ORACLE

REST API for OCI

Scenarios to use REST API for Cloud Services Interaction

November 24, 2022 | Version 1.01

PURPOSE STATEMENT

This document provides an overview of features and enhancements included in release <Release>. It is intended solely to help you assess the business benefits of upgrading to <Release> and to plan your I.T. projects.

DISCLAIMER

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle software license and service agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement, nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

This document is for informational purposes only and is intended solely to assist you in planning for the implementation and upgrade of the product features described. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle. Due to the nature of the product architecture, it may not be possible to safely include all features described in this document without risking significant destabilization of the code.

DISCLAIMERS FOR PRE-RELEASE, PRE-GA PRODUCTS

The **revenue recognition disclaimer** on this page is required for any white paper that addresses future functionality or for products that are not yet generally available (GA). If you are unsure whether your statement of direction needs the disclaimer, read the revenue recognition policy. If you have further questions about your content and the disclaimer requirements, e-mail REVREC_US@oracle.com.

The **testing disclaimer** in the copyright section on the last page (highlighted in yellow) is provided by the FCC for hardware products. It must appear in the copyright section for all pre-release, pre-GA hardware products. Be sure to remove the yellow highlighting before publishing. When the product becomes GA, update your collateral by removing the disclaimer from the copyright section. If your product is already GA or if you are writing about a software product, delete the disclaimer from the copyright section.

Important: If your product is not GA, then you cannot include any regulatory compliance information in the statement of direction. Regulatory compliance information may be included for GA products only if you have completed all required safety and emissions testing, and you have received the certificates issued by the testing organization

TABLE OF CONTENTS

LIST OF IMAGES

LIST OF TABLES

OVERVIEW

A REST API (also known as RESTful API) is an application programming interface (API or REST API for short) that is based in constraints of REST architectural style and allows for interaction with RESTful web and most commonly cloud services.

This technology that nowadays is a standard for the cloud service consumption was created by computer scientist Roy Fielding, and by now is the most common representational state transfer of information used in cloud environments.

Oracle Cloud Infrastructure provides are several APIs to interact with its services, and even we can said that use of APIs is a typical way of interaction for cloud resource creation, deletion and in general interaction; just remember with REST APIs we need to use HTTPS (requests and responses), so we should be authenticated within our OCI console before use them, and within this document we will describe all those considerations in order to know the key elements using RESTs APIs for OCI.

REST API

To understand how interact with cloud services with APIs, we must start first with definition of REST. This concept basically is a set of architectural constraints, not a protocol or a standard. Many developers can implement REST in a variety of ways.

When a request is made through a RESTful API, it transfers information, where the state of the resource will be affected at the endpoint.

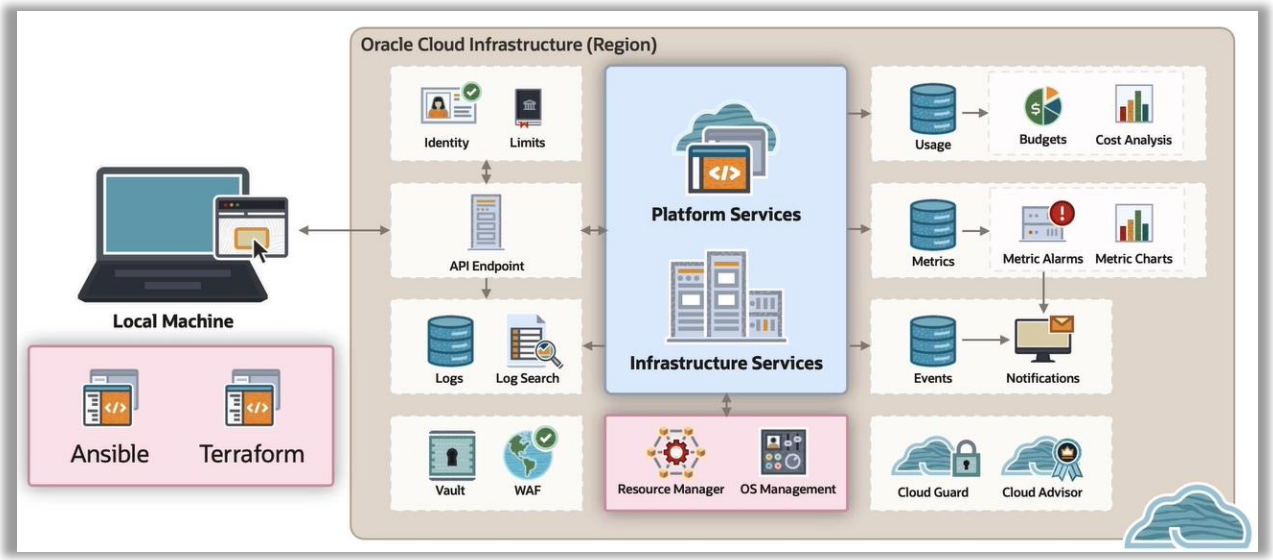
The expected state, or expected information is delivered to the requester in well-known formats, such as JSON (Javascript Object Notation), HTML, XLT, Python, PHP, plain text, even CURL format. However, the most common format is JSON due to its simplicity and understanding by both humans and machines.
Something else to keep in mind: Headers and parameters are also important in the HTTP methods of a RESTful API HTTP request, as they contain important identifier information as to the request's metadata, authorization, uniform resource identifier (URI), caching, cookies, and more. There are request headers and response headers, each with their own HTTP connection information and status codes.

REST API can be executed follow an architecture server, and to use REST API, you should consider the following points:

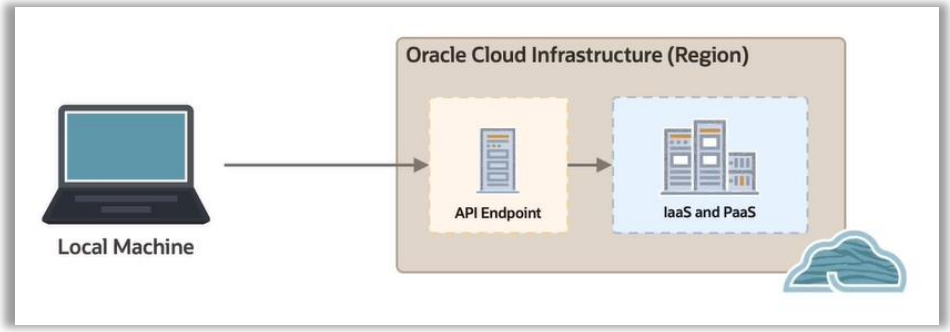
- Based on client-server architecture, for interaction with servers and resources, we should use requests managed through HTTP.
- Cacheable data that streamlines client-server interactions.
- A client-server architecture made up of clients, servers, and resources, with requests managed through HTTP.
- Under the client-client-server architecture, many of the requests made through a client are not stored, since each request made is independent and not connected.
- Does exist many clients in order perform consumption of cloud resources through REST API.

