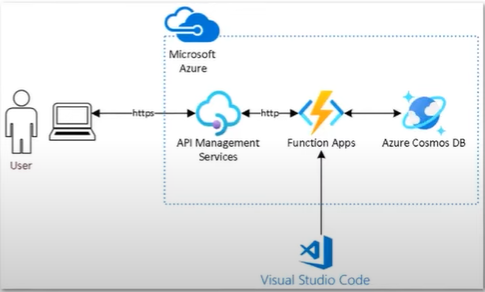
**Azure APIM**

Azure API Management (APIM) integrates existing back-end services into modern **API gateways.**



It follows the API-first approach, **decoupling** **front-end** and **back-end** teams with the help of API mocking.

Azure API Management handles the full management of APIs.

It **centralizes** the security, versioning, documentation, and compliance of your **back-end -services** in a **single** point.

**KEY CONCEPTS:**

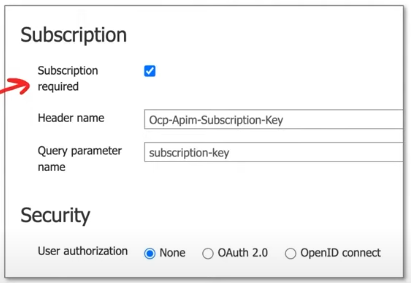
* API represents a set of **operations**. API Operation connects an API endpoint to its backend.
* Product: A logical grouping of APIs
* A single or a group of APIs make up a product, which is how APIs are presented to developers. It can be either public or private.
* Backend represents back-end services in API.
* Group, used to manage the visibility of the products to developers:
  + **Administrators** have full access to the API management.
  + **Developers**, users with access to the developer’s portal with permissions to build applications.
  + **Guests,** users without access to the developer’s portal but with reading permissions in some services.
* **Developer** belongs to one or more Product groups, and each developer has a primary and secondary key to call the product’s APIs.
* **Policies:** configurations and validations that are applied in progress to incoming requests and the outcome responses.
* **Named Values:** key-value pairs used with policies. Valued can be a result of an expression.
* **Gateway:** where API calls are received, and policies are applied to incoming requests.
* **Developer Portal:** where developers can access all APIs and products listed by APIM alongside its API’s operation and documentation. Developers can also request access to APIs from the developer’s portal.

**Echo API service**

When the APIM gateway is created by default Echo API is created which is a non-production Azure service that is used to test Azure API Management.

**API Authentication**

In order to authenticate with our APIs, we configure those settings under the subscription section.



If the subscription is required, only developers with a valid access key can use it.

If it is not checked, **anonymous** requests are allowed.

Here, we can configure where the API will receive the access keys, which can be sent as a header or query string.

**Groups**

Groups are used to manage the visibility of products to developers.

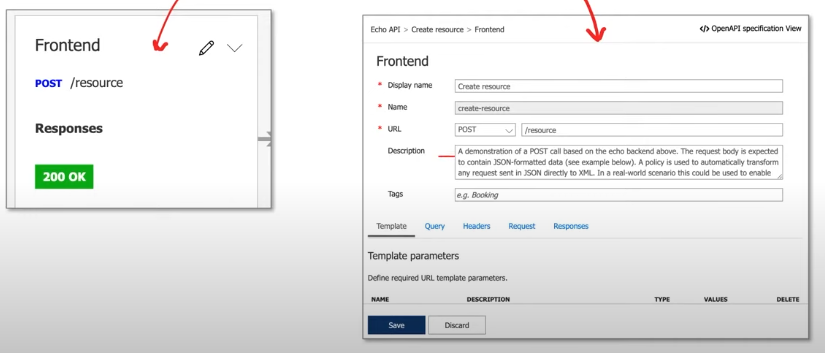
1. Administrators: Manage API management service instances and create the APIs, operations, and products that are used by developers.
2. Developers: Authenticated developer portal, users that build applications using APIs. Developers are granted access to the developer portal and build applications that call the operations of an API.
3. Guests: Unauthenticated developer portal users, such as prospective customers visiting the developer portal. They can be granted certain read-only access, such as the ability to view APIs but not to call them.

