Key Concepts

Let's take a look at some concepts and terminology that you need to know in order to follow the use cases.

WSO2 Cloud's user model

You can register to all WSO2 Public Cloud offerings (e.g., the Integration Cloud, the API Cloud) using a valid email address, and create multiple organizations under the same login. The system creates a unique tenant domain name for each organization you create. The user who creates an organization is assigned admin rights to it.

In WSO2 Public Cloud, users are authorized by their email addresses and are decoupled from organizations and tenant domains. This allows one user to be in multiple organizations (tenant domains) at the same time and log in to each organization using a single email address. If users are added to more than one organization, they are prompted to select the organization before logging in to its Cloud.

![WSO2 Cloud's user model](image)

Diagram: WSO2 Cloud's user model

API Cloud components

WSO2 API Cloud consists of the API Publisher, Store, Gateway, Key Manager, Traffic Manager, and Analytics. The following diagram illustrates the components and the message flow between components:
**API Publisher**: Enables complete API lifecycle governance—from creating to publishing, deprecating and retiring APIs. This component allows you to easily publish APIs, share documentation, and gather feedback on APIs, including their features, quality and usage.

**API Store**: An intuitive consumer-like online store where,
- API publishers can host and advertise their APIs.
- API consumers can discover, evaluate, subscribe to and use secured, protected, authenticated APIs.

**API Gateway**: Secures, protects, manages, and scales API calls. It is a simple API proxy that intercepts API requests and applies policies such as throttling and security. It also gathers API usage statistics.

**Key Manager**: Handles all security and access token-related operations. The Gateway connects with the Key Manager to check the validity of OAuth tokens, subscriptions, and API invocations. When a subscriber creates an application and generates an access token to the application using the API Store, the Store makes a call to the API Gateway, which in turn connects with the Key Manager to create an OAuth client and obtain an access token. Similarly, to validate a token, the API Gateway calls the Key Manager, which fetches and validates the token details from the database. The Key Manager also provides a token API to generate OAuth tokens that can be accessed via the Gateway. All tokens used for validation are based on the OAuth 2.0.0 protocol. Secure authorization of APIs is provided by the OAuth 2.0 standard for key management. The API Gateway supports API authentication with OAuth 2.0, and enables organizations to enforce rate limits and throttling policies.

**Traffic Manager**: Helps to regulate API traffic, make APIs and applications available to consumers at different service levels, and secure APIs against security attacks. The Traffic Manager features a dynamic throttling engine to process throttling policies in real-time, including rate limiting of API requests.

**Analytics**: The monitoring component provides a host of statistical graphs as well as an alerting mechanism on predetermined events.
Users and roles

The API Cloud offers distinct community roles that are applicable to most enterprises:

- **APICloudPublisher**: a publisher manages a set of APIs across the enterprise or business unit and controls the API lifecycle, subscriptions, and monetization aspects. The publisher is also interested in usage patterns for APIs and has access to all API statistics.
- **APICloudSubscriber**: a subscriber uses the API store to discover APIs, read the documentation, and forums, rate/comment on the APIs, subscribes to APIs, obtain access tokens, and invoke the APIs.

API Lifecycle

An API is the published interface, while the Service is the implementation running in the backend. APIs have their own lifecycles that are independent of the backend Services they rely on. This lifecycle is exposed in the API publisher Web interface and is managed by the API publisher role.

The following stages are available in the default API lifecycle:

- **CREATED**: API metadata is added to the API Store, but it is not deployed in the API gateway and therefore, is not visible to subscribers in the API Store.
- **PROTOTYPED**: the API is deployed and published in the API Store as a prototype. A prototype API is usually a mock implementation made public in order to get feedback about its usability. Users can invoke the API without a subscription.
- **PUBLISHED**: The API is visible in the API Store and available for subscription.
• **DEPRECATED:** When an API is deprecated, new subscriptions are disabled. But the API is still deployed in the Gateway and is available at runtime to existing subscribers. Existing subscribers can continue to use it as usual until the API is retired.