INTERACTING WITH OCI: CONSOLE, CLI, SDK AND API

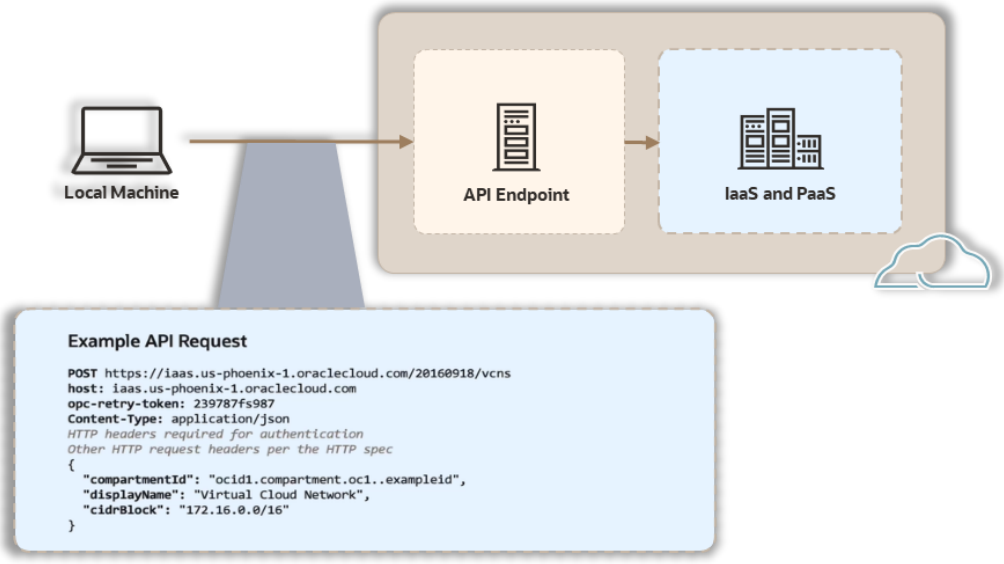
In this document, we'll touch on some of the main ways of interacting with OCI, including the Console, CLI, SDK, and REST API. Starting from the top, our goal is to connect to OCI from our local machine and manage services and infrastructure in OCI, and we need to know the tool we're likely familiar with for that is the Console. But it's important that we dig a little deeper.



One of the starting points is to understand is how to use the core services, to using the data that services emit, to securing and governing tenancies, and start with understanding that OCI is fundamentally **API-based**, and of various methods of generating those API calls, including the cloud console, the command line interface, and the software development kits.



Under the hood every interaction with OCI is through a REST API endpoint. So for example, whenever we want to create a VCN, Instance, Notification service, Kubernetes cluster, Object Storage, etc. to interact, we have to hit the endpoint with a request that looks something like this.

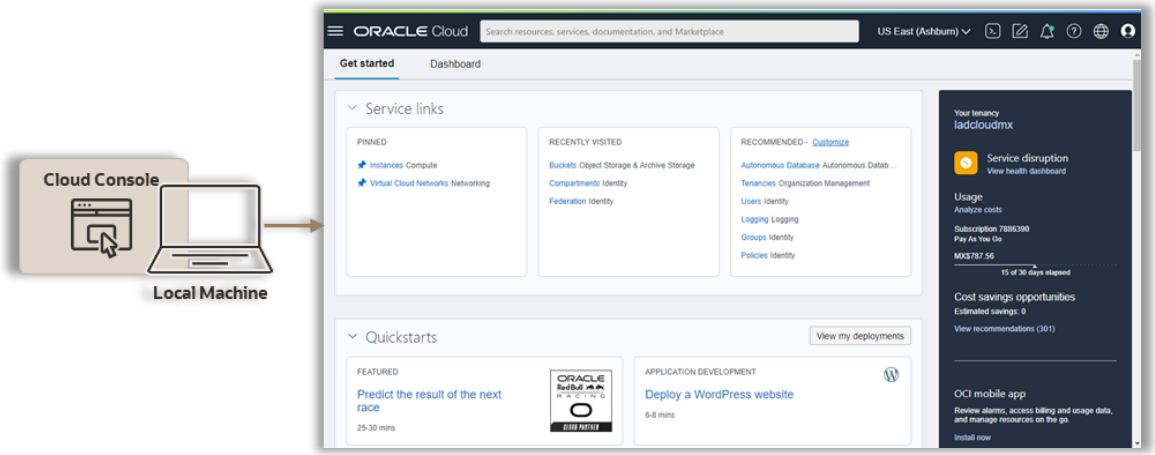


Obviously working with the API directly is rather unwieldy. Writing these comes up in specific use cases, like in application integration. But often, it's easier to work with high level tools that generate these calls for you.

The important thing is to be loosely familiar with the API for two main reasons:

- **First, permissions are granted at approximately the API level. So, to implement the principle of least privilege**, we need to understand which API calls we grant to each actor.
- **Second, when things go wrong, figuring out exactly which API call failed is often a great first step in debugging**. But for the most part, we'll generate these calls through other tools.

The first and most familiar way is with the Cloud Console. This is the web application where you can click around to manage your environments. It's the easiest tool to use and is great for prototyping or for small environments.

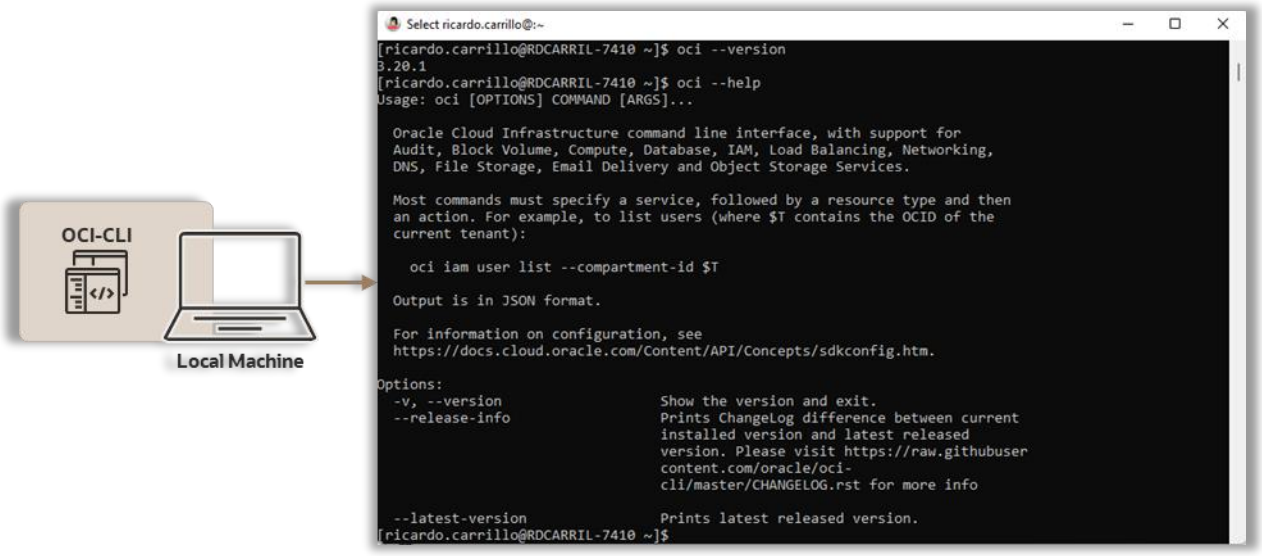


The main benefits of using **Cloud Console**:

- It is the default console.
- Because to be a web application, a cloud admin can interact with all the OCI services have deployed.
- It is mainly used as access for the administration of small environments.

The cloud console, for complex tasks, it comes to more complicated actions, for example, the creation of 100 compute instances, it is not effective as these activities become complicated and error prone.

By other way, exist another way, it is a **simple text-based utility** that can be installed on most operating system's terminals, including Mac OS, Windows, and Linux and Unix-like systems.