Administrators can also create custom groups or use external groups in an associated Azure Active Directory tenant to give developers visibility and access to API products.

A user can belong to more than one group.

**Frontends**

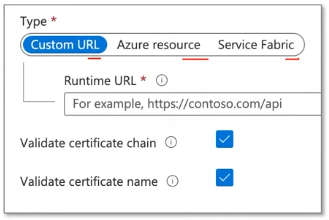
Frontends define the **route/endpoint** and the **documentation and configuration** around that endpoint.



API does not host APIs; it creates facades for APIs.

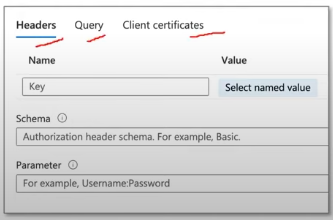
**Backends**

For Backends, we have:



* Custom URL: point to the server where the service is running.
* Azure Resource: Integrate directly to an Azure resource eg.
  + Azure Functions
  + App Service
  + Container App
  + Logic App
* Azure Service Fabric

**Authorization credentials** present authorization request credentials to the backend service.



* Headers: HTTP headers
  + Can fetch from Named values.
* Query: query string
  + Can fetch from Named values.
* Client certificates: x.509 certificates
  + Certificates stored in Azure key vault.

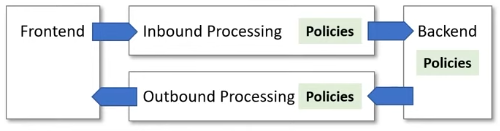
**Policies**

API Management Policies allow to change the behavior at multiple stages of an endpoint’s request lifecycle.

Any part of the request and response messages can be updated eg. headers, body, URLs, etc.

These are four areas where policies can be applied:

* Inbound: for incoming requests.
* Backend: before requests reach to backend.
* Outbound: before sending a response back to the client.
* Error: when a request encounters an error.

