Azure API Management Best Practices

An overview

ENTERPRISE PRODUCTIVIT

Contents

What is API Management?

API Management is the practice of managing application programming interfaces (APIs), often using scalable enterprise software for API creation, publication, security, monitoring, and analytics. API management enables enterprises or developers that publish or consume an API to monitor the interface's lifecycle and ensure that the API is performing as it was designed.

Microsoft's AZURE API management tools support open API specifications like Swagger, WADL (Web Application Description Language), and WSDL (Web Services Description Language).

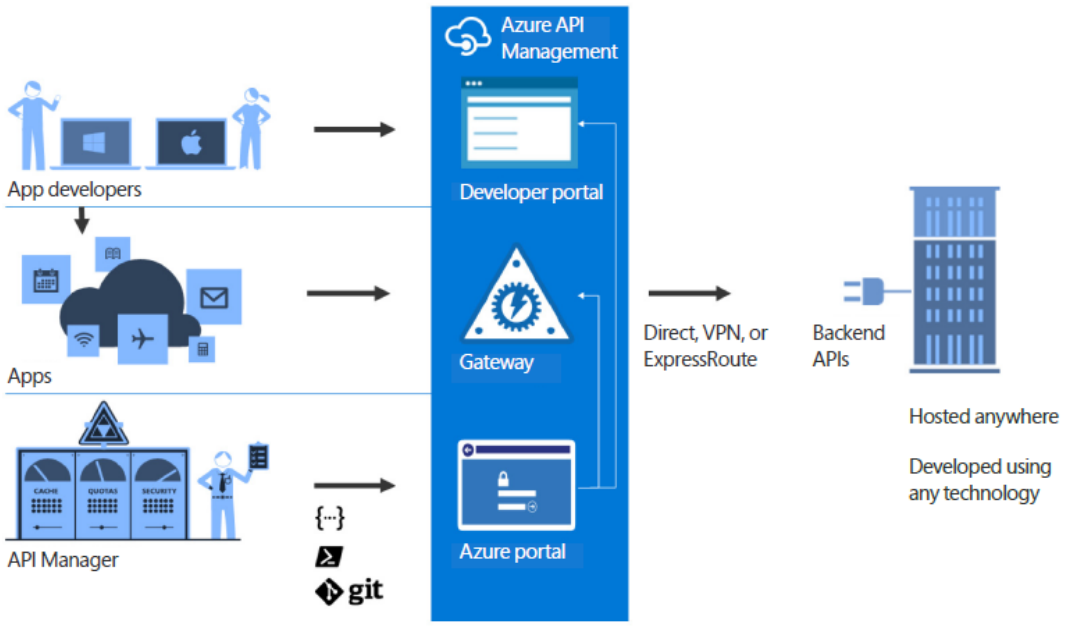
RESTful APIs have become synonymous to the term APIs because they use the capabilities of the existing HTTP supported methods like GET, PUT, POST, DELETE, and are lightweight. With the support of JSON, the request and response aren't needed to transform, as they can easily be understood by all the JavaScript frameworks.

What Is the Purpose of API Management?

- Helps in packaging and deployment of the APIs.
- Publishes the APIs to developers and other consumers.
- Provides documentation and sample requests/responses for developers to understand the APIs.
- Provides an API console that you can use to test the API operations. (Most of API management tools provide mock endpoints that we can test before testing the actual endpoints.)
- It acts a proxy or façade for existing backend services.
- To provide reliability of the service and analytics dashboard.

How Azure API Management Works:

A Brief Overview The following diagram provides a high-level overview of how Azure API Management works:



API Gateway	Azure portal	Developer portal
The API Gateway is the endpoint that: • Accepts API calls and routes them to your backend services—even if your backend services are hosted on-premises or on an isolated private network • Verifies API keys, JSON Web Tokens (JWTs), certificates, and other credentials • Enforces usage quotas and rate limits • Transforms your API on the fly, without code modifications • Caches backend responses where this has been set up.Logs metadata on API calls for analytics purposes	The Azure portal provides the administrative interface where you implement your APIs. You can use it to: • Define or import API schema • Package APIs into products • Set up policies such as quotas or transformations on your APIs • Manage users • Get insights from built-in analytics	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

Azure API Management runs on the Microsoft Azure platform as a private, single-tenant cloud implementation. This provides a secure, isolated environment with an allocated set of resources, enhancing performance and privacy. It also ensures predictable performance, enables governance, and eliminates the “noisy neighbor” problem that can be characteristic of multi-tenant applications. (Noisy neighbor is a phrase used to describe a cloud computing infrastructure co-tenant that monopolizes bandwidth, disk I/O, CPU and other resources, and can negatively affect other users’ cloud performance.)

Best Practices for RESTful API:

- Use nouns but no verbs: Do not use verbs such as /getAllCars, /createNewCar, /deleteAllRedCars
- GET method and query parameters should not alter the state: Use PUT, POST and DELETE methods instead of the GET method to alter the state. Do not use GET for state changes
- Use plural nouns: Do not mix up singular and plural nouns. Keep it simple and use only plural nouns for all resources.
- Use sub-resources for relations: If a resource is related to another resource use sub resources.
- Use HTTP headers for serialization formats: Both, client and server, need to know which format is used for the communication. The format has to be specified in the HTTP-Header. Content-Type defines the request format. Accept defines a list of acceptable response formats.
- Use HATEOAS: Hypermedia as the Engine of Application State is a principle that hypertext links should be used to create a better navigation through the API.
- Provide filtering, sorting, field selection and paging for collections
- Filtering: Use a unique query parameter for all fields or a query language for filtering.
- Sorting: Allow ascending and descending sorting over multiple fields.
- Field selection: Mobile clients display just a few attributes in a list. They don't need all attributes of a resource. Give the API consumer the ability to choose returned fields. This will also reduce the network traffic and speed up the usage of the API.
- Paging: Use limit and offset. It is flexible for the user and common in leading databases. The default should be limit=20 and offset=0
- Version your API: Make the API Version mandatory and do not release an unversioned API. Use a simple ordinal number and avoid dot notation such as 2.5.
- Handle Errors with HTTP status codes: It is hard to work with an API that ignores error handling. Pure returning of a HTTP 500 with a stack trace is not very helpful.

- Use error payloads: All exceptions should be mapped in an error payload
- Allow overriding HTTP method: Some proxies support only POST and GET methods. To support a RESTful API with these limitations, the API needs a way to override the HTTP method. Use the custom HTTP Header X-HTTP-Method-Override to override the POST Method.

References:

1. https://blog.mwaysolutions.com/2014/06/05/10-best-practices-for-better-restful-api/

2. https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

3. http://azpodcast.com/post/Episode-226-API-Management-Best-Practices

4. https://blog.mexia.com.au/azure-api-management-ci/cd-best-practices

5. https://blog.kloud.com.au/2016/09/09/azure-api-management-step-by-step/

6. https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9?gi=377c02d71e14