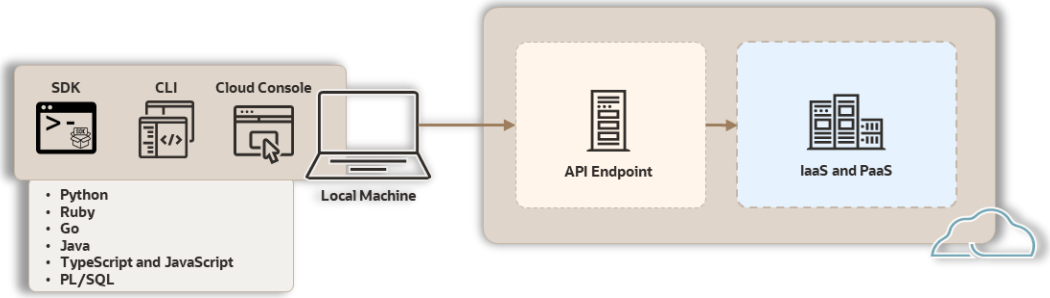


This way we can leverage shell scripts to manage infrastructure. If we want to provision 100 compute instances, we could write a quick Bash script that loops 100 times or for example, if we wanted to change the shape of every compute instance in a compartment, we could write a script that queries for all of the compute instances and then iterates over them, changing their shape.

The main benefits of using **OCI-CLI** are:

- Automation of repetitive tasks.
- Can be installed on various operating systems (Mac OS, Windows, and Linux/Unix-like systems).
- Shell scripts can be developed for infrastructure management.
- Use for complex cases, e.g. the creation of 100 compute instances, a shell script could be developed and repeated 100 times for the creation of these resources.

But just like the Cloud Console, and OCI-CLI, when use cases get even much more complex, the CLI can also become unwieldy. Suppose we want to automatically provision a new server and decommission an old one if certain health checks fail or suppose we want a pilot light DR environment that automatically replicates prod but scaled down. For these, the appropriate tool would be **the SDK**.



The Software Development Kit, abbreviated SDK, consists of libraries for various programming languages that will generate API calls for you. This way we can leverage the full generality of languages like Python, Ruby, or Go, just to name a few.

The main benefits of using SDK:

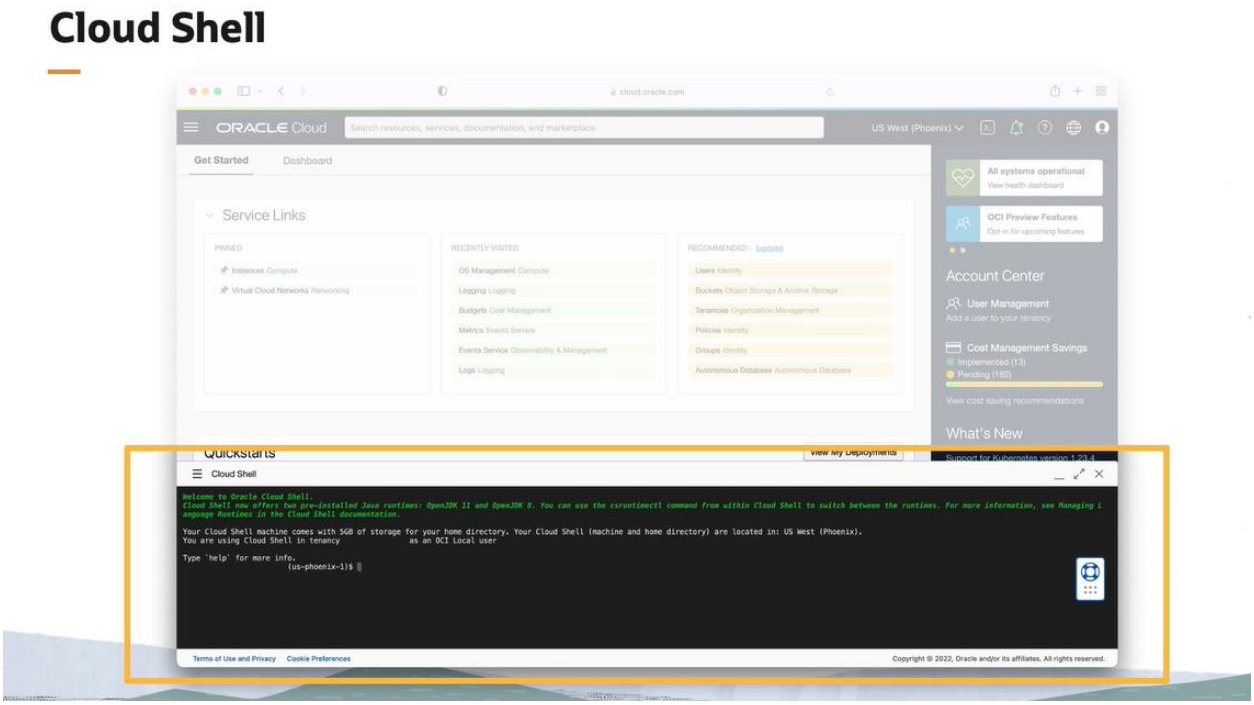
- Being able to choose a programming language of preference so that through its libraries API calls can be generated.
- Unlike OCI-CLI, SDKs are used to perform highly complex administration tasks.
- SDKs can be used for serverless functions that are executed based on customized API calls.

As opposed to the CLI, the SDK becomes the most natural tool when complex logic is involved and additionally, the SDK can be used for serverless functions that are executed based on custom API calls or infrastructure made at events.

To summarize, we have that OCI is fundamentally REST API-based, but we generally use higher level tools to generate those calls:

- The Console is best for simple use cases, experimenting, and prototyping.
- The CLI is best for simple but repetitive tasks and automation.
- And the SDK is best for complex automation.

The last thing I'll mention is the Cloud Shell. The Cloud Shell is a web application within the Cloud Console that streamlines CLI usage. It works by automatically provisioning a small compute instance with preconfigured security tokens and connects to it through your browser. Then the CLI client itself runs from the compute instance so it can attach its security tokens to your API requests.

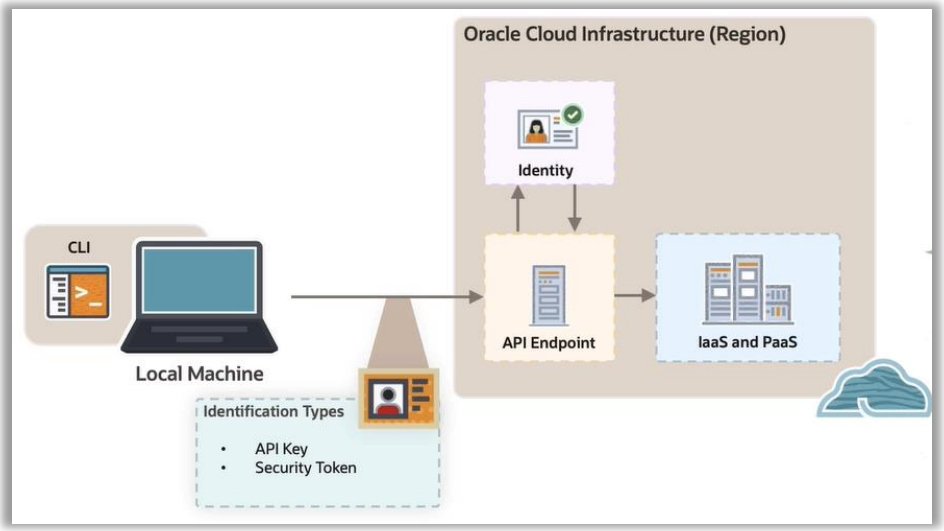


For the most part, this is where we'll be doing all our CLI work. And that's it. So we have that the two basic ways of authenticating from the CLI are API keys and security tokens. Machines within OCI can use their metadata for instance principal and resource principal authentication.

