Module 4: Understanding and Using APIs

DevNet Associate v1.0

Module Objectives

Module Title: Understanding and Using APIs

Module Objective: Create REST API requests over HTTPS to securely integrate services.

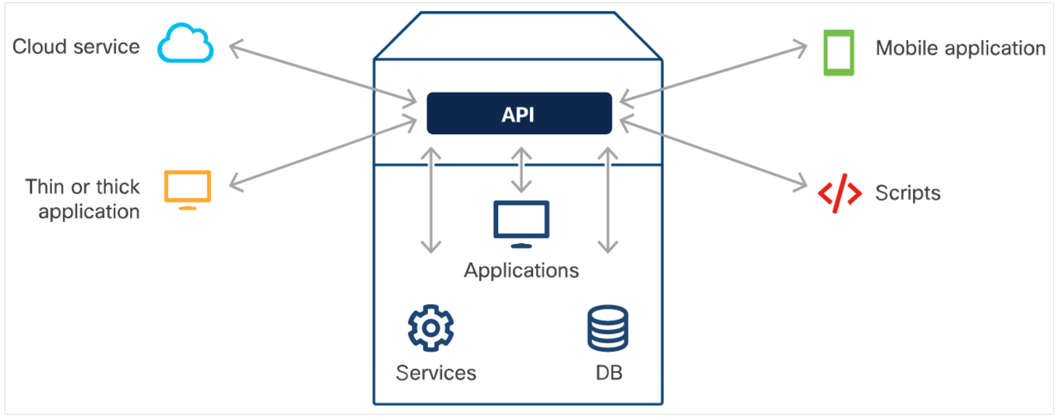
Topic Title	Topic Objective
Introducing APIs	Explain the use of APIs.
API Design Styles	Compare synchronous and asynchronous API design styles.
API Architecture Styles	Describe common API architecture styles.
Introduction to REST APIs	Explain the functions of REST APIs.
Authenticating a REST API	Create REST API requests over HTTPS to securely integrate services.
API Rate Limits	Explain the purpose of API rate limits
Working with Webhooks	Explain the use of webhooks.
Troubleshooting API calls	Explain how to troubleshoot REST APIs

4.1 Introducing APIs

What is an API?

- An Application Programming Interface (API) allows one piece of software talk to another.
- It uses common web-based interactions or communication protocols and its own proprietary standards.
- An API determines what type of data, services, and functionality the application exposes to third parties.
- By providing APIs, applications can control what they expose in a secure way.

Example of different types of API integrations



Why use APIs?

- APIs are built to be consumed programmatically by other applications and they can also be used by humans who want to interact with the application manually.

Use cases of APIs are as follows:

- **Automation tasks** – Build a script that performs manual tasks automatically and programmatically.
- **Data integration** – An application can consume or react to data provided by another application.
- **Functionality** – An application can integrate another application's functionality into its product.

Understanding and Using APIs

Why are APIs so popular?

- APIs have existed for decades, but exposure and consumption of APIs has grown exponentially in the last 10 years or so.
- Most modern APIs are designed into the product and are thoroughly tested.
- Simplified coding languages such as Python have made it possible for non-software engineers to build applications and consume APIs.

4.2 API Design Styles

API Design Styles

Types of Design Styles

- A product’s set of APIs may consist of both synchronous and asynchronous designs, where each API’s design is independent of the others.
- The application consuming the API manages the response differently depending on the API design.

Synchronous APIs

- Synchronous APIs respond to a request directly by providing data immediately.

When are APIs synchronous?

- APIs are synchronous when the data for the request is readily available.