• **RETIRED:** The API is unpublished from the API gateway and deleted from the store.

• **BLOCKED:** Access to the API is temporarily blocked. Runtime calls are blocked and the API is not shown in the API Store anymore.

**Applications**

An application is a logical collection of APIs. An application is primarily used to decouple the consumer from the APIs. It allows you to:

• Generate and use a single key for multiple APIs
• Subscribe multiple times to a single API with different SLA levels

You subscribe to APIs through an application. Applications are available at different SLA levels and have application-level throttling tiers engaged in them. A throttling tier determines the maximum number of calls you can make to an API during a given period of time.

The API Cloud comes with a pre-created, default application. You can also create your own applications.

**Access tokens**

An **access token** is a simple string that is passed as an HTTP header of a request. For example, "Authorization: Bearer NtBQkXoKE1u0HIalfQ0DWfo61X4a." Access tokens authenticate API users and applications, and ensure better security (e.g., prevent **DoS attacks**). If a token that is passed with a request is invalid, the request is discarded in the first stage of processing. Access tokens work equally well for SOAP and REST calls.

There are two types of access tokens:

• **User access tokens**
• **Application access tokens**

**User access tokens**

Tokens to authenticate the final user of an API. User access tokens allow you to invoke an API even from a third-party application like a mobile app. You generate/renew a user access token by calling the Login API through a REST client. For more information, see **Token API**.

**Application access tokens**

Tokens to authenticate an application, which is a logical collection of APIs. They allow you to access all APIs associated with an application using a single token, and also subscribe multiple times to a single API with different SLA levels. Application access tokens leverage OAuth 2.0 to provide simple key management.

The steps below describe how to generate/renew application access tokens:

1. Log in to the API Store.
2. Click the **Applications** menu and then click the application to open it. For example, under the **Production Keys** tab, scroll down and click **Regenerate**.

3. **Throttling**

Throttling allows you to limit the number of successful hits to an API during a given period of time, typically in cases such as the following:

- To protect your APIs from common types of security attacks such as denial of service (DoS).
- To regulate traffic according to infrastructure availability.
- To make an API, application or a resource available to a consumer at different levels of service, usually for monetization purposes.

See **Working with Throttling in the WSO2 API Manager documentation** for detailed information about throttling, throttling limits, policies etc.

**API visibility and subscription**

Visibility settings prevent certain user roles from viewing and modifying APIs created by another user role.
- **Public**: the API is visible to all users (registered and anonymous), and can be advertised in multiple stores (central and non-WSO2 stores).
- **Visible to my domain**: the API is visible to all users who are registered to the API's tenant domain.
- **Restricted by Roles**: The API is visible to its tenant domain and only to the user roles that you specify. Accepted roles are admin, publisher, subscriber, and appclouduser.

Given below is how visibility levels work for users in different roles:

- **API creator** and **publisher** roles can see all APIs in their tenant store even if you restrict access to them. This is because those roles have permission to view and edit all APIs in the API Publisher, and therefore, does not have to be restricted in the Store.
- Anonymous users can only see APIs that have visibility as **Public**.
- Registered users can see
  - public APIs of all tenant domains
  - all APIs in the registered user's tenant domain as long as the API is not restricted to a role that the user is assigned to

An API's visibility is directly related to the API's **subscription availability** because you cannot subscribe to something you don't see in the Store. The diagram below depicts this relationship:
An API is made up of one or more resources. Each resource handles a particular type of request and is analogous to a method (function) in a larger API.