OCI CLI Authentication

Before you can do anything in the OCI-CLI, you'll have to set up authentication that lets **OCI know who you are**. This section we'll have seen different methods or ways in how a OCI user can authenticate to OCI. Recall that all interaction with OCI is through a REST endpoint and that all the CLI does is generate and sends requests to that endpoint.

But before the endpoint passes that request onto any other service, it asks the identity and access management service to verify that **you are who you say you are**, also known as **authentication**, and that **you are allowed** to perform the action that you're requesting, also known as **authorization**.



To facilitate this, your CLI client attaches some sort of identification to its **API requests**. Just to get a little more concrete, in the API call picture below, for creating a VCN, that just means adding some headers right here that function kind of like a **username** and **password**. But again, knowing the details of the API isn't that important.

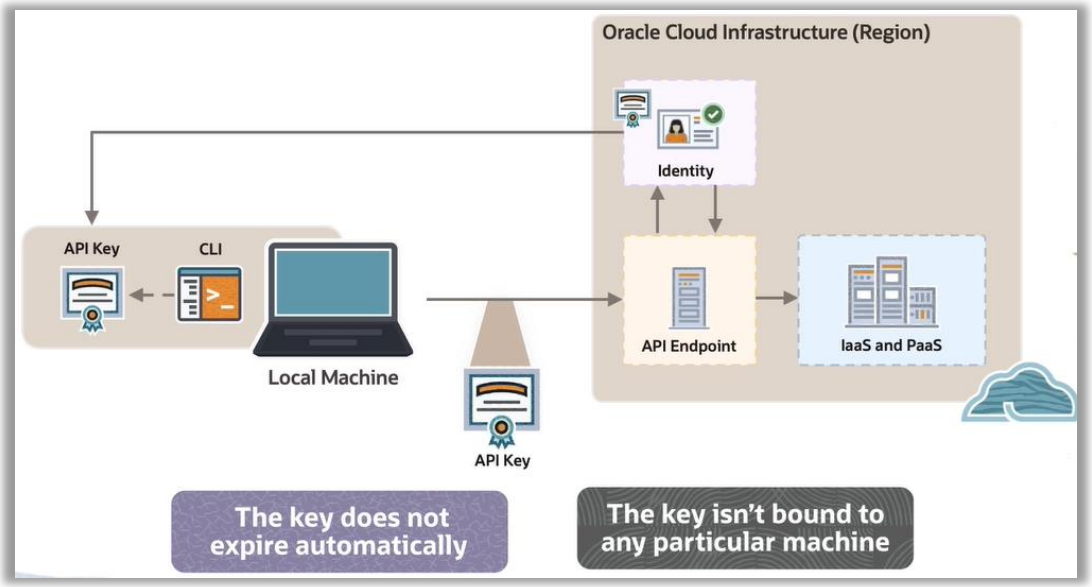
Example API Request

```
POST https://iaas.us-phoenix-1.oraclecloud.com/20160918/vcns
host: iaas.us-phoenix-1.oraclecloud.com
opc-retry-token: 239787fs987
Content-Type: application/json
HTTP headers required for authentication
Other HTTP request headers per the HTTP spec
{
  "compartmentId": "ocid1.compartment.oc1..exampleid",
  "displayName": "Virtual Cloud Network",
  "cidrBlock": "172.16.0.0/16"
}
```

In any case, the two types of identification that we'll be going over our **API keys** and **security tokens**. API keys are the **easier** but **less secure method** and to use it, you must generate it in through the OCI console, download it onto your machine, and then point your CLI to it. Then your CLI can attach it to all of the API requests it makes.

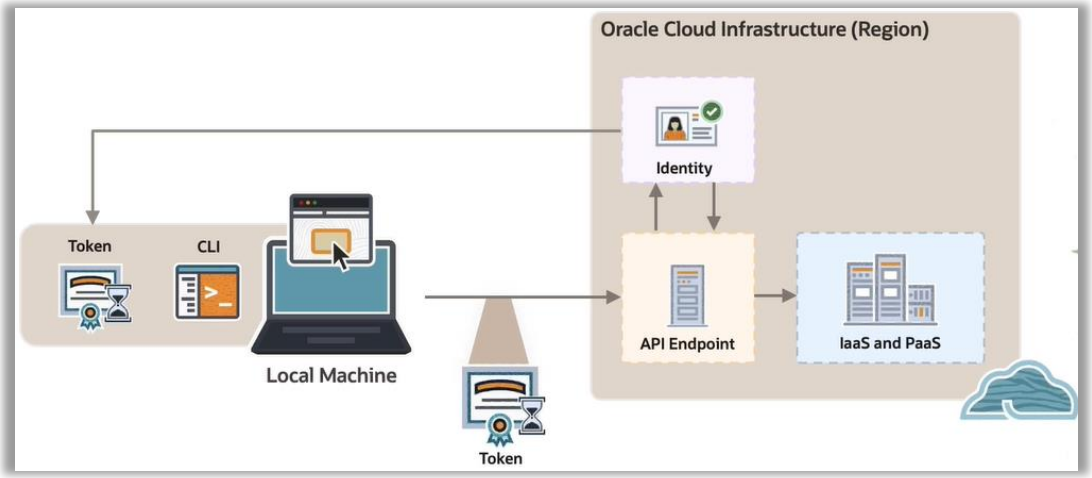
To keep in mind, there are two main reasons it's not particularly secure:

- **First**, the key does not automatically expire and It's valid until you specifically deactivate it with OCI.
- **Second**, the key isn't bound to any machine, so any machine that has the key can authenticate with it



While API keys are nice for use cases that don't require tight security, Oracle generally recommends the other method, **security tokens**.

To use a **security token**, you first tell your CLI that you want to use one. Then it sends a request to OCI, which in turn redirects you to a browser where you can sign in. Then OCI gives you a machine a session-based token that automatically expires after an hour unless refreshed. During this session, your CLI can attach the token to your API requests authenticating as you. So while you do have to sign in every time that you sit down to use the CLI, security tokens are much more secure than API tokens.



So far we've talked about using the CLI from local machines. But you can also use the CLI from machines within OCI like compute instances and functions and for this, exist another way, it is use compute instances itself, this is

called instance **principle authentication** and for functions, it's called resource principal authentication, but this is out of the scope of this document.

INTEGRATION REST API CLI WITH OCI SERVICES

One of the most common tasks to building REST or HTTP APIs is the interaction with a client. There are plenty of articles about API design, API description formats like **OpenAPI**, and lots of libraries and code generators used to talk to the API from code and other more were talk about client configuration and interaction.

But there's `curl`, and other many command clients to interact with the REST API, in fact much of the case use to provide examples, we find it first using `curl` formant, besides curl is a good client to interact with HTTP request, sometimes we require a graphical interaction and at this point is where postman client comes in action.

What is Postman?

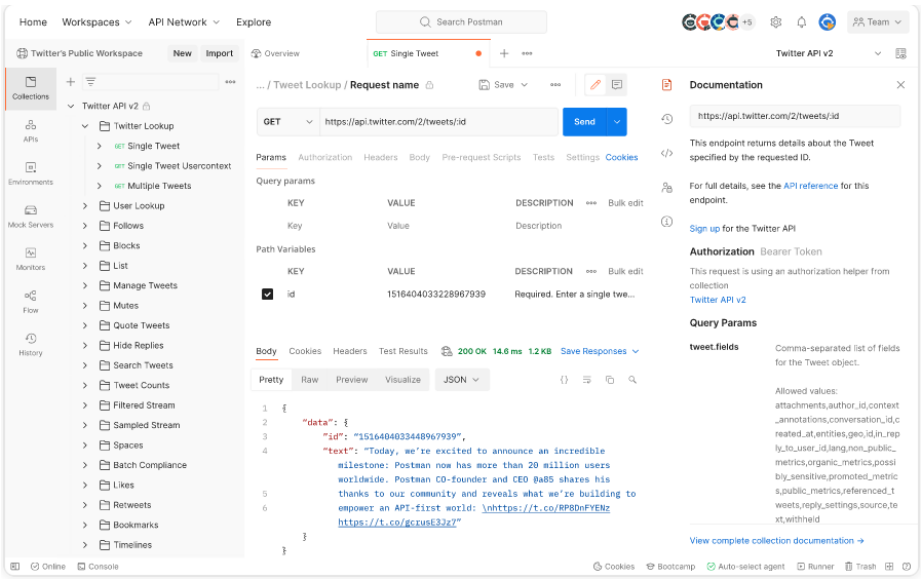
Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create better APIs faster.