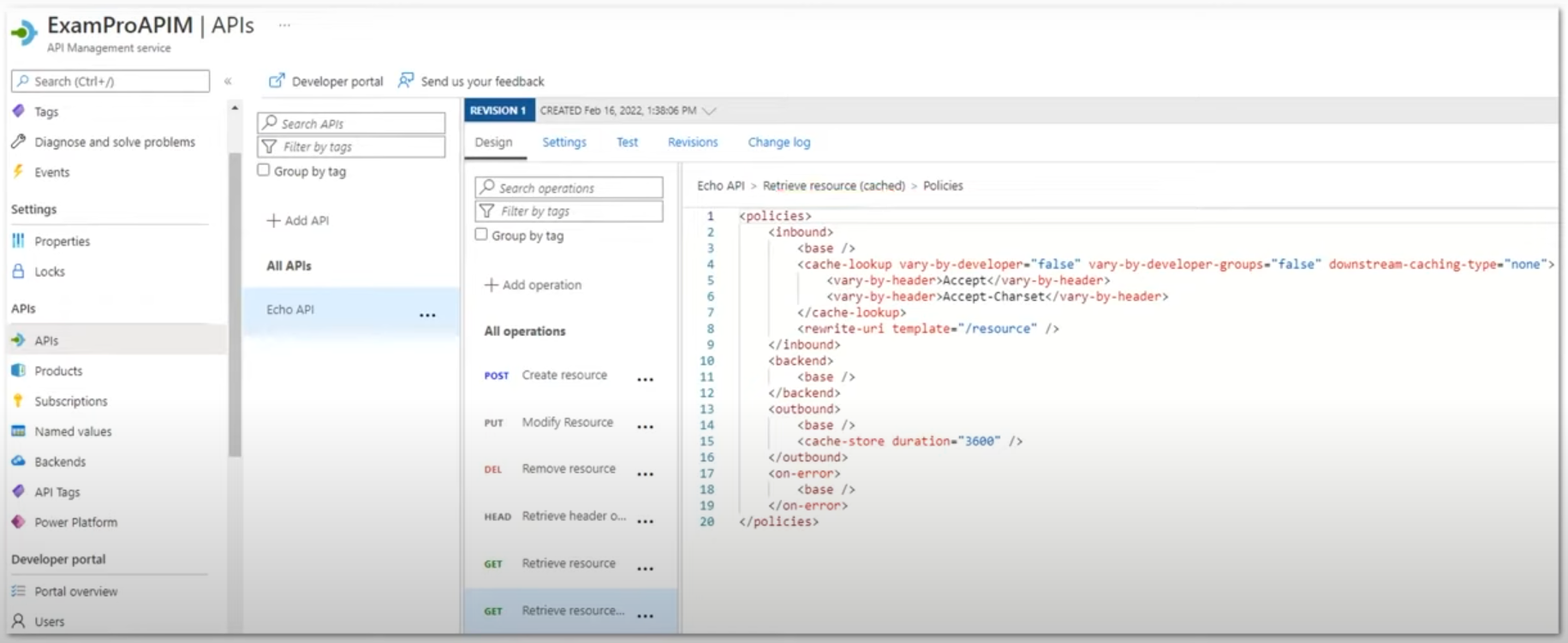


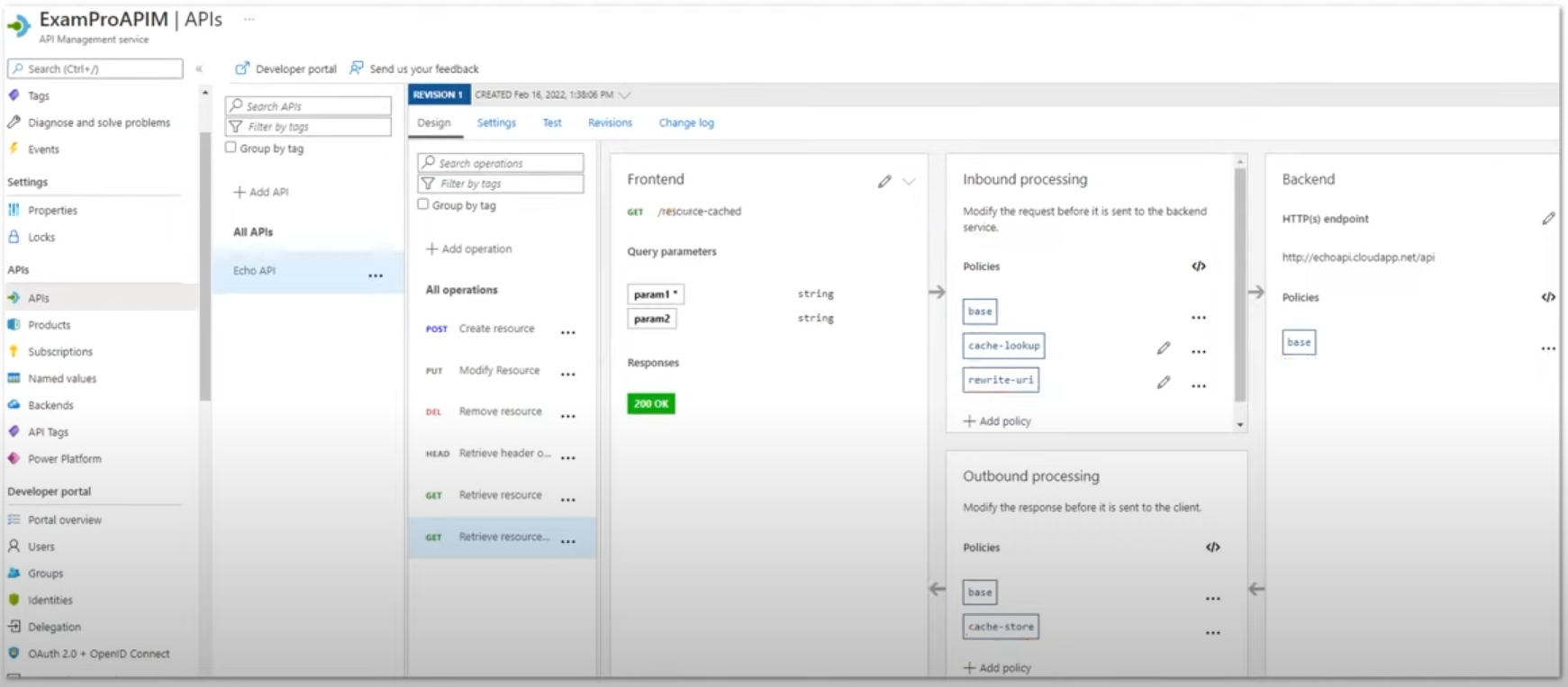
Azure has a collection of policy groups which contain many policies we can apply:

