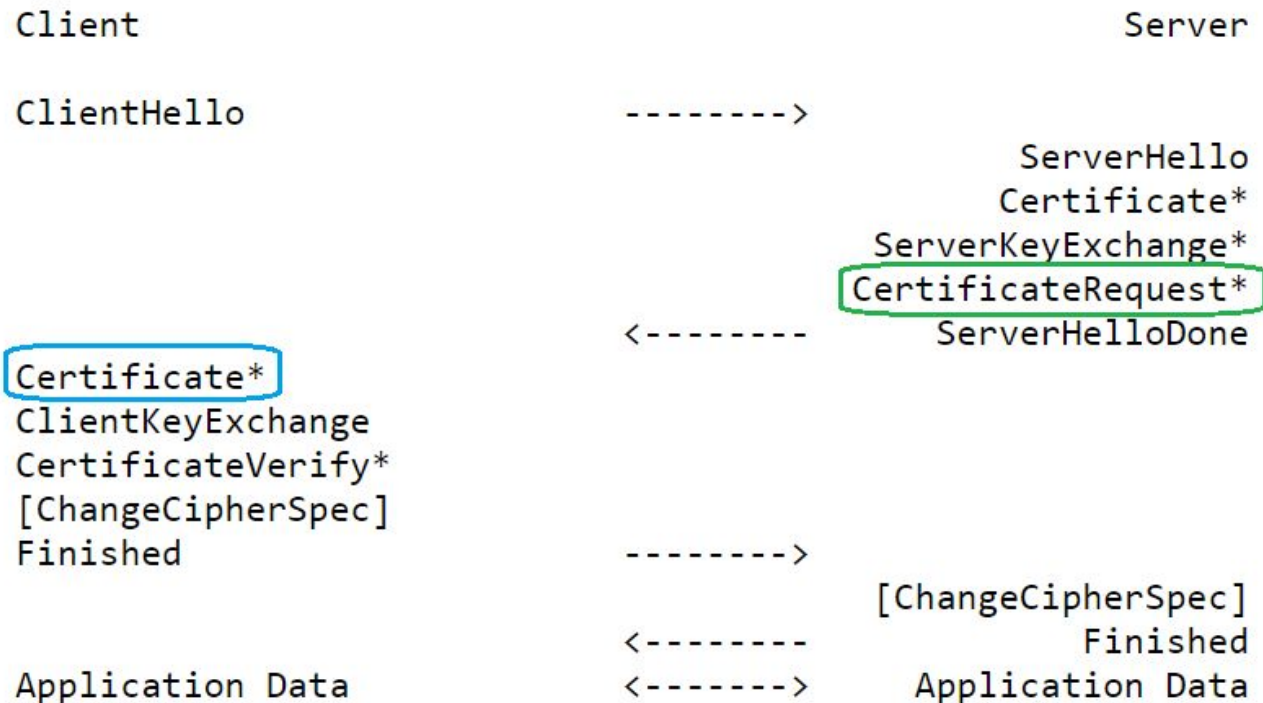


Figure 1. Message flow for a full handshake

Check the thumbprint of a client certificate

Every client certificate includes a thumbprint, which is a hash, calculated from other certificate properties.

```
<choose>
  <when condition="@context.Request.Certificate == null || context.Re
    <return-response>
      <set-status code="403" reason="Invalid client certificate" /
    </return-response>
  </when>
</choose>
```

The thumbprint ensures that the values in the certificate have not been altered since the certificate was issued by the certificate authority. You can check the thumbprint in your policy. The above example checks the thumbprint of the certificate passed in the request:

Check the thumbprint against certificates uploaded to API Management

In the previous example, only one thumbprint would work so only one certificate would be validated. Usually, each customer or partner company would pass a different certificate with a different thumbprint.

```
<choose>
  <when condition="@context.Request.Certificate == null || !context.R
    <return-response>
      <set-status code="403" reason="Invalid client certificate" /
    </return-response>
  </when>
</choose>
```

To support this scenario, obtain the certificates from your partners and use the Client certificates page in the Azure portal to upload them to the API Management resource. Then add this above code to your policy:

Check the issuer and subject of a client certificate

This example checks the issuer and subject of the certificate passed in the request:

```
<choose>
  <when condition="@ (context.Request.Certificate == null || context.Re
    <return-response>
      <set-status code="403" reason="Invalid client certificate" /
    </return-response>
  </when>
</choose>
```



Monitor published APIs

Monitoring API

API Management emits metrics every minute, giving you near real-time visibility into the state and health of your APIs.