With this graphical client has a lot of features**,** but we must consider that is scalable API testing tool that quickly integrates into the CI/CD pipeline, and generally used for browsing and testing APIs without coding. It is available on Windows, macOS, and Linux, and it started in 2012 as Abhinav Asthana's side project to simplify API workflow in testing and development.

Consider use Postman if:

- You are new to APIs.
- You want to get familiar with our platform without needing to build code.
- You need a troubleshooting and debugging tool when developing against the API.

And before working with the testing of API using POSTMAN and use one of the nicest features, we need to download and install it. After installation, the POSTMAN application interface can be used for reading, writing, and deleting the contents in API using client HTTP.

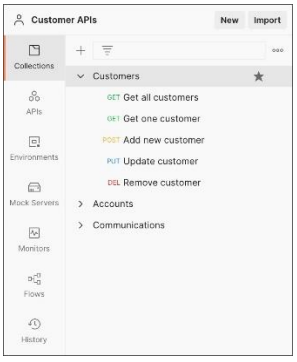


Postman is a GUI-based REST API invocation tool, has we have mentioned, this tool it's very popular among developers. However, even though Postman provides a lot of features, using the authentication scheme above for OCI REST APIs can be challenging.

Thankfully Postman allows automating such custom requirements using Sandboxes, Environments, and , Collections (as we have mentioned before), but first we need to provide the information related with the authentication.

What is a Postman Collection?

You can group your Postman requests and examples into collections to keep your workspace organized, to collaborate with teammates, to generate API documentation and API tests, and to automate request runs, and you can select **Collections** in the sidebar for the list of collections in a workspace.

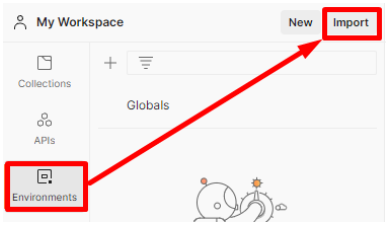


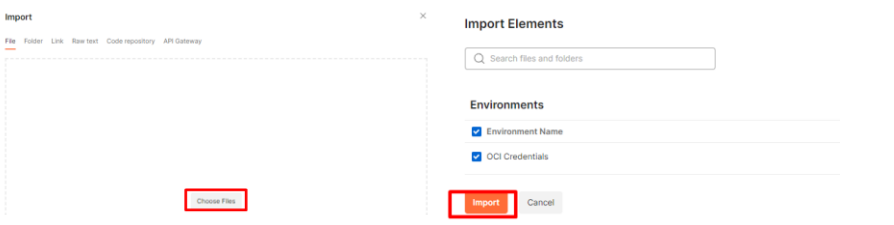
Proceed to the next section of this document to learn how to get started with the application and **OCI REST API** integration. For Postman application support, visit the Postman Support Center *(external link)* and to know how to manage collections, please visit the official postman documentation.

OCI Credentials Environment

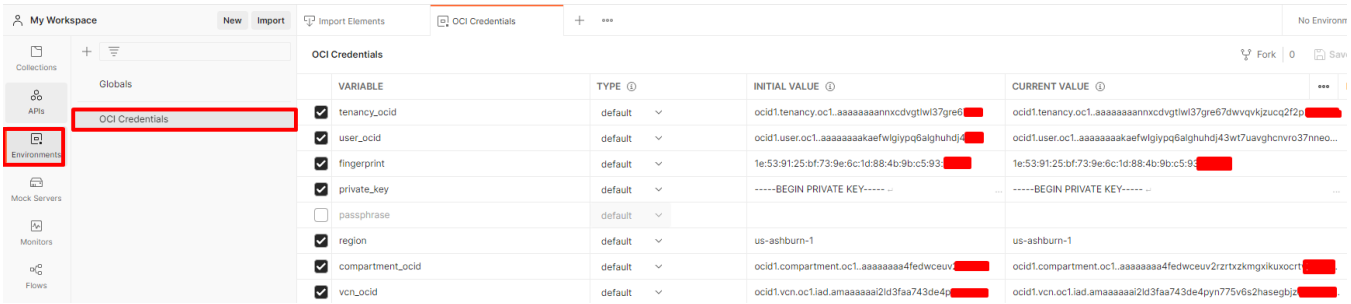
For the goal of this document, we have based on the file provided for Arshis Singh, where we have taken those code to define our main variables in order to authenticate for our REST API defined, so, for this we can load collection named `OCI Credentials.postman_environment` file, considering within it we have defined our

Postman Environment Variables, and basically it's here where we have our `OCI Credentials` information (API Key) and Under environment into Postman, click on 'Import', follow the process to see the OCI Collections related with the credentials imported:

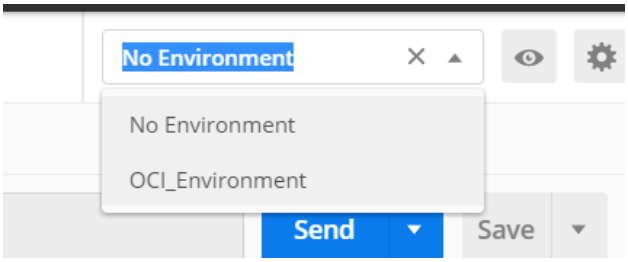




Once imported we can see the details something like this picture:



Or such as from the top right drop-down as shown below:



In fact, the information defined in OCI Credentials Postman environment, can be used within any of the SDK, CLI or REST API method, must remark define a few prerequisites (those were taken by the article published by Arshis Singh):

Table 1 Get OCI Credentials used for the Postman variables

POSTMAN VARIABLE	VALUE	EXPLANATION
tenancy_ocid	OCID of your tenancy. To get the value.	Require the OCID of the tenancy to get access through OCI-CLI o any API invocation. Make note of your OCI tenancy's OCID and the specific user's OCID. See Required Keys and OCIDs.
user_ocid	OCID of the user calling the API. **Example**: ocid1.user.oc1..<unique_ID>	Require the user OCID to perform REST API calls through postman.

		Consider only an IAM User can invoke the API, to consume the service, so this should exist. The information it's gotten by API Key generation. See Required Keys and OCIDs.
fingerprint	Fingerprint for the public key that was added to this user. To get the value.	It's generated once we have created and uploaded API public key. See Required Keys and OCIDs.
private_key	Contents of the private key file. You can open the file in an editor and copy the contents or use a command like `pbcopy < ~/.oci/oci_api_key.pem`	These key pair are generated once you click creation button each in the process of API Key creation. The key pair must be in PEM format. For instructions on generating a key pair in PEM format, see Required Keys and OCIDs.
passphrase	Passphrase used for the key.	This information is only required if its key is encrypted
region	An Oracle Cloud Infrastructure region.	See Regions and Availability Domains. Example: `us-ashburn-1`
compartment_ocid	OCID of the compartment that will be used for the API calls	

OCI Collections

Before to perform another task, we need to explain the following collections needed to perform API calls with OCI:

POSTMAN VARIABLE	EXPLANATION
OCI Object Storage Service API.postman_collection	Collection related with all the API calls (PUT, DEL, GET, LIST, etc. HTTP request) related for **Object Storage**.
OCI Notifications API.postman_collection	Collection related with all the API calls (PUT, DEL, GET, LIST, etc. HTTP request) related for related for OCI Notification service.
OCI Container Engine for Kubernetes API.postman_collection	Collection related with all the API calls (PUT, DEL, GET, LIST, etc. HTTP request) related for OCI Kubernetes Service.