* Access restriction Policies
* Advanced Policies
* Authentication Policies
* Caching Policies
* Cross-Domain Policies
* Transformation Policies
* Dapr Integration Policies
* Validation Policies
* Graph QL Validation Policies

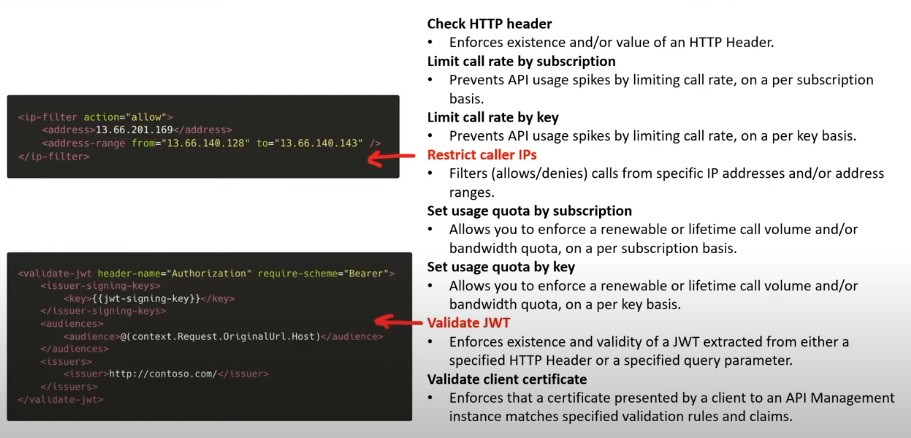
When an error occurs, no other policies are applied except the error policies; however, if other policies were in effect before the error, they will not be removed.

Product-level policies apply to all API operations within a product.

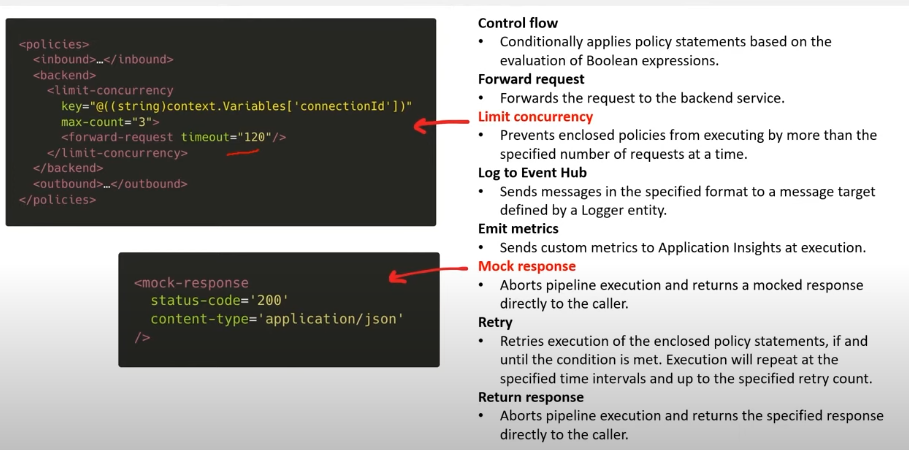
APIM Authentication Policies Example:  