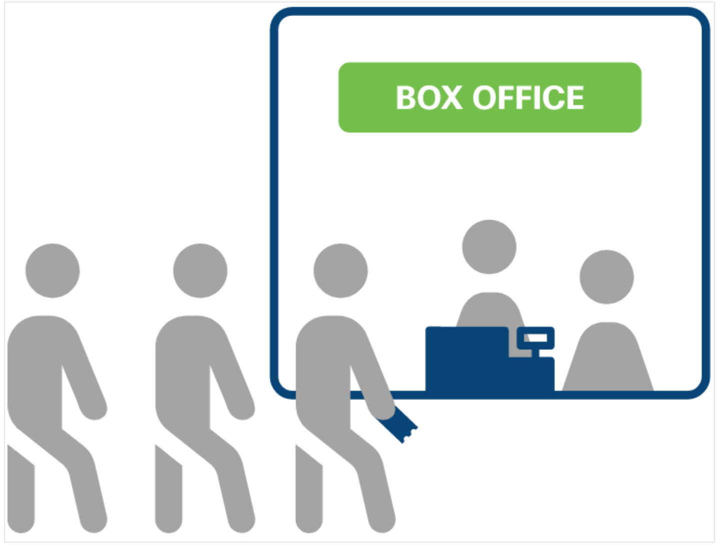
Benefits of a synchronous API design

- Synchronous APIs enable the application to receive data immediately. If the API is designed correctly, the application performance will be better.

Client side processing

- The application that is making the API request must wait for the response before performing any additional code execution tasks.

Synchronous APIs



Tickets are sold in first-come, first served order. This is a synchronous process.

Asynchronous APIs

- Asynchronous APIs provide a response (with no data) to signify that the request has been received.

When are APIs asynchronous?

- APIs are asynchronous when the request takes some time for the server to process or if data isn't readily available.

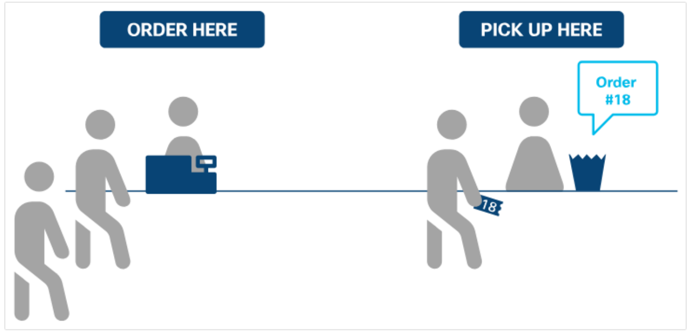
Benefit of asynchronous API design

- Asynchronous APIs allow the application to continue execution without being blocked till the server processes the request, thus resulting in better performance.

Client-side processing

- With asynchronous processing, the design of the API on the server side defines the requirement on the client side.

Asynchronous APIs



4.3 API Architectural Styles

API Architectural Styles

Common Architectural Styles

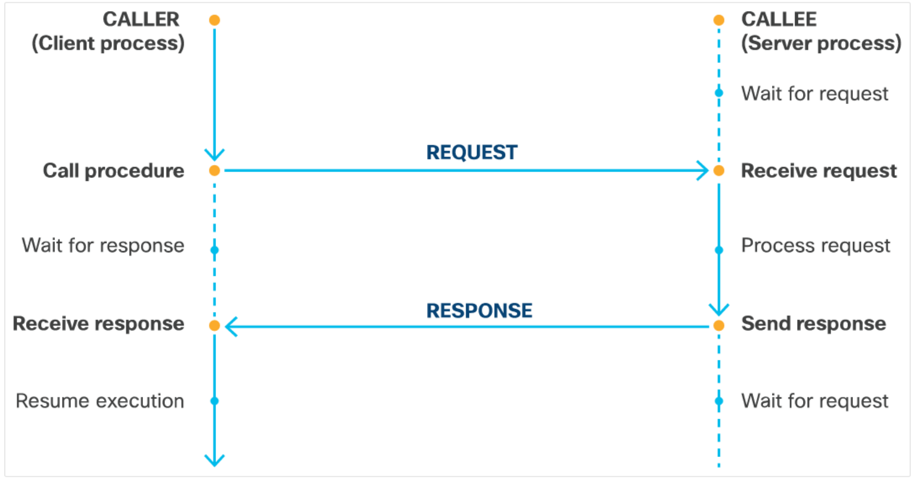
- There are certain standards, protocols, and specific architectural styles which make it easier for consumers of the API to learn and understand the API.
- The three most popular types of API architectural styles are :
  - RPC
  - SOAP
  - REST