These collections will be the core to perform REST API calls.

CONFIGURE REST API CLIENT AND IMPORTING COLLECTIONS

As cloud admin, if you have defined any authentication method to login and use the service through any SDK or OCI client, we can access then to the REST API from any application or programming platform that correctly and completely understands the Hypertext Transfer Protocol (HTTP) and has Internet connectivity.

Before we advance in this document, it is important to take considerations related with the service APIs with Oracle Cloud Infrastructure.

Policy for changes or new versions of APIs

Oracle will provide a notice of change or modification of any API related to a service with a notice of 12 months, for customers to make changes or code updates on the new published version.

API EndPoints API Reference for Object Storage

Each OCI service, there is an API Reference and respective Endpoints, for the POSTMAN variable environment, we have the following API Endpoints:

- Object Storage Service API
- Notifications API
- Container Engine for Kubernetes API

Validate the synchronization of the EndPoints

APIs Check the date and time, as well as if the 401 (NotAuthenticated) status is returned. This is due to the time desynchronization of + 5 min.

```
#curl -s --head <endpoint>|grep Date
#curl -s --head https://notification.us-ashburn-1.oci.oraclecloud.com|grep Date
```

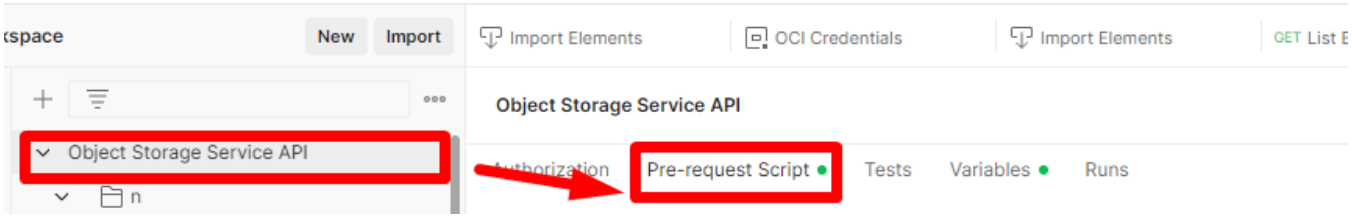
OCI API Authentication Scheme - All requests require to be encrypted

All OCI API requests must be signed, for the purpose of authentication, and OCI uses the IETF-draft-cavage-http-signatures-08 specification as its authentication scheme, which guarantees that the client is authenticated and the message integrity is preserved (HTTPS and TLS 1.2 protocols are required to establishes confidentiality). Please feel free to summary of Signing Steps (Request Signatures).

The authentication scheme is a bit complicated, below is a summary of the core requirements:

1. Generate ***current date***
2. Generate **API_Key** as "**tenancyOCID/UserOCID/APIKeyFingerprint**"
3. Generate Signing String
   a. If POST/PUT, generate **SHA256**, **base64** encoded hash of the body and include as part of signing string)
4. Generate RSA Digital Signature of the **SHA256** hash of the Signing String generated above.
5. Set the formatted Authorization header that includes API Key, Digital Signature (from #4 above) and other parameters.
6. Invoke the API with '`Authorization`', '`Date`', '`Content-Type`', and '`x-content-sha256`' hash of the body (a from the #3) as `Header` parameters.

To perform digital for encryption and digital signatures over each request made we can perform right click on Object Storage Service API and then go to pre-request Script and paste the we require to script section with the 'jsrsasign-all-min.js' content.



That's it!! Now you can use the OCI Object Storage API to invoke any REST APIs:

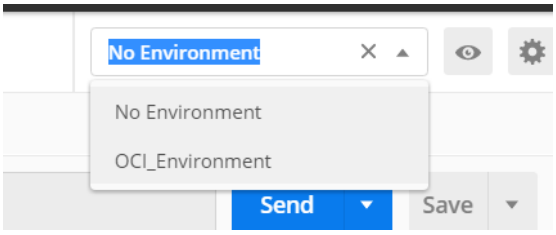
This script will execute before every request in this collection. Learn more about Postman's execution order ↗

```
/**
* Copyright (c) 2021, 2022, Oracle and/or its affiliates.  All rights reserved.
* This software is dual-licensed to you under the Universal Permissive License (UPL) 1.0 as shown at https://oss.oracle.com/licenses/upl
* or Apache License 2.0 as shown at http://www.apache.org/licenses/LICENSE-2.0. You may choose either license.
*/

// -----------------------------------------
// Oracle Cloud Infrastructure
// Request Signature and OKE Kubernetes token
// v 2.0.0
// -----------------------------------------

// Checks if is an Oracle Cloud domain. Do nothing if not.
if (!pm.variables.replaceIn(pm.request.url.getHost()).includes("oraclecloud.com")) {return 0}

// Check OCI Credentials variables
if (!pm.environment.get("tenancy_ocid")) { throw new Error("Tenancy OCID (tenancy_ocid) Variable Not Set") }
if (!pm.environment.get("user_ocid")) { throw new Error("User OCID (user_ocid) Variable Not Set") }
if (!pm.environment.get("fingerprint")) { throw new Error("Key Fingerprint (fingerprint) Variable Not Set") }
if (!pm.environment.get("private_key")) { throw new Error("Private Key (private_key) Variable Not Set") }
if (!pm.environment.get("region")) { throw new Error("OCI Region (region) Variable Not Set") }

// jsrsasign (RSA-Sign JavaScript Library)
// https://github.com/kjur/jsrsasign LICENSE: MIT License
let navigator = {}
let window = {}
eval(pm.collectionVariables.get("jsrsasign"))

// Const Variables
const apiKeyId = `${pm.environment.get("tenancy_ocid")}/${pm.environment.get("user_ocid")}/${pm.environment.get("fingerprint")}`
const privateKey =  pm.environment.get("private_key")
const passphrase =  pm.environment.get("passphrase")
const rawClusterUrl = "https://containerengine.{{region}}.oraclecloud.com/cluster_request/{{cluster_ocid}}"

let authorizationString = (type) => {

    // Resolve host and paths
    let rawUrl = pm.request.url.getHost()
    let pathWithQuery = pm.request.url.getPathWithQuery()
    pathWithQuery.endsWith("?")?pathWithQuery=pathWithQuery.slice(0, -1):""
    if (type=="oke") {
        rawUrl = rawClusterUrl
        pathWithQuery = ""
```