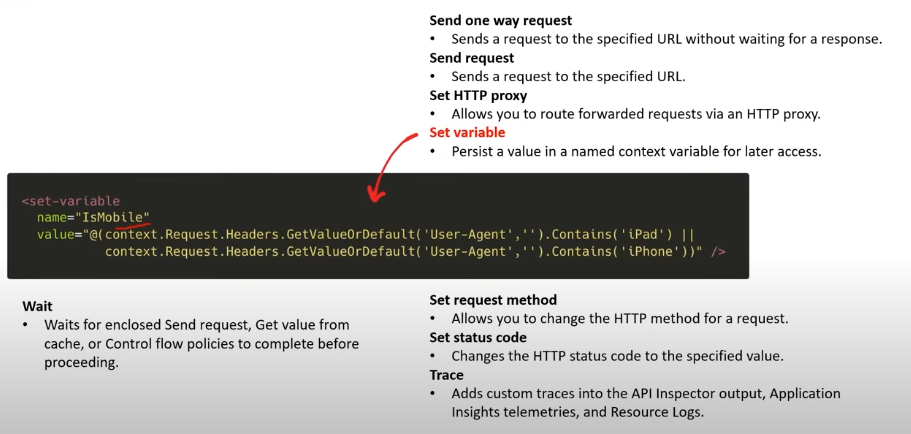


**Access Restriction Policies**

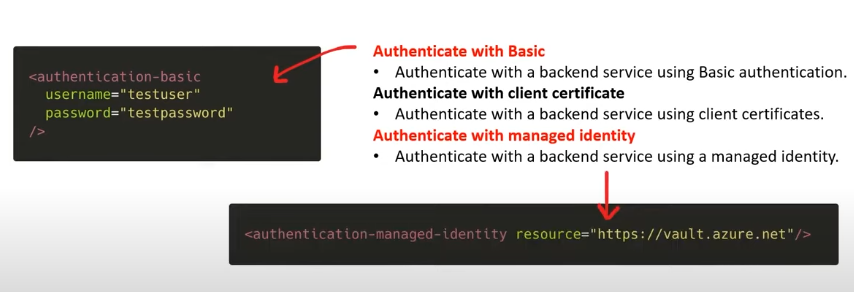


**Advanced Policies**

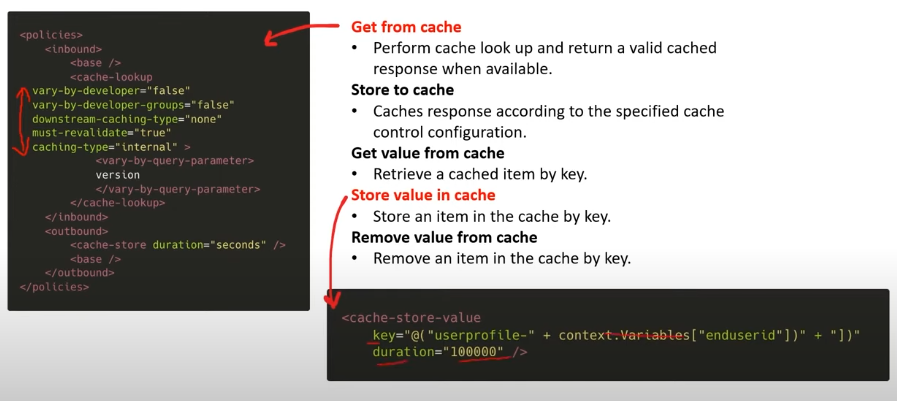




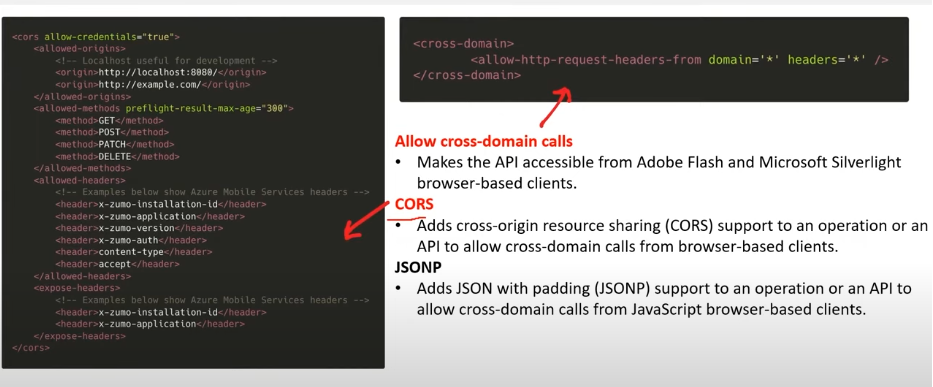
**Authentication Policies**



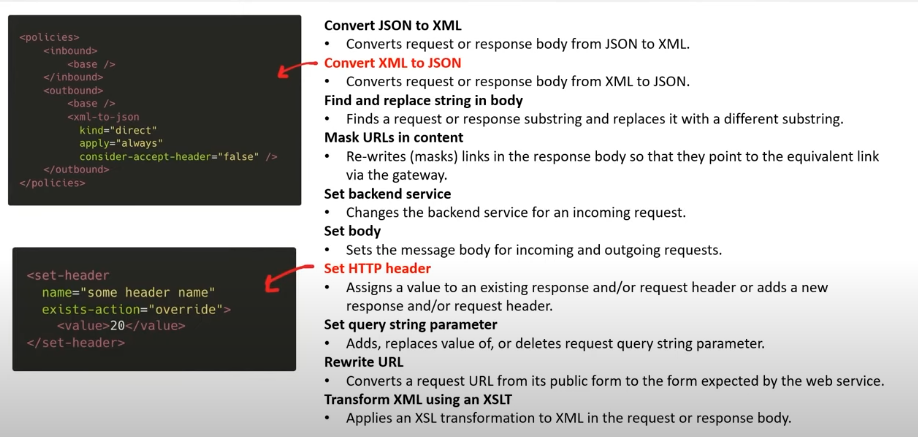
**Caching Policies**



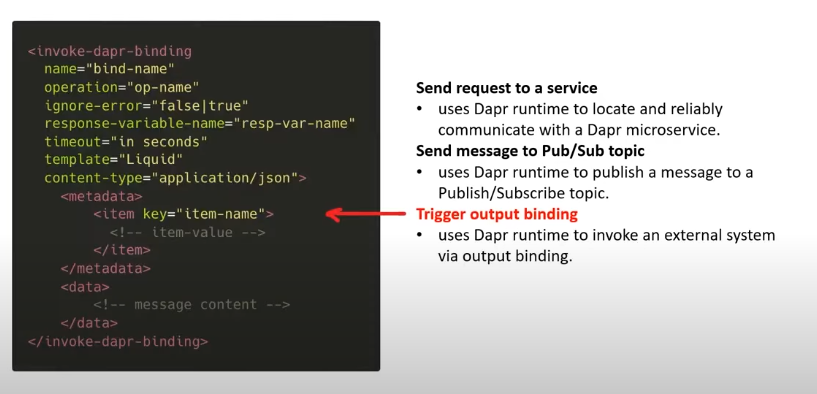