The following are the two most frequently used metrics:

- **Capacity** - helps you make decisions about upgrading/downgrading your APIM services. The metric is emitted per minute and reflects the gateway capacity at the time of reporting. The metric ranges from 0-100 calculated based on gateway resources such as CPU and memory utilization.
- **Requests** - helps you analyze API traffic going through your API Management services. The metric is emitted per minute and reports the number of gateway requests with dimensions including response codes, location, hostname, and errors.



Azure Monitor

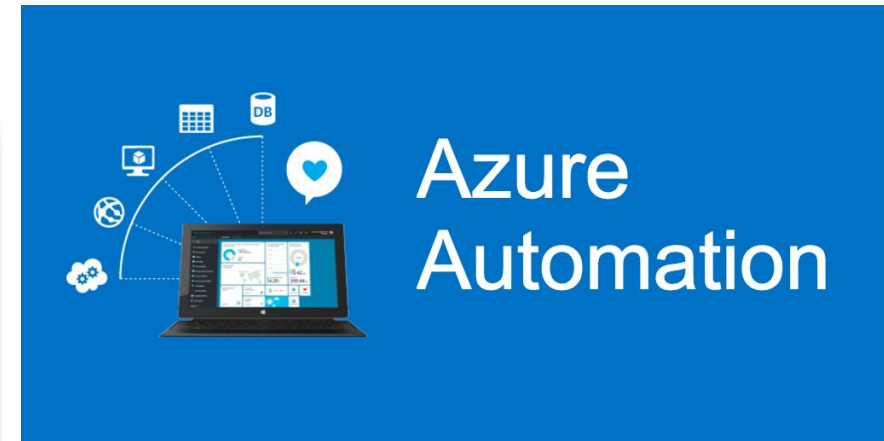


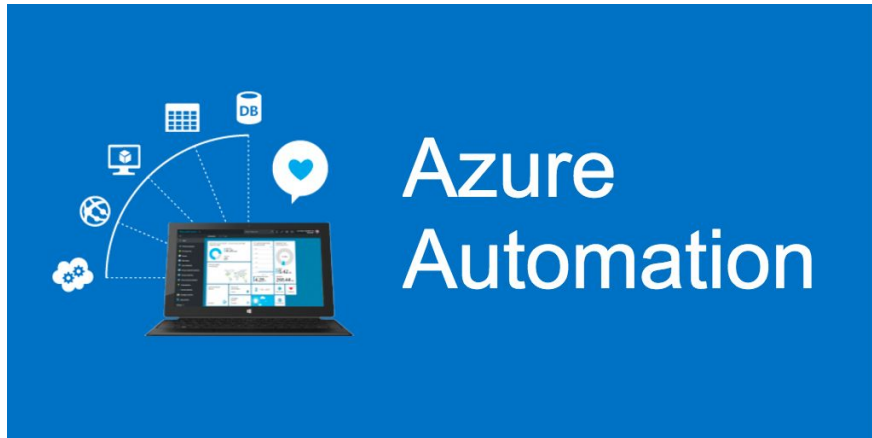
Azure Automation

AZURE AUTOMATION

Azure Automation is an Azure service for simplifying cloud management through process automation. Using Azure Automation, manual, repeated, long-running, and error-prone tasks can be automated to increase reliability, efficiency, and time to value for your organization.

Azure Automation provides a highly reliable, highly available workflow execution engine that scales to meet your needs. In Azure Automation, processes can be kicked off manually, by 3rd-party systems, or at scheduled intervals so that tasks happen exactly when needed.





How can Azure Automation help manage Azure API Management?

API Management can be managed in Azure Automation by using the [Windows PowerShell cmdlets for Azure API Management API](#). Within Azure Automation, you can write PowerShell workflow scripts to perform many of your API Management tasks using the cmdlets. You can also pair these cmdlets in Azure Automation with the cmdlets for other Azure services, to automate complex tasks across Azure services and 3rd party systems.

API Management soft-delete (preview)

With API Management soft-delete (preview), you can recover and restore recently deleted API Management (APIM) instances.



API Management soft-delete (preview)

Azure will schedule the deletion of the underlying data corresponding to APIM instance for execution after the predetermined (48 hour) retention interval. The DNS record corresponding to the instance is also retained for the duration of the retention interval. You cannot reuse the name of an API Management instance that has been soft-deleted until the retention period has passed.

If your APIM instance is not recovered within 48 hours, it will be hard deleted (unrecoverable). You can also choose to [purge](#) (permanently delete) your APIM instance, forgoing soft-delete retention period.

How can we help you?





India: +91-7022374614

US: 1-800-216-8930 (TOLL FREE)



sales@intellipaat.com



**24/7 Chat with Our Course
Advisor**