Load REST API call collections to Postman

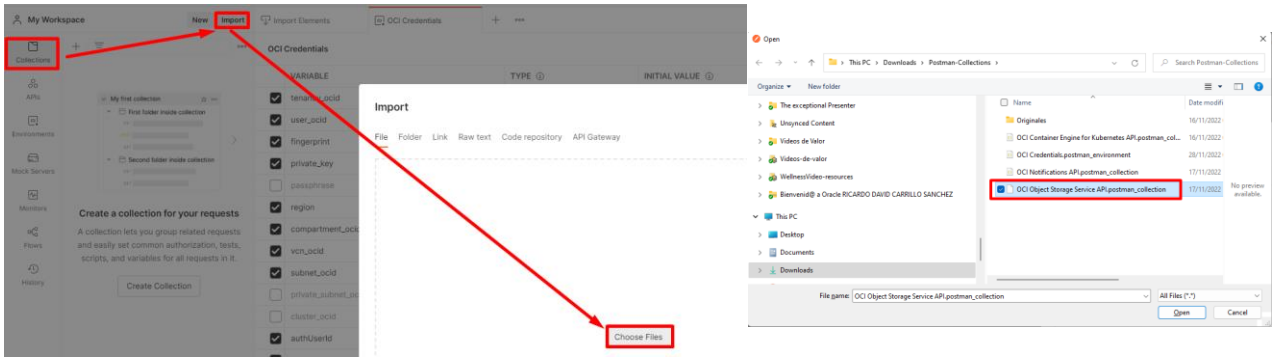
To load all Postman Collections used to generate REST API Calls, the procedure it's the same described in OCI Credentials Environment section:

1. First, select the OCI Credentials environment:

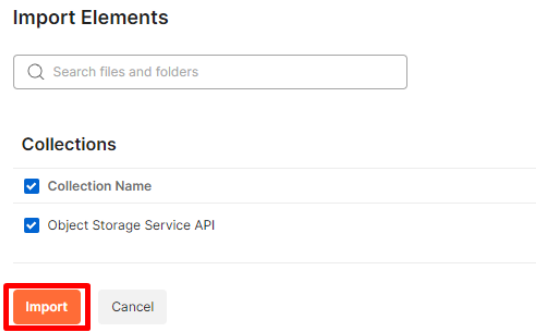


2. Perform the following steps:

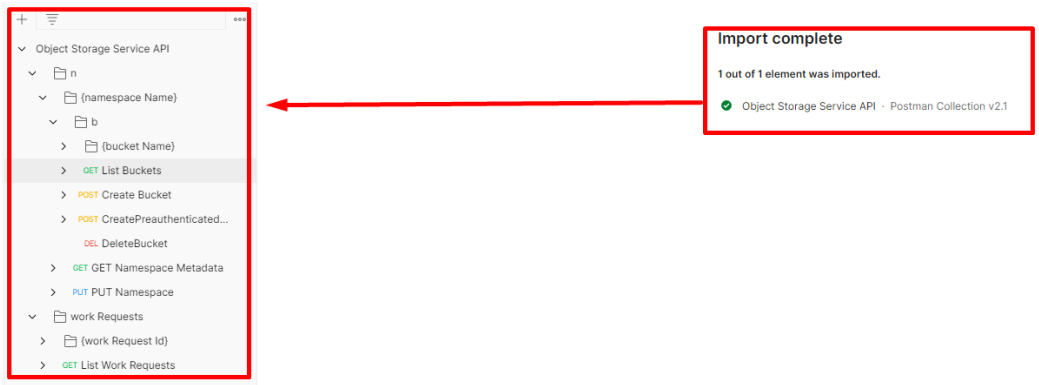
a. Open Postman, and under Collection section, click import button and chose the Collection needed (OCI Object Storage API collection):



b. Import the Collection selected:



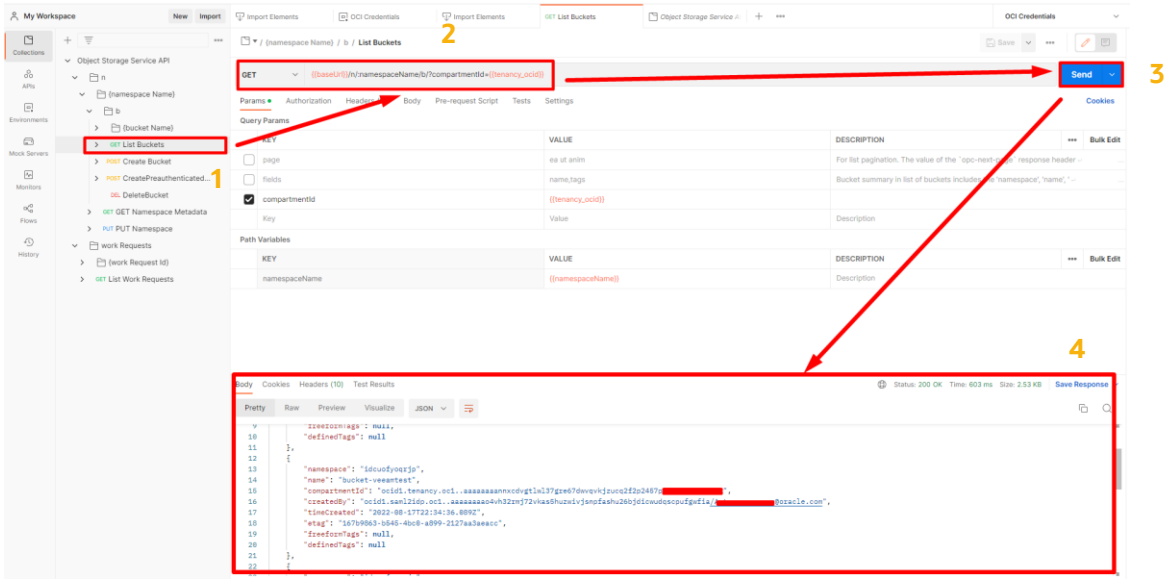
c. Once imported, we can see in left menu the OCI Object Storage API list request methods to be used:



d. Repeat the step from **a** to **c**, with the other Collections (`OCI Notifications API.postman_collection`, and `OCI Container Engine for Kubernetes API.postman_collection`).

REST API OCI SERVICES

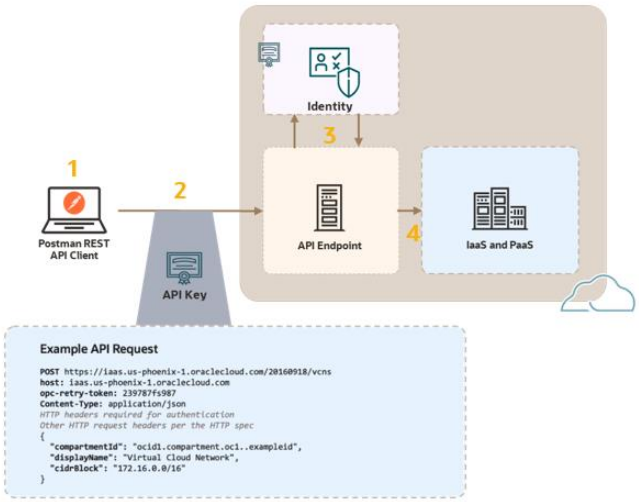
Once we have imported the collections, we can start to perform some activities such as create or execute any request defined in **Object Storage API**, for example in the following image, we see the List Bucket **based** in GET method:



The flow showed in image above can be described as following:

1. Select the method to execute,
2. Validate the GET method
3. Click on send button to perform execution
4. Get the result of execution.