**Cross Domain Policies**



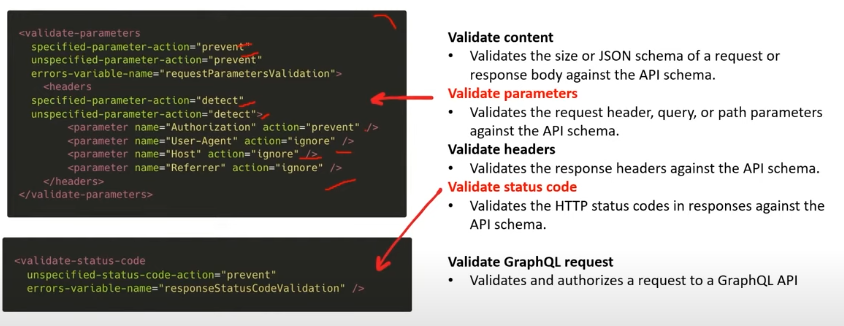
**Transformation Policies**



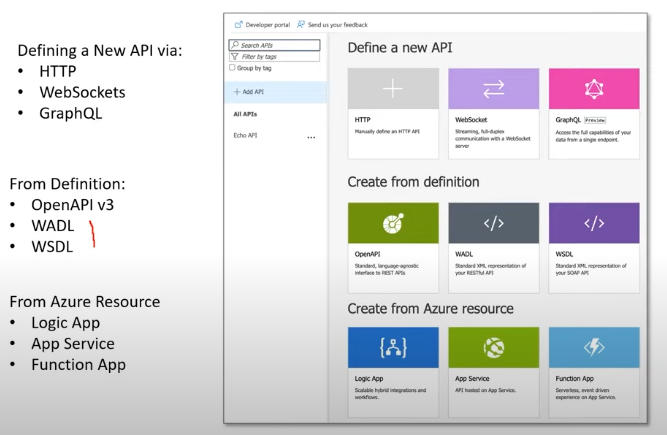
**Dapr Integration Policies**



**Validation Policies**



**APIs**



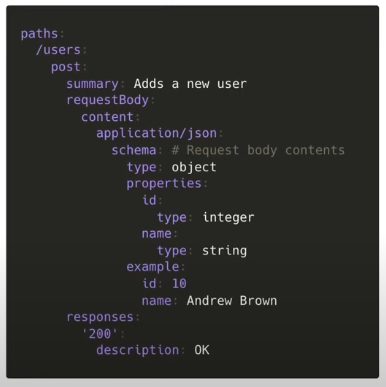
**OpenAPI**

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.

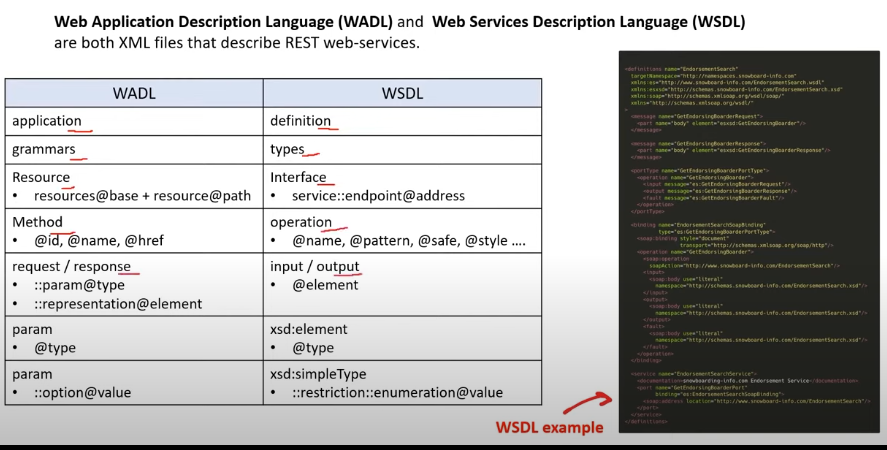
Swagger and OpenAPI used to be the same thing but as of OpenAPI V3, Swagger and OpenAPI are two different things

* OpneAPI = Specification
* Swagger = Tools for implementing the specification

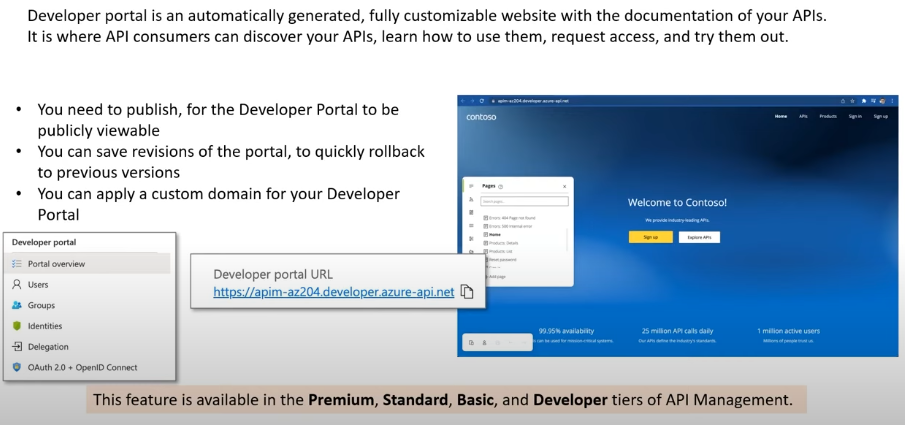
OpenAPI can be represented as either **JSON** or **YAML**.