### API Definition

<table>
<thead>
<tr>
<th>URL Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/{context}/v{version}/</td>
<td>Url Pattern E.g.: path/to/resource</td>
</tr>
</tbody>
</table>

- GET
- POST
- PUT
- DELETE
- PATCH
- HEAD

#### Attribute Name | Description
--- | ---
OAuth Scopes | OAuth scopes

URL Pattern can be one of the following types:

- As a url-mapping. E.g., /{state}/town/*
- As a uri-template. E.g., /{state}/{town}

The terms url-mapping and uri-template come from Synapse configuration language. When an API is published in the API Publisher, a corresponding XML definition is created in the API Gateway. This XML file has a dedicated section for defining resources. See examples below:

```xml
<resource methods="POST GET" url-mapping="/state/town/*">
  <resource methods="POST GET" uri-template="/texas/{houston}">\n    url-mapping performs a one-to-one mapping with the request URL, whereas the uri-template performs a pattern matching.

Parametrizing the URL allows the API Manager to map the incoming requests to the defined resource templates based on the message content and request URI. Once a uri-template is matched, the parameters in the template are populated appropriately. As per the above example, a request made to `http://gatewa_host:gateway_port/api/v1/texas/houston` sets the value of state to texas and the value of town to houston. You can use these parameters within the synapse configuration for various purposes and gain access to these property values through the `uri.var.province` and `uri.var.district` properties. For more information on how to use these properties, see Introduction to REST API and the HTTP Endpoint of the WSO2 ESB documentation.


HTTP Verb | The HTTP methods that specify the desired action to be performed on the resource. These methods can be GET, POST, PUT, DELETE or OPTIONS. Multiple methods can be selected.
--- | ---
Auth Type | The authentication type of each HTTP method of the resource. You can give one of the following:
--- | ---
- **None**: No authentication is applied and the API Gateway skips the authentication process
Authentication is done by the application. The resource accepts application access tokens.

**Application User:** Authentication is done by the application user. The resource accepts user access tokens.

**Application and Application User:** Both application and application user level authentication is applied. Note that if you select this option in the UI, it appears as Any in the API Manager’s internal data storage and data representation, and Any will appear in the response messages as well.

**Note** that for the resources that have HTTP verbs (GET, POST etc.) requiring authentication (i.e., Auth Type is not NONE), set None as the Auth type of OPTIONS. This is to support CORS (Cross Origin Resource Sharing) between the API Store and Gateway.

The auth type is cached in the API Manager for better performance. If you change the auth type through the UI, it takes about 15 minutes to refresh the cache. During that time, the server returns the old auth type from the cache.

Once a request is accepted by a resource, it will be mediated through an in-sequence. Any response from the backend is handled through the out-sequence. Fault sequences are used to mediate errors that might occur in either sequence. The default in-sequence, out-sequence, and fault sequences are generated when the API is published.

**Swagger API Console**

WSO2 API Store provides an integrated Swagger UI, which is part of the Swagger project. You can use it for interactive API documentation and to invoke APIs.

**Swagger** is a 100% open source, standard, language-agnostic specification and a complete framework for describing, producing, consuming, and visualizing RESTful APIs, without the need of a proxy or third-party services. Swagger allows consumers to understand the capabilities of a remote service without accessing its source code and interact with the service with a minimal amount of implementation logic. Swagger helps describe services in the same way that interfaces describe lower-level programming code.

The Swagger UI is a dependency-free collection of HTML, JavaScript, and CSS that dynamically generate documentation from a Swagger-compliant API. Swagger-compliant APIs give you interactive documentation, client SDK generation, and more discoverability. The Swagger UI has JSON code and its UI facilitates easier code indentation, keyword highlighting and shows syntax errors on the fly. You can add resource parameters, summaries and descriptions to your APIs using the Swagger UI.

The API Publisher has integrated Swagger to facilitate simple, interactive API documentation and invocation. The documentation is given in a Swagger API definition, which is the JSON representation of the API that is created using the information provided at the time the API is created. The Swagger JSON files are saved in the registry. You can edit the API definition using the Edit Swagger Definition button in the API Publisher.

Also, see the [Swagger 2.0 specification](#).

**OAuth scopes**

Scopes enable fine-grained access control to API resources based on user roles. You define scopes to an API's resource. When a user invokes the API, his/her OAuth 2 bearer token cannot grant access to any API resource beyond its associated scopes.

You can apply scopes to an API resource at the time the API is created or modified. In the API Publisher, click the API -> Add menu (to add a new API) or the Edit link next to an existing API. Then, navigate to the Manage tab and scroll down to see the Add Scopes button. A screen such as the following appears:
Define Scope

| Scope Key | A unique key for identifying the scope. Typically, it is prefixed by part of the API's name for uniqueness but is not necessarily reader-friendly. |
| Scope Name | A human-readable name for the scope. It typically says what the scope does. |
| Roles | The user role(s) that are allowed to obtain a token against this scope. E.g., manager, employee. |

To illustrate the functionality of scopes, assume you have the following scopes attached to resources of an API:

Assume that users named Tom and John are assigned the employee role and both the employee and manager roles respectively.

Tom requests a token through the Token API as `grant_type=password&username=tom&password=xxxx&scope=news_read news_write`. However, as Tom is not in the manager role, he will only be granted a token bearing the `news_read` scope. The response from the Token API will be similar to the following:

```json
"scope":"news_read",
"token_type":"bearer",
"expires_in":3299,
"refresh_token":"8579facb65d1d3eba74a395a2e78dd6",
"access_token":"eb51eff0b4d85cda1ebd312c5b6a3b8"
```
Next, John requests a token as `grant_type=password&username=john&password=john123&scope=news_read news_write`. As John has both roles assigned, the token will bear both the requested scopes and the response will be similar to the following:

```
"scope":"news_read news_write",
"token_type":"bearer",
"expires_in":3299,
"refresh_token":"4ca244fb321bd555bd3d555df39315",
"access_token":"42a377a0101877d1d9e29c5f30857e"
```

This means that Tom can only access the GET operation of the API while John can access both as he is assigned to both the employee and manager roles. If Tom tries to access the POST operation, there will be an HTTP 403 Forbidden error as follows:

```
<ams:fault xmlns:ams="http://wso2.org/apimanager/security">
  <ams:code>900910</ams:code>
  <ams:message>The access token does not allow you to access the requested resource</ams:message>
  <ams:description>Access failure for API: /orgnews, version: 1.0.0 with key: eb51eff0b4d85cda1eb1d312c5b6a3b8</ams:description>
</ams:fault>
```

Endpoints

An endpoint is a specific destination for a message such as an address, WSDL, a failover group, a load-balance group etc.

WSO2 API Manager has support for a range of different endpoint types, allowing the API Gateway to connect with advanced types of backends. It supports **HTTP endpoints**, **URL endpoints**, **WSDL endpoints**, **Failover endpoints**, and **Load-balanced endpoints**. For more information about endpoints and how to add, edit or delete them, see the **WSO2 ESB documentation**.

Note the following:

- You can expose both REST and SOAP services to consumers through APIs.
- You cannot call backend services secured with OAuth through APIs created in the API Publisher. At the moment, you can call only services secured with username/password.
- The system reads gateway endpoints from the `api-manager.xml` file. When there are multiple gateway environments defined, it picks the gateway endpoint of the production environment. You can define both HTTP and HTTPS gateway endpoints as follows:

  ```
  ```

- If both types of endpoints are defined, the HTTPS endpoint will be picked as the server endpoint.

Sequences

The API Cloud has a default mediation flow that is executed in each API invocation. There are 3 default sequences engaged as `in`, `out` and `fault`.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
</table>
In-sequence is the first place that will be mediated through, once a request is dispatched to a resource of an API. At the end of the in-sequence the request can be forwarded to a back-end application for further processing.

Any responses coming from the back-end system are mediated through the out-sequence of the resource of the API.

Fault sequence is there to handle any errors that may occur while mediating a message through a resource.

When the sequence or the proxy service encounters an error during mediation or while forwarding a message, the message that triggered the error is delegated to the specified fault sequence. Using the available mediators it is possible to log the erroneous message, forward it to a special error-tracking service, and send a SOAP fault back to the client indicating the error. We need to configure the fault sequence with the correct error handling instead of simply dropping messages. For more information, see Error Handling.

### Mediators

Sometimes, the backend implementation of a Web service does not exactly match your desired API design. In that case, you do various message transformations and orchestrate multiple backend services to achieve the design you want.

The API Cloud comes with a powerful mediation engine that can transform and orchestrate API calls on the fly. It is built on the WSO2 ESB and supports a variety of mediators that you can use as building blocks for your sequences. Given below is the list of mediators supported in the API Cloud:

<table>
<thead>
<tr>
<th>Category</th>
<th>Mediator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>Call</td>
<td>Invokes a service in non-blocking, synchronous manner.</td>
</tr>
<tr>
<td></td>
<td>Sequence</td>
<td>Inserts a reference to a sequence.</td>
</tr>
<tr>
<td></td>
<td>Drop</td>
<td>Drops a message.</td>
</tr>
<tr>
<td></td>
<td>Enrich</td>
<td>Enriches a message.</td>
</tr>
<tr>
<td></td>
<td>Property</td>
<td>Sets or removes properties associated with the message.</td>
</tr>
<tr>
<td></td>
<td>Log</td>
<td>Logs a message.</td>
</tr>
<tr>
<td>Filter</td>
<td>Filter</td>
<td>Filters a message using XPath, if-else type of logic.</td>
</tr>
<tr>
<td></td>
<td>Validate</td>
<td>Validates XML messages against a specified schema.</td>
</tr>
<tr>
<td></td>
<td>Switch</td>
<td>Filters messages using XPath and switch logic.</td>
</tr>
<tr>
<td></td>
<td>ConditionalRout er</td>
<td>Implements complex routing rules (header-based routing, content-based routing, and other rules).</td>
</tr>
<tr>
<td>Transform</td>
<td>XSLT</td>
<td>Performs XSLT transformations on the XML payload.</td>
</tr>
<tr>
<td></td>
<td>FastXSLT</td>
<td>Performs XSLT transformations on the message stream.</td>
</tr>
<tr>
<td></td>
<td>URLRewrite</td>
<td>Modifies and rewrites URLs or URL fragments.</td>
</tr>
<tr>
<td>XQuery</td>
<td>Performs XQuery transformation.</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------</td>
<td></td>
</tr>
<tr>
<td>Header</td>
<td>Sets or removes SOAP headers.</td>
<td></td>
</tr>
<tr>
<td>Fault (also</td>
<td>Creates SOAP Faults.</td>
<td></td>
</tr>
<tr>
<td>called</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Makefault)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PayloadFactory</td>
<td>Transforms or replaces message</td>
<td></td>
</tr>
<tr>
<td></td>
<td>content in between the client</td>
<td></td>
</tr>
<tr>
<td></td>
<td>and the backend server.</td>
<td></td>
</tr>
<tr>
<td>Advanced</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cache</td>
<td>Evaluates messages based on</td>
<td></td>
</tr>
<tr>
<td></td>
<td>whether the same message came</td>
<td></td>
</tr>
<tr>
<td></td>
<td>to the ESB.</td>
<td></td>
</tr>
<tr>
<td>ForEach</td>
<td>Splits a message into a number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>of different messages by finding</td>
<td></td>
</tr>
<tr>
<td></td>
<td>matching elements in an XPath</td>
<td></td>
</tr>
<tr>
<td></td>
<td>expression of the original</td>
<td></td>
</tr>
<tr>
<td></td>
<td>message.</td>
<td></td>
</tr>
<tr>
<td>Clone</td>
<td>Clones a message.</td>
<td></td>
</tr>
<tr>
<td>Iterate</td>
<td>Splits a message.</td>
<td></td>
</tr>
<tr>
<td>Aggregate</td>
<td>Combines a message.</td>
<td></td>
</tr>
<tr>
<td>Callout</td>
<td>Blocks Web service calls.</td>
<td></td>
</tr>
<tr>
<td>Transaction</td>
<td>Executes a set of mediators</td>
<td></td>
</tr>
<tr>
<td></td>
<td>transactionally.</td>
<td></td>
</tr>
<tr>
<td>DBReport</td>
<td>Writes data to the database.</td>
<td></td>
</tr>
<tr>
<td>DBLookup</td>
<td>Retrieves information from the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>database.</td>
<td></td>
</tr>
<tr>
<td>Entitlement</td>
<td>Evaluates user actions against</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a XACML policy.</td>
<td></td>
</tr>
<tr>
<td>Script</td>
<td>Executes a mediator written in</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Scripting language.</td>
<td></td>
</tr>
</tbody>
</table>

**Also, see the list of WSO2 ESB Mediators.**

You can extend the API Gateway's default mediation flow to do custom mediation by providing an extension as a Synapse mediation sequence. You design all sequences using a tool like WSO2 Developer Studio and then store the sequence in the Gateway's registry. See [Change the Default Mediation Flow of API Requests](#).

**Response caching**

Response caching stores the response messages per each API in a cache. It improves performance because the backend server does not have to process the same data for a request multiple times. To offset the risk of stale data in the cache, you can set a time period after which the cache is cleared periodically.

Response caching is automatically disabled in the Cloud. You can enable it when creating a new API or editing an existing API. Go to the API's Manage tab in the API Publisher, and select Enable from the Response Caching drop-down list and give a time-out period.
Gateway caching

When an API call hits the API Gateway, the Gateway carries out security checks to verify if the token is valid. During these verifications, the API Gateway extracts parameters such as access token, API, and API version that are passed to it. Since all traffic to APIs goes through the API Gateway, this verification process can become an overhead if the API name, version, and other validation information are not cached. Therefore, caching is enabled at the Gateway by default.

When caching is enabled and a request hits the Gateway, it first populates the cached entry for a given token. If a cache entry does not exist in the cache, it calls the Key Manager server. This process is carried out using Web Service calls. Once the Key Manager server returns the validation information, it gets stored in the Gateway. Because the API Gateway issues a Web service call to the key manager server only if it does not have a cache entry, the number of Web service calls to the key manager server is reduced.

Admin Portal

WSO2 API Cloud provides you an admin portal which allows to perform administration on the Applications and APIs you have published. The left navigation bar of the admin portal directs to different administration tasks as below.
**TASKS**

TASKS are refered to the workflows in WSO2 API Manager that you can configure constraints on top of actions performed. There are four main workflows as User SignUp workflow, Application Creation Workflow, Subscription Creation workflow and Application Registration workflow. See Managing Workflow Extensions for more information.

**SETTINGS**

By navigating to SETTINGS tab, you are facilitated to create new upload newly created tenant themes to on top of publisher, store and admin portal user interfaces.

**THROTTLING**
### POLICIES

<table>
<thead>
<tr>
<th>POLICIES</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>THROTTLING POLICIES</td>
<td>Contains the settings for the throttling configurations of API Manager. Under that,</td>
</tr>
<tr>
<td>ADVANCED THROTTLING</td>
<td>Create and Update Advanced Throttling Policies. See <a href="#">Working with Advanced Throttling Policies</a> for more information.</td>
</tr>
<tr>
<td>APPLICATION TIERS</td>
<td>Create and Update Application level throttling tiers. See <a href="#">Working with Application Throttling Policies</a> for more information.</td>
</tr>
<tr>
<td>SUBSCRIPTION TIERS</td>
<td>Create and Update API Subscription level throttling tiers. See <a href="#">Working with Subscription Throttling Policies</a> for more information.</td>
</tr>
<tr>
<td>BLACK LIST</td>
<td>Blacklist requests by APIs, Applications, Users and IP Addresses. See <a href="#">Working with Blacklist Policies</a> for more information.</td>
</tr>
</tbody>
</table>

### LOG ANALYZER

See [Analyzing Logs with the Log Analyzer](#)

### ANALYTICS

<table>
<thead>
<tr>
<th>ANALYTICS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API AVAILABILITY</td>
<td>Provides statistics of API Availability</td>
</tr>
<tr>
<td>MANAGE ALERTS</td>
<td>Provides configuring alerts for different behaviors of APIs, requests, response times etc. See <a href="#">Managing Alerts with Real-time Analytics</a></td>
</tr>
</tbody>
</table>

Other than that, you can also use the admin portal of API Cloud to [Customize Cloud Domains](#).