The result expected is to get the bucket list within the tenancy and to understand the method performed, we need to see the main flow how the API request works in OCI:

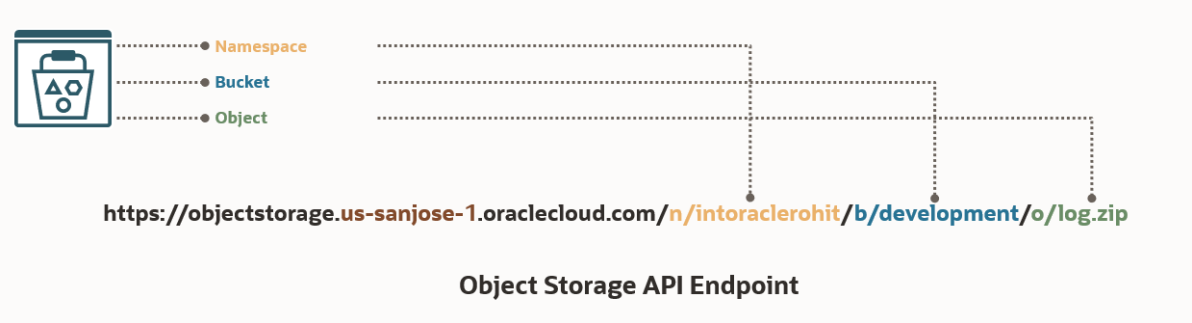


The image above we can see the flow as follows:

1. We see the Postman client make the request, and at the same time the Postman client,
2. The Postman Client, also perform to the API Key.
3. The request made to get the authentication and authorization between the end points and the IDCS
4. Perform the API request to interact with the OCI Service.

Some Object Storage API calls

To understand the Object Storage API, would be convenient to understand how the object storage works:



Object Storage API Endpoint

From image above, we can observe 3 different elements:

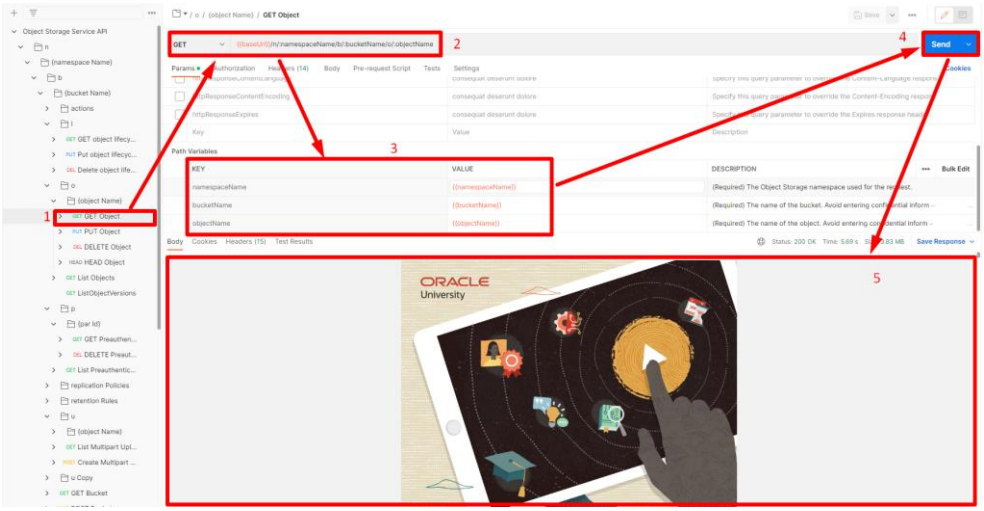
- **Namespace**: Is the name space form the tenancy.
- **Bucket**: is the name of the bucket
- **Object**: Is the file uploaded and stored within the bucket.

GET Object (Specific object)

To get a specific object with the GET method:

GET Object	{{baseUrl}}/n/:namespaceName/b/:bucketName/o/:objectName

If we use Postman client:



To explain the above flow, we have the following points:

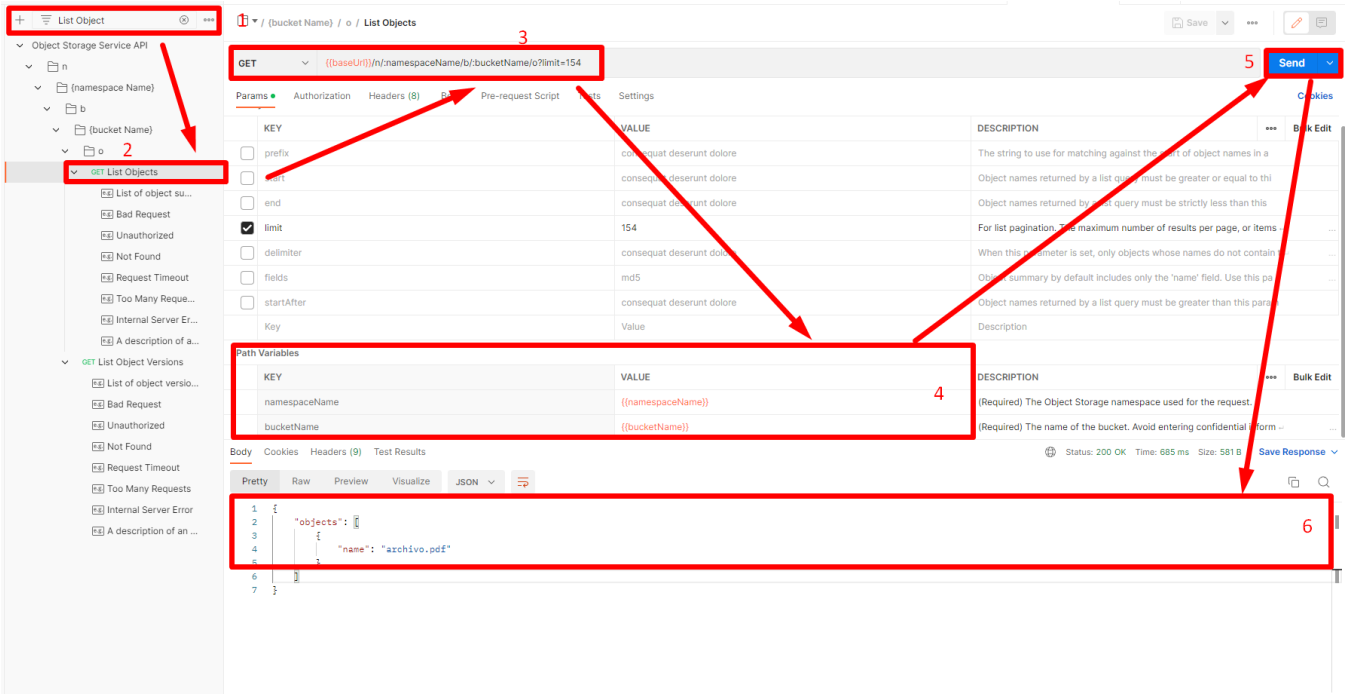
1. Detect or locate the API Call that we want to perform or execute.
2. Check the method related with the action
3. Modify the mandatory variables, in this case the mandator variables are defined as general scope within the OCI Environment file.
4. After validating the API mandatory variables, we can perform click send button …
5. And wait to see the downloaded file by REST API method.

GET Object (All objects)

To List all the objects within a bucket we use List Objects method:

GET Object	{{baseUrl}}/n/:namespaceName/b/:bucketName/o?limit=154

If we use Postman client:



From the image above:

1. Look for the method help me to list all the objects
2. Select the request method “List Objects” (in the background is get http method)
3. If it is necessary modify the key values, in this example we have {{namespaceName}} and {{bucketName}, these are global variables that we can use to specify the namespace of the bucket and the bucket name.
4. If we do not have any change to perform in the variables, we proceed to click on send button
5. Once we have clicked on the send button, we can expect a list in JSON format of all objects saved on the bucket.

CONNECT WITH US

Call +1.800.ORACLE1 or visit oracle.com.
Outside North America, find your local office at oracle.com/contact.

blogs.oracle.com facebook.com/oracle twitter.com/oracle

REST API for OCI
November 2222
Author:
Contributing Authors: