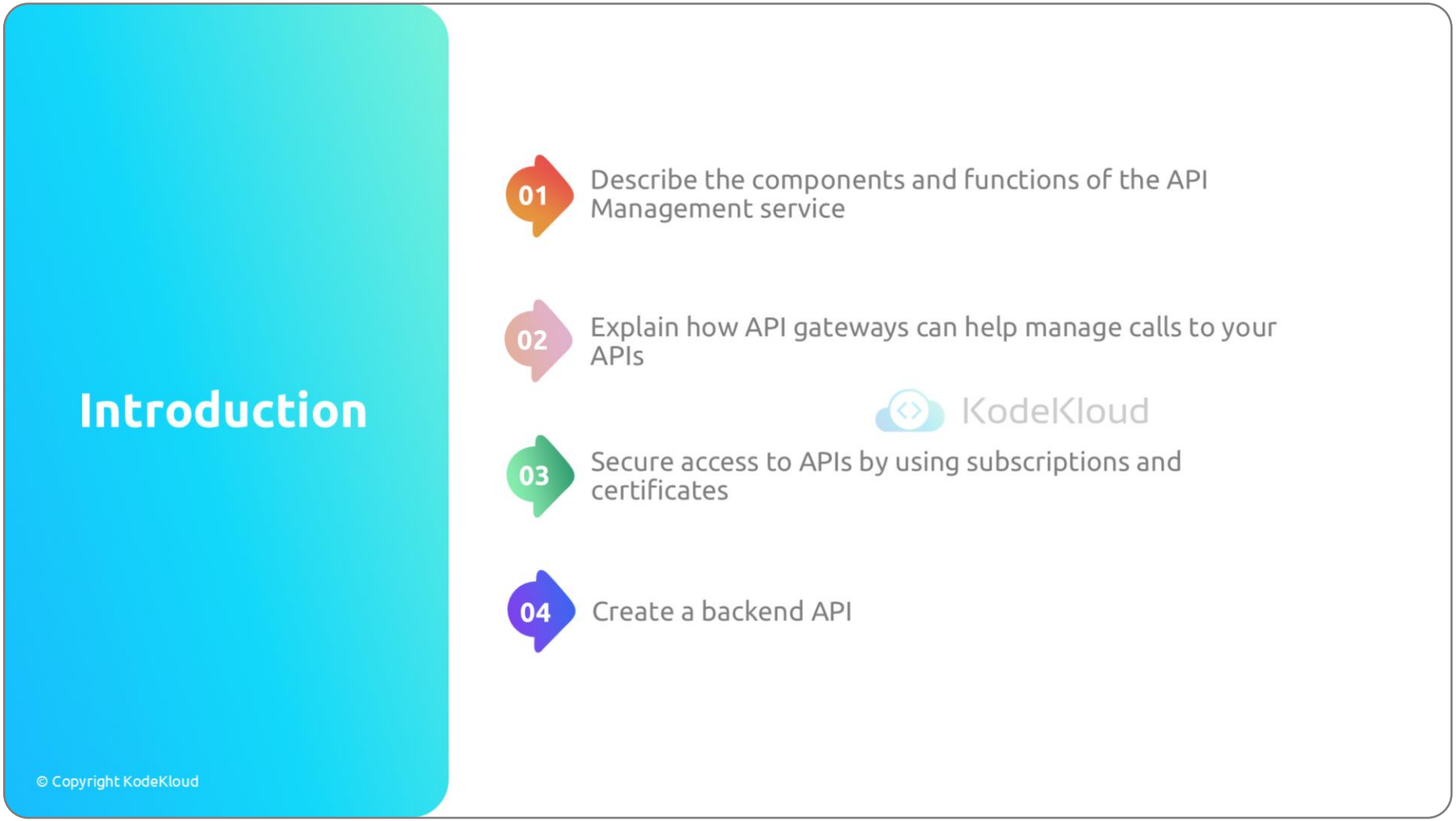
KodeKloud

Visit www.kodekloud.com to learn more.

Exploring API Management



Module overview

Discovering API Management Service

Role of API Management

Discovering API Management service In this module, we will dive into the role of a PA management, which is a crucial component when it comes to building scalable and secure API driven architectures

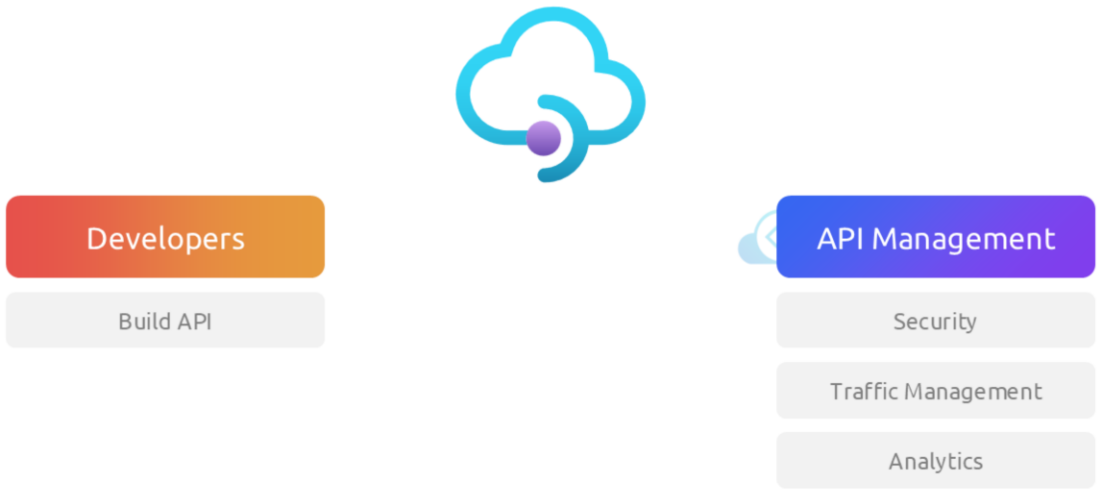
What is API Management?

Before we start, let's understand what is Azure API Management? Azure A PA Management is a fully managed service that allows organizations to publish, secure, transform, maintain, and monitor APIs

What is API Management?

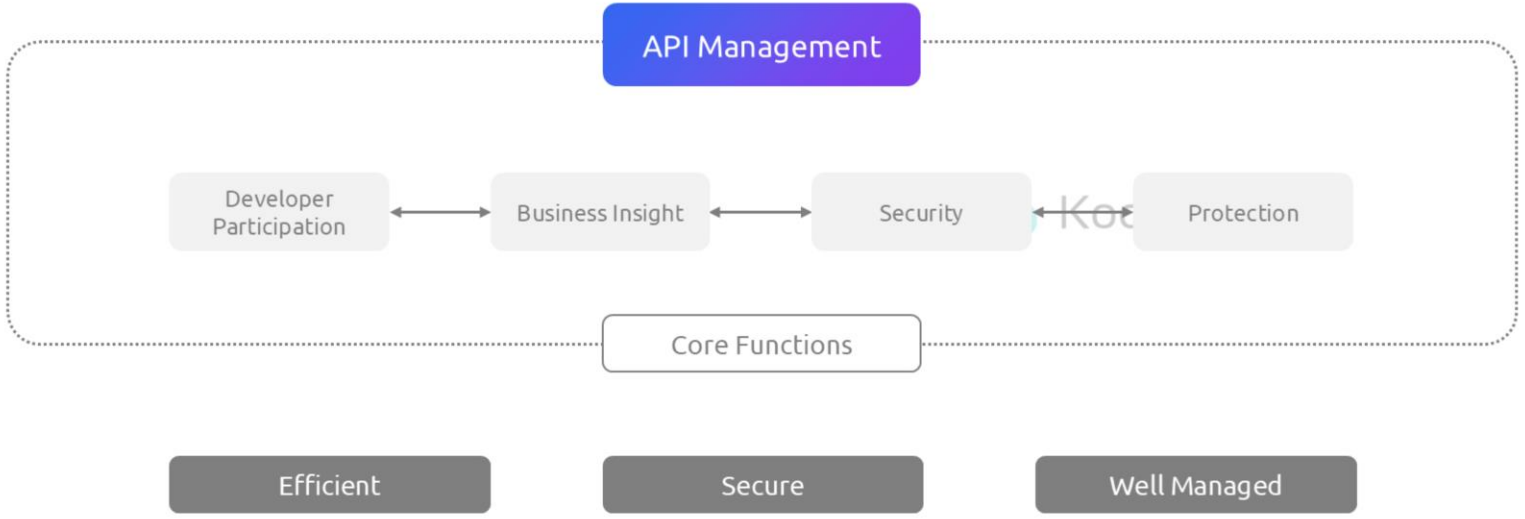
It act as a gateway between your API consumers, whether they are internal teams, partners, or external developers, or your backend services

What is API Management?



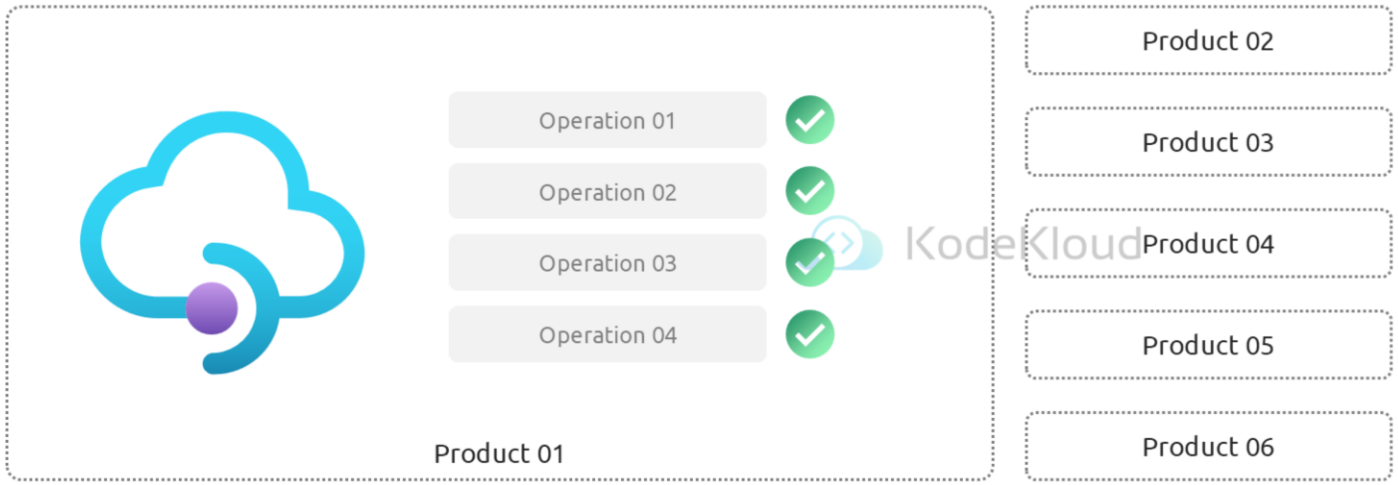
This enables developers to focus on building APIs While a PA management handles security, traffic management, analytics and more

What is API Management?



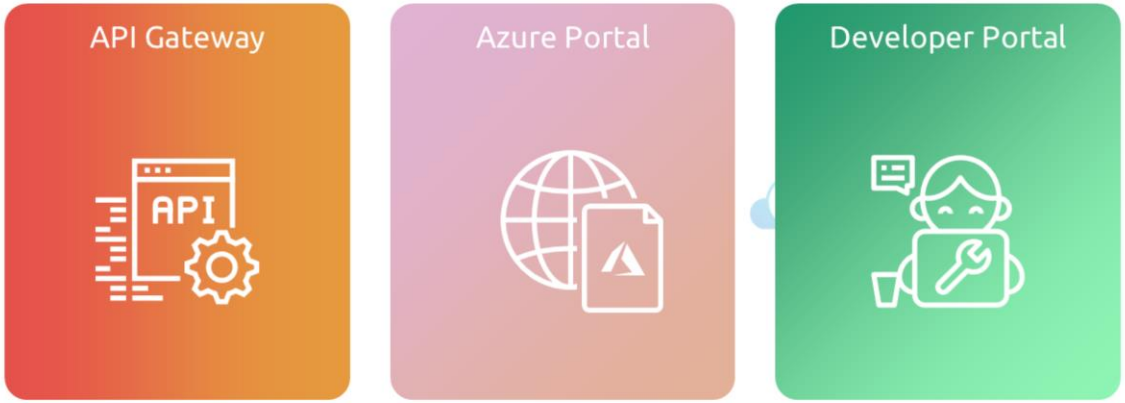
A PA management provides several core functions These include enabling developers to participate in the API program, providing business insights, enforcing security policies, and protecting APIs from salicious attacks or misuse With these core functions, API Management not only helps developers, but also ensures APIs are efficient, secure, and well managed each API

What is API Management?



Within Azure, a p management consists of one or more operations These operations represents the individual functions or endpoints of the API Additionally, APIs can be bundled into one or more products We'll talk about these components at a later point in this module So, in summary, API MANAGEMENT act as a backbone of any API driven system by providing necessary tools to ensure security, scalability, and developer engagement Now that you know the role of API management in an environment,

API Management Components



The system is made up of the following components:
The API gateway is the endpoint that:

- Accepts API calls and routes them to your backend(s).
- Verifies API keys, JWT tokens, certificates, and other credentials.
- Enforces usage quotas and rate limits.
- Transforms your API on the fly without code modifications.
- Caches backend responses where set up.

- Logs call metadata for analytics purposes.

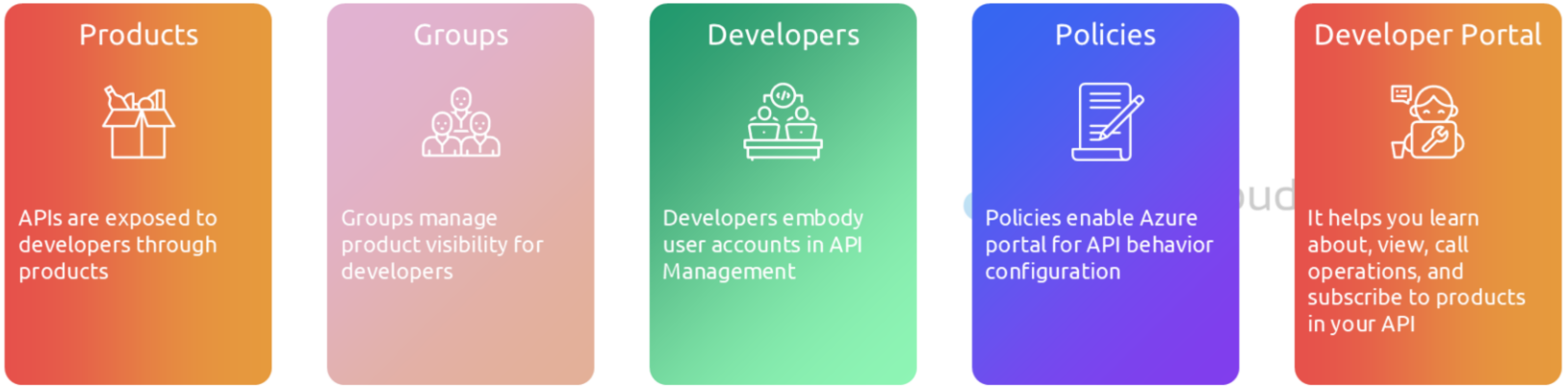
The Azure portal is the administrative interface where you set up your API program. Use it to:

- 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.

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.

Understanding APIs



Products
Products are how APIs are surfaced to developers. Products in API Management have one or more APIs, and are configured with a title, description, and terms of use. Products can be Open or Protected. Protected products must be subscribed to before they can be used, while open products can be used without a subscription. Subscription approval is configured at the product level and can either require administrator approval, or be auto-approved.

Groups

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

Administrators - Azure subscription administrators are members of this group. Administrators manage API Management service instances, creating the APIs, operations, and products that are used by developers.
Developers - Authenticated developer portal users fall into this group. Developers are the customers that build applications using your APIs. Developers are granted access to the developer portal and build applications that call the operations of an API.
Guests - Unauthenticated developer portal users, such as prospective customers visiting the developer portal of an API Management instance fall into this group. They can be granted certain read-only access, such as the ability to view APIs but not call them.
In addition to these system groups, administrators can create custom groups or leverage external groups in associated Azure Active Directory tenants.

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

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

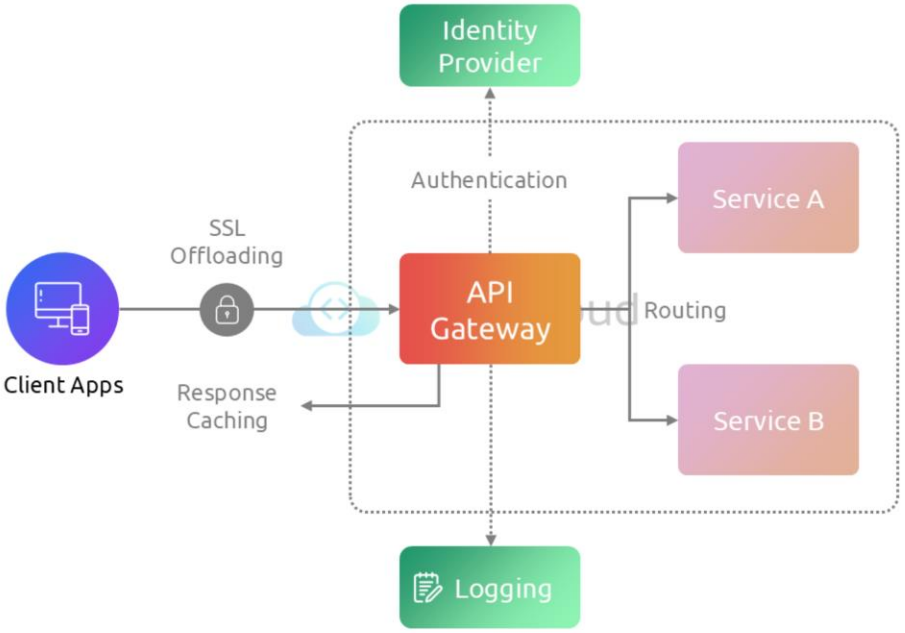
Developer portal
The developer portal is where developers can learn about your APIs, view and call operations, and subscribe to products. Prospective customers can visit the developer portal, view APIs and operations, and sign up. The URL for your developer portal is located on the dashboard in the Azure portal for your API Management

service instance.

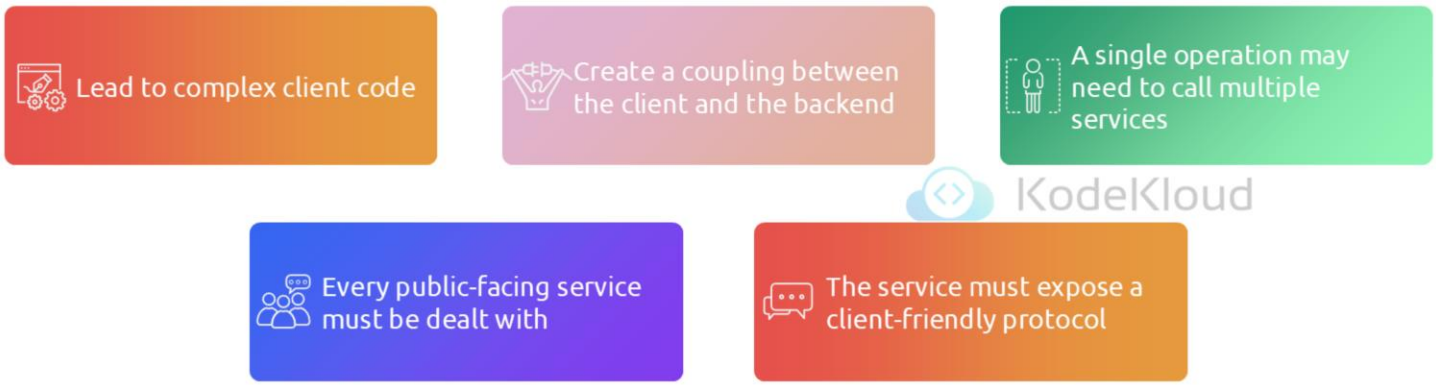
Exploring API Gateways

API Gateway Role

- An API gateway sits between clients and services.
- It acts as a reverse proxy, routing requests from clients to services.
- It may perform tasks such as authentication, SSL termination, and rate limiting.



Potential Issues When Deploying an API Without a Gateway



Potential issues when deploying an API without a gateway

The client must keep track of multiple endpoints, and handle failures in a resilient way.
The client needs to know how the individual services are decomposed. That makes it harder to maintain the client and also harder to refactor services.
A single operation might require calls to multiple services. That can result in multiple network round trips between the client and the server, adding significant latency.

Each public-facing service must handle concerns such as authentication, SSL, and client rate limiting.
Services must expose a client-friendly protocol such as HTTP or WebSocket. This limits the choice of communication protocols.
Services with public endpoints are a potential attack surface and must be hardened.

Functional Design Patterns

Functional design patterns
A gateway helps to address these issues by decoupling clients from services.
Gateways can perform a number of different functions, and you may not need all of them.

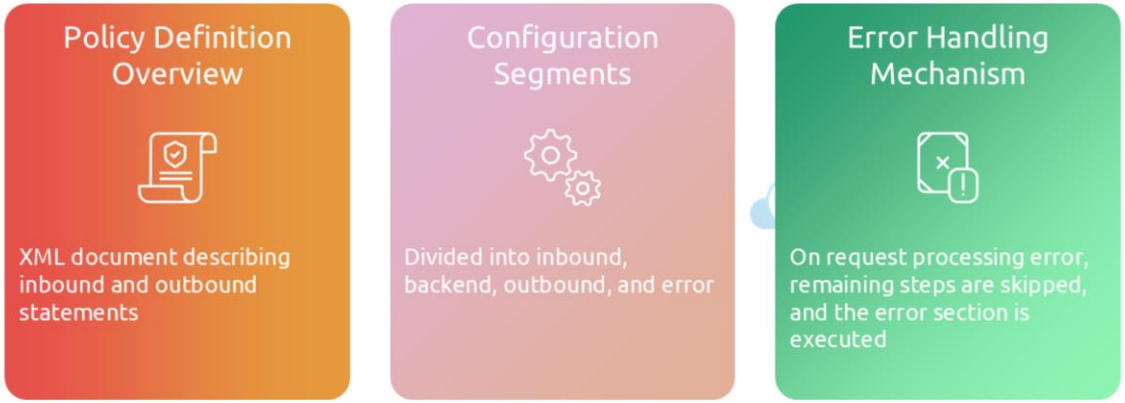
Here are some examples of functionality that could be offloaded to a gateway:
SSL termination

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

API Management Policies

Role of Policies

Policy Configuration



Policy Examples

Example 01

```
<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>
```

Example 02

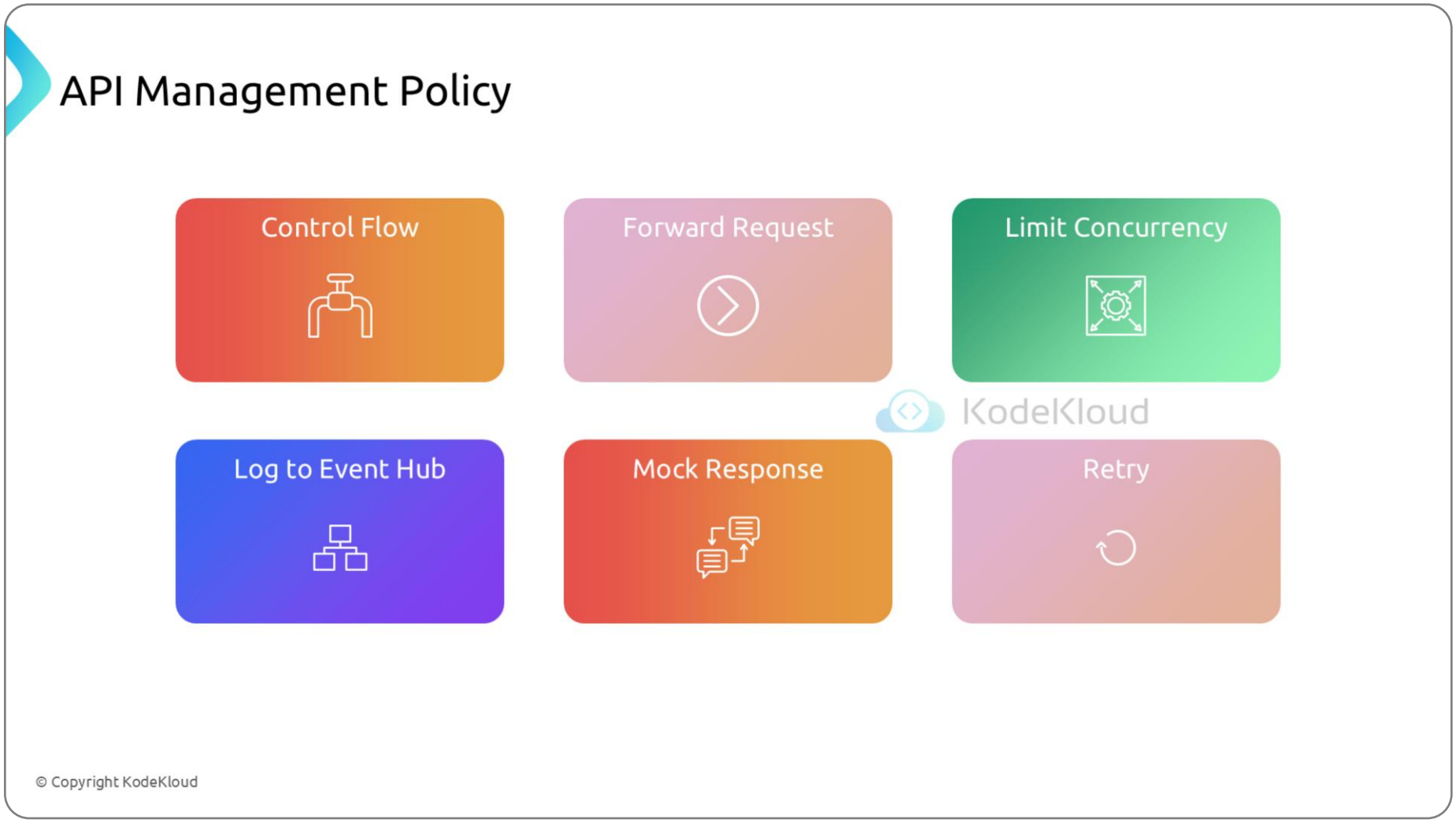
```
<policies>
    <inbound>
        <cross-domain />
        <base />
        <find-and-replace from="xyz" to="abc" />
    </inbound>
</policies>
```


Left example blank template
The policy definition is a simple XML document that describes a sequence of inbound and outbound statements. The XML can be edited directly in the definition window.
The configuration is divided into inbound, backend, outbound, and on-error. The series of specified policy statements executes in order for a request and a response.

Right example: policies specified at different scopes

If you have a policy at the global level and a policy configured for an API, then whenever that particular API is used both policies will be applied.
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.

Creating Advanced Policies



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

Additional resources

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

Examples

Example 01

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

KodeKloud

Example 02

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


Left example blank template
The policy definition is a simple XML document that describes a sequence of inbound and outbound statements. The XML can be edited directly in the definition window.
The configuration is divided into inbound, backend, outbound, and on-error. The series of specified policy statements executes in order for a request and a response.

Right example: policies specified at different scopes

If you have a policy at the global level and a policy configured for an API, then whenever that particular API is used both policies will be applied.
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.

Examples

Example 03

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

    <!-- Expression returning a string to be logged -->

</log-to-eventhub>
```


Left example blank template
The policy definition is a simple XML document that describes a sequence of inbound and outbound statements. The XML can be edited directly in the definition window.
The configuration is divided into inbound, backend, outbound, and on-error. The series of specified policy statements executes in order for a request and a response.

Right example: policies specified at different scopes

If you have a policy at the global level and a policy configured for an API, then whenever that particular API is used both policies will be applied.
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.

Securing APIs by Using Subscriptions

Securing APIs by using subscription

Authorized Applications

Azure AP Management Subscription Keys

Securing APIs by using subscriptions Securing APIs is a critical step to ensure that only authorized applications can interact with them Azure a p Management provides subscription keys to control access effectively

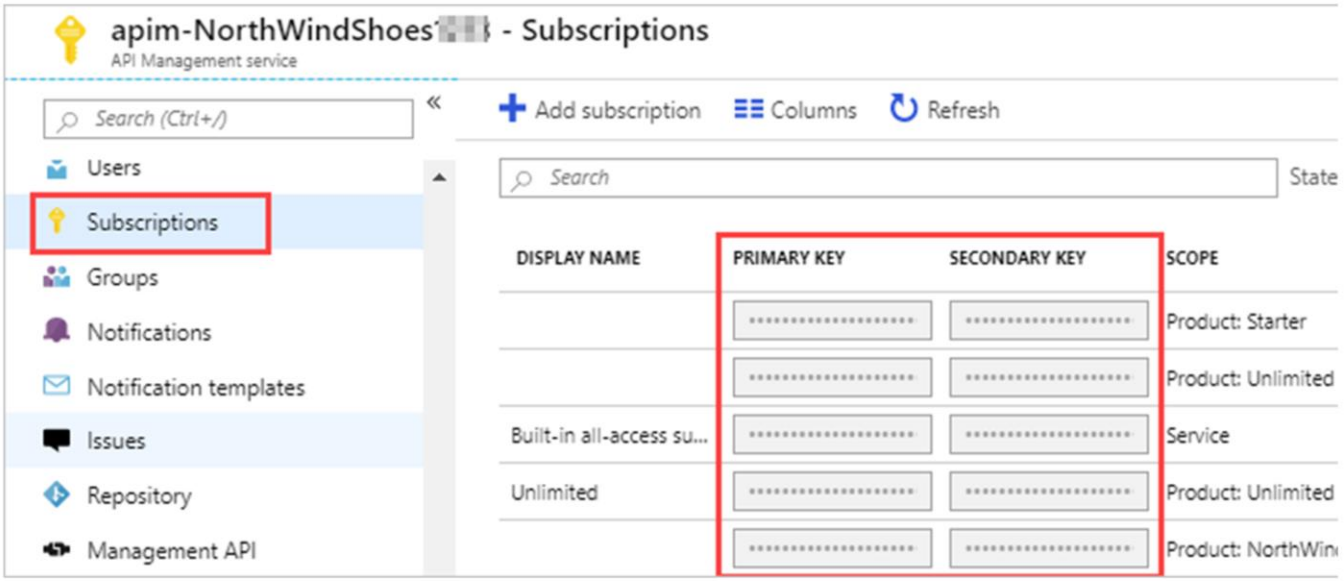
Subscription Key – Scopes

Scope	Details
All APIs	Applies to every API accessible from the gateway
Single API	Applies to a single imported API and all its endpoints
Product	A product is a group of configured APIs in API Management, assignable to multiple products with varying access rules, usage quotas, and terms

Note: API Management supports various access security mechanisms, such as OAuth 2.0, client certificates, and IP allow listing.

When you publish APIs through API Management, it's easy and common to secure access to those APIs by using subscription keys.
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.
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.

Securing API Consumer Applications



Must include the key in every request

You can regenerate these subscription keys at any time

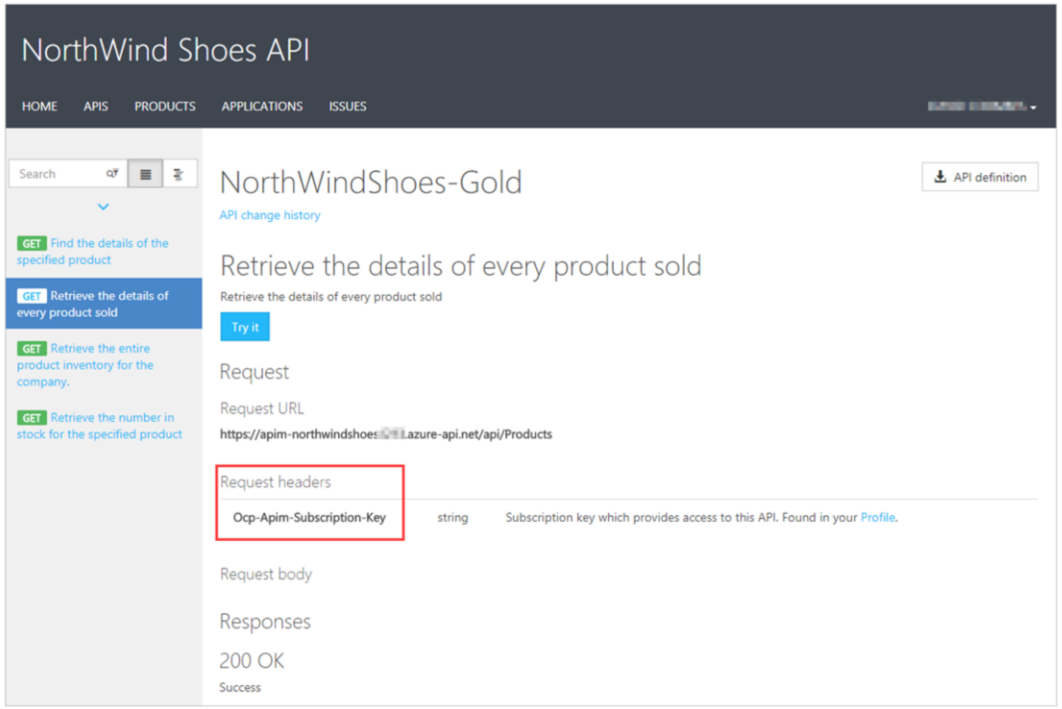
Every subscription has two keys – a primary and a secondary

API Access With Subscription Key

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**

Use the developer portal to test out API calls

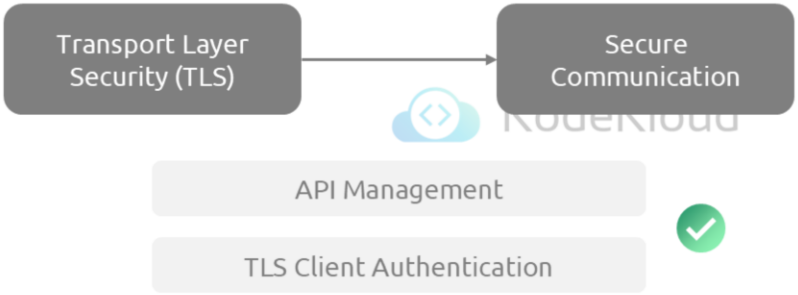


Here's how you can pass a key in the request header using curl:
curl --header "Ocp-Apim-Subscription-Key: <key string>" https://<apim gateway>.azure-api.net/api/path

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 APIs by Using Certificates

Securing APIs by using Certificates



Securing APIs by using certificates The TLS is a critical protocol for secure communication over a network The context of API management TLS client authentication is crucial to validate that the client interacting with the service is authorized and trusted

Transport Layer Security Client Authentication

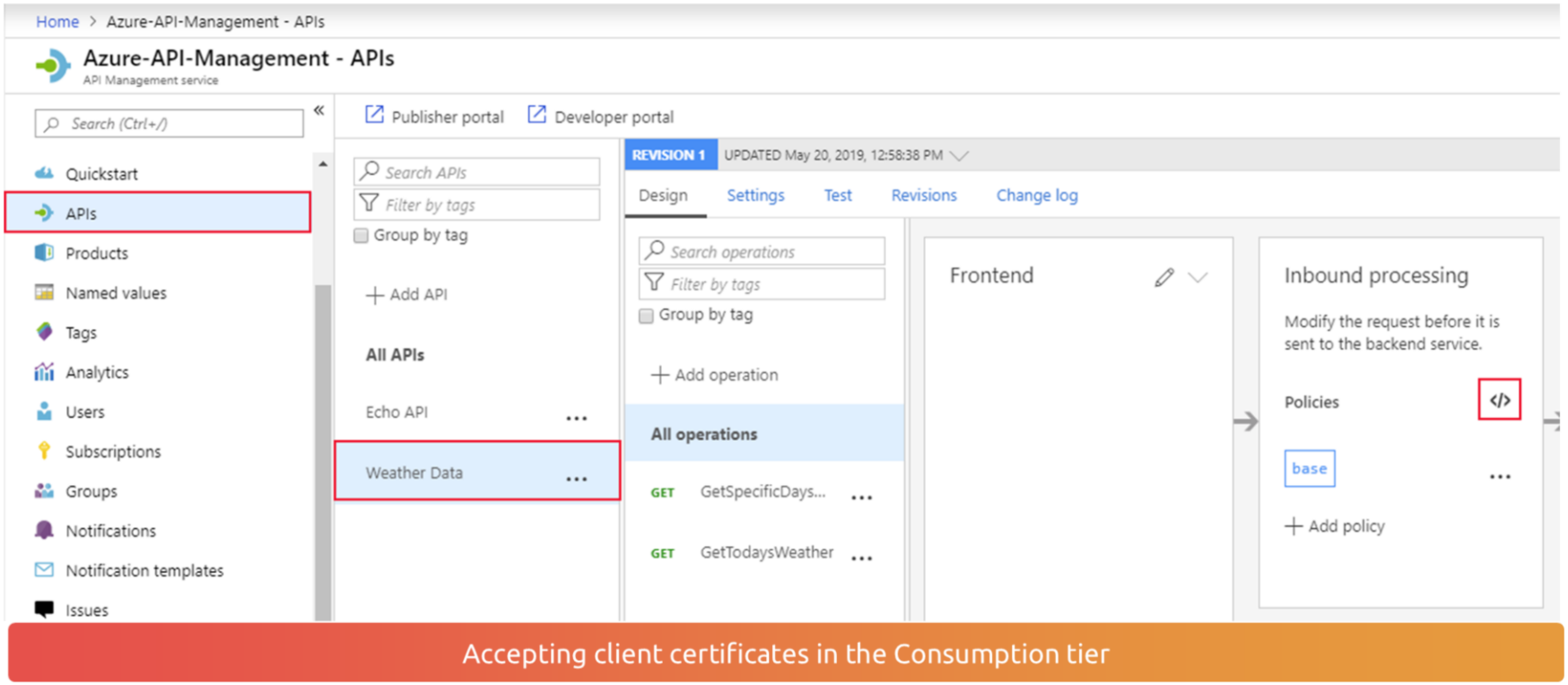
Property	Description
Certificate Authority (CA)	Only allow certificates signed by a particular CA
Thumbprint	Allow certificates containing a specified thumbprint
Subject	Only allow certificates with a specified subject
Expiration Date	Only allow certificates that have not expired

Certificates can be used to provide Transport Layer Security (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.

Certificate Verification Methods

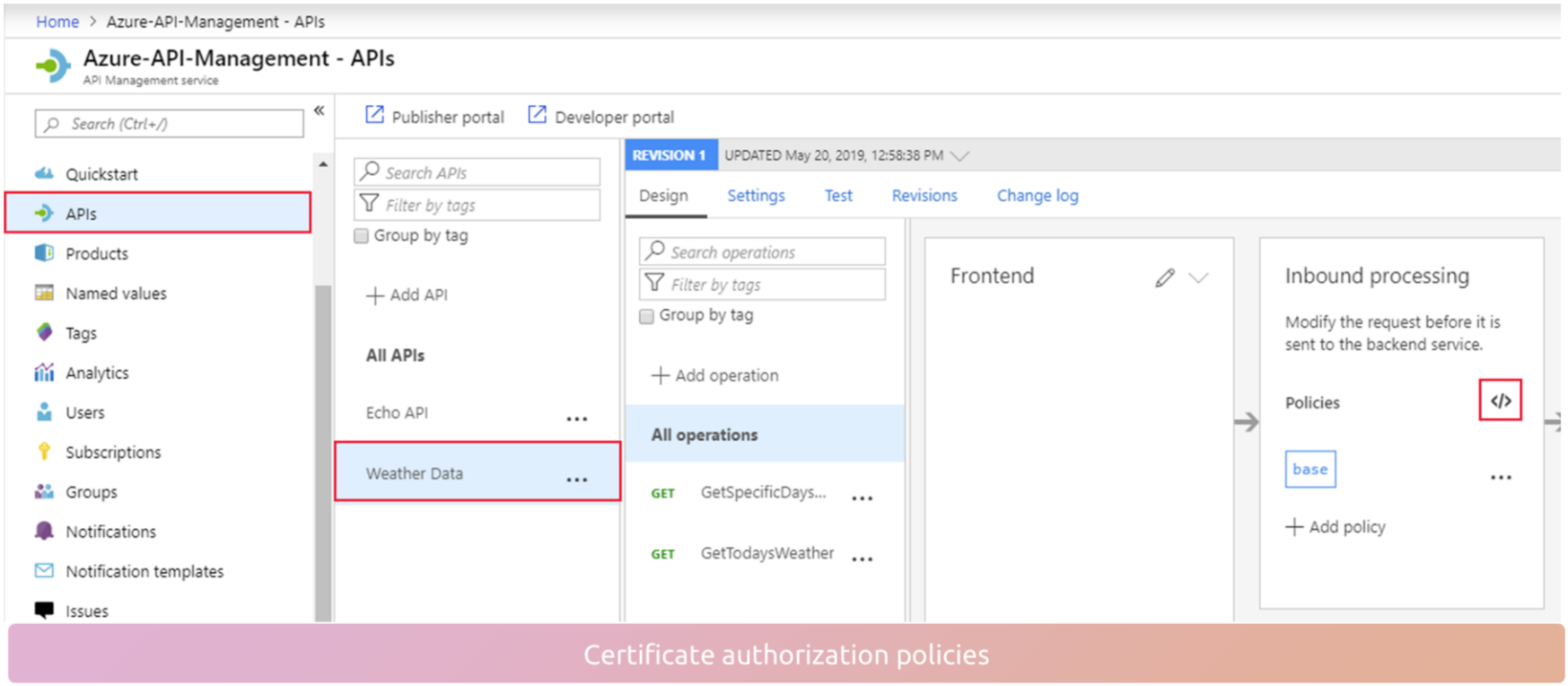
Verify the certificate issuer

Validate the certificate's origin by confirming it was issued by the partner

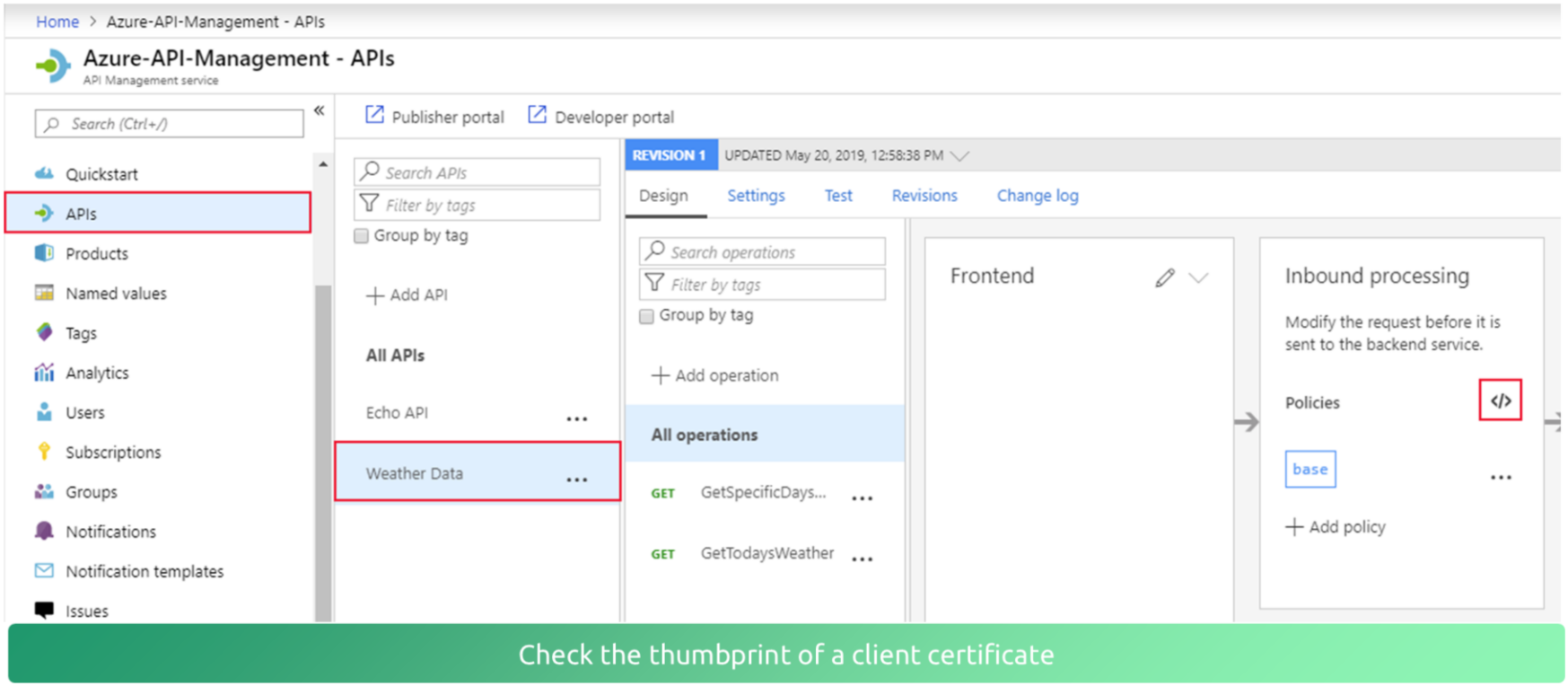


Accepting client certificates in the Consumption tier

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.

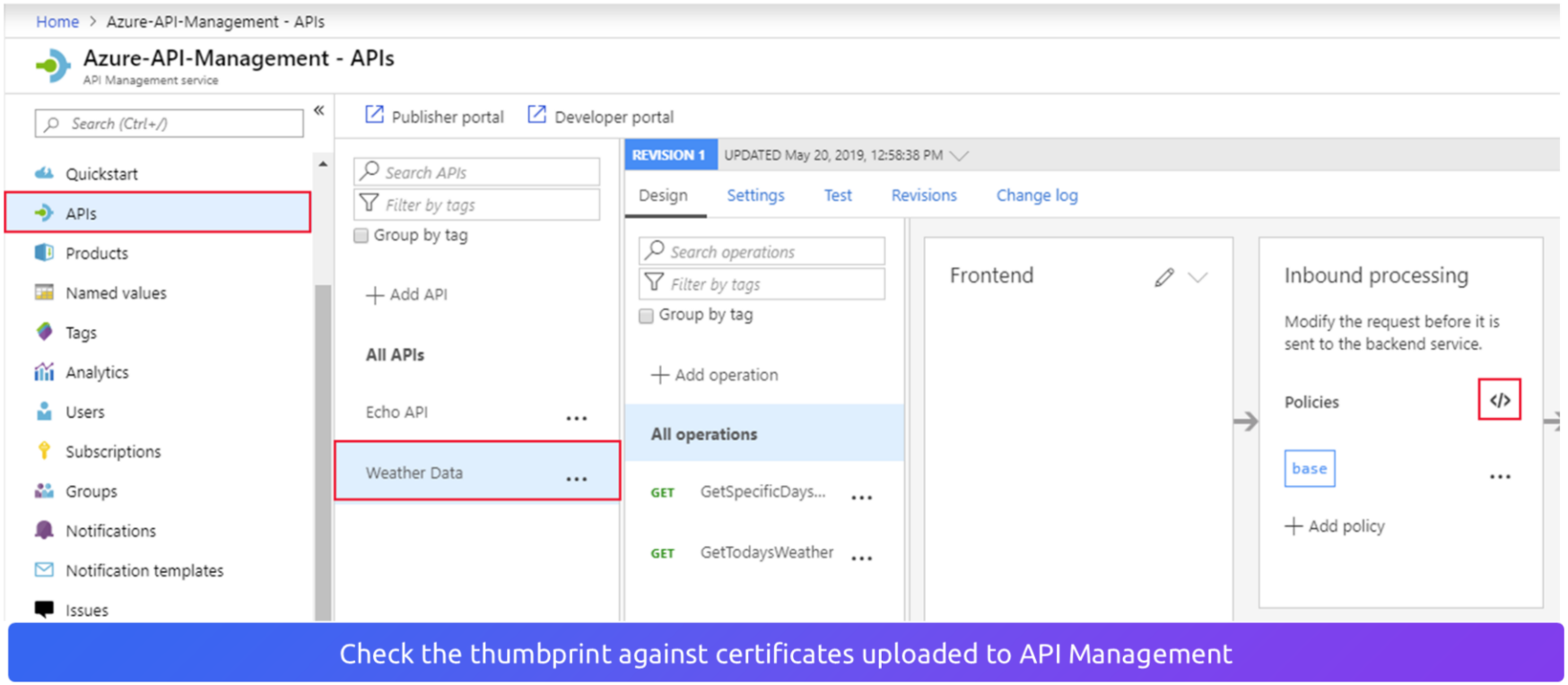


Certificate Authorization Policies
Create these policies in the inbound processing policy file within the API Management gateway (image shown on slide)

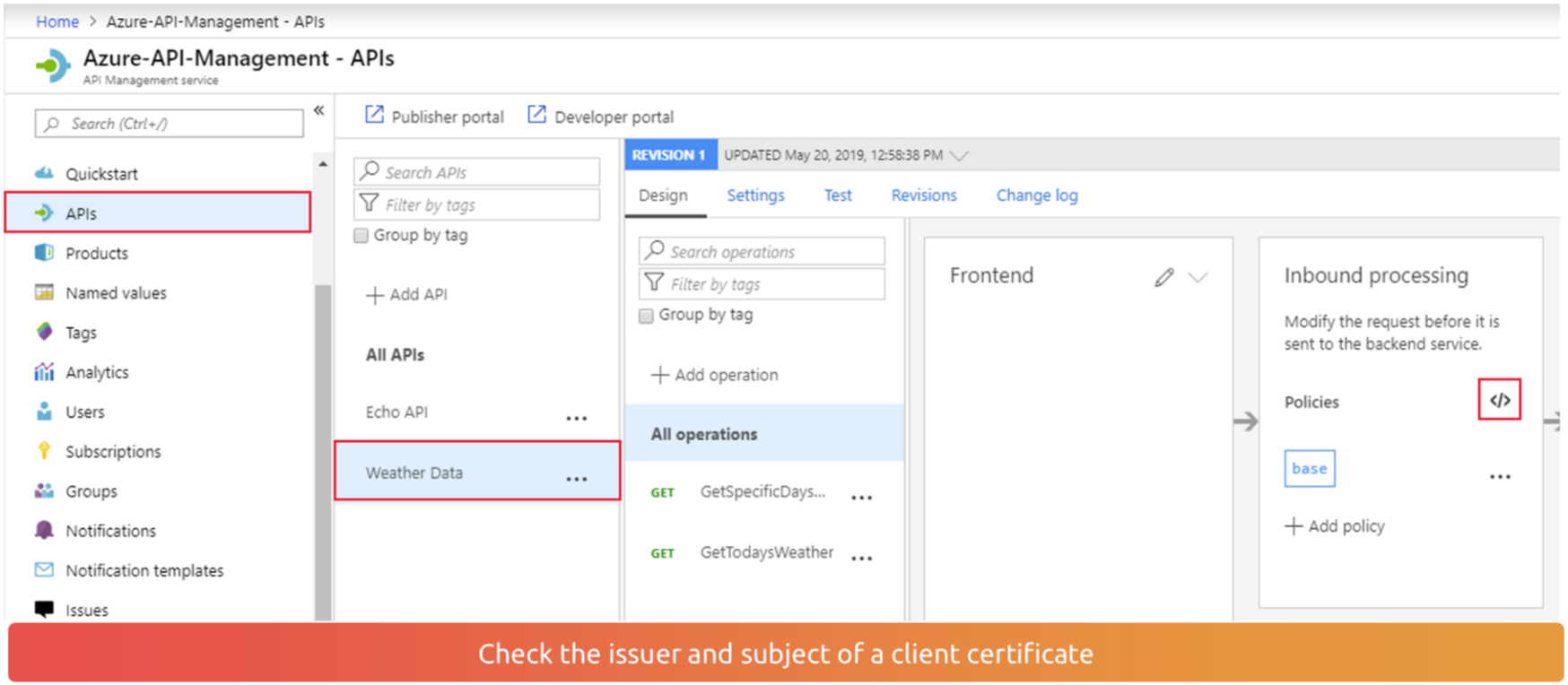


Check the thumbprint of a client certificate

Every client certificate includes a thumbprint, which is a hash, calculated from other certificate properties. 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 following example checks the thumbprint of the certificate passed in the request:



Check the thumbprint against certificates uploaded to API Management
Usually, each customer or partner company would pass a different certificate with a different thumbprint. 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 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:

Example 01

Example 01

```
<!--Check the thumbprint of a client certificate-->
<choose>
    <when condition="@(context.Request.Certificate == null ||
context.Request.Certificate.Thumbprint != "desired-thumbprint")" >
        <return-response>
            <set-status code="403" reason="Invalid client certificate" />
        </return-response>
    </when>
</choose>
```

Example 02

Example 02

```
<!--Check the issuer and subject of a client certificate-->
<choose>
    <when condition="@(context.Request.Certificate == null ||
context.Request.Certificate.Issuer != "trusted-issuer" ||
context.Request.Certificate.SubjectName.Name != "expected-subject-name")" >
        <return-response>
            <set-status code="403" reason="Invalid client certificate" />
        </return-response>
    </when>
</choose>
```

KodeKloud

Visit www.kodekloud.com to learn more.