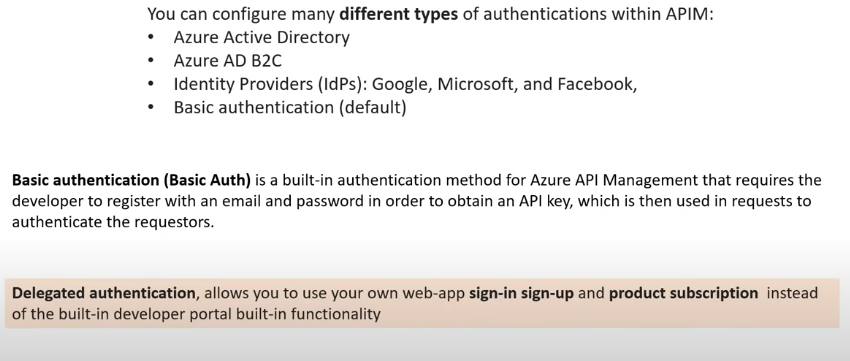
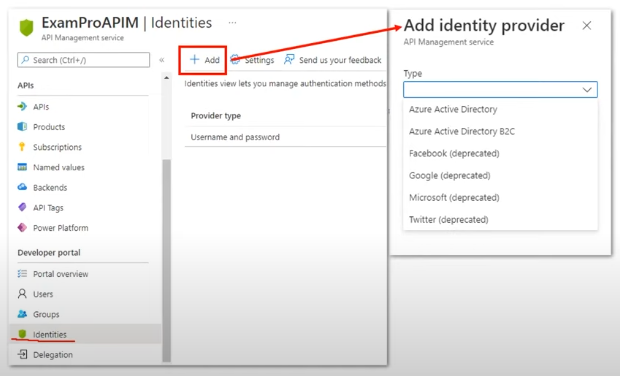
**WADL and WSDL**



**Developers Portal**



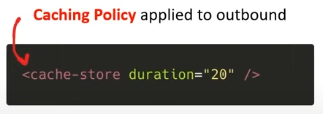
**Authentication – Developer Portal**

**Builtin Cache**

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.



The built-in cache is **volatile** and is **shared by all units in the same region** in the same API Management service.

**External Cache**

Redis cache can be utilized externally instead of using a built-in cache.

Using an external cache allows to overcome a few limitations of the built-in cache:

* Avoid having cache periodically cleared during API Management updates.
* Have more control over cache configuration.
* Cache more data than your API Management tier allows to
* Use caching with the Consumption tier of API Management
* Enable caching in the API Management self-hosted gateways.

We need to just provide a connection string to Redis Cache.

---------------------------------------------------------------------------------------------------------------------

**Practical Implementation**

Create APIM service:

1. Create a **Resource group** with a subscription and region.
2. Create a **Log Analytics Workspace** with a subscription, resource group.
   1. Instance details: Name, region.
3. Create an **Application Instance** with a subscription, resource group.
   1. Instance details: Name, Region, Resource mode.
   2. Workspace details: subscription, Log analytics workspace.
4. Create an **APIM service** with a subscription and resource group:
   1. Instance details: region, name, organization name, administrator email.
   2. Pricing Tier: Developer, standard, basic, premium, etc.

It takes 30-40 min. to get ready.

**RESOURCES:**

1. TLS and SSL: <https://www.youtube.com/watch?v=j9QmMEWmcfo>
2. Certificates: <https://www.youtube.com/watch?v=r1nJT63BFQ0>