API Architectural Styles

Remote Procedure Call (RPC)

- Remote Procedure Call (RPC) is a request-response model that allows an application to make a procedure call to another application.
- When RPC is called to a client, the method gets executed and the results get returned.
- RPC is an API style that can be applied to different transport protocols such as:
  - XML-RPC
  - JSON-RPC
  - NFS (Network File System)
  - Simple Object Access Protocol (SOAP)

Remote Procedure Call client-server request/response model



Simple Object Access Protocol (SOAP)

- Simple Object Access Protocol (SOAP) is a XML- based messaging protocol. It is used for communication between applications on different platforms or built with different programming languages.

SOAP is:

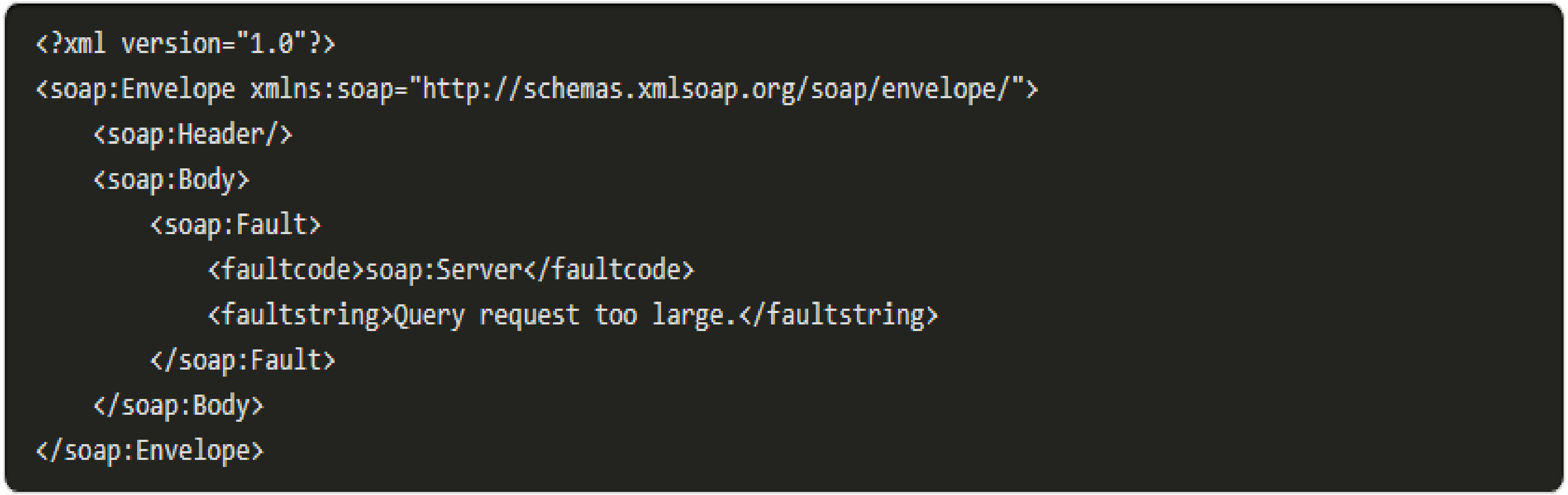
- **Independent**: All applications can communicate with each other and run on different operating systems
- **Extensible**: Add features such as reliability and security
- **Neutral**: Can be used over any protocol, including HTTP, SMTP, TCP, UDP, or JMS

A SOAP message is an XML document that may contain the following four elements:

- **Envelope** – the root element of XML document.
- **Header** - contains application-specific information such as authorization, SOAP-specific attributes and so on
- **Body** - contains the data to be transported to the recipient
- **Fault** - provides error and/or status information.

Simple Object Access Protocol (SOAP) (Contd.)

- The following screenshot is an example of a SOAP message:



```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header/>
    <soap:Body>
        <soap:Fault>
            <faultcode>soap:Server</faultcode>
            <faultstring>Query request too large.</faultstring>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

API Architectural Styles

REpresentational State Transfer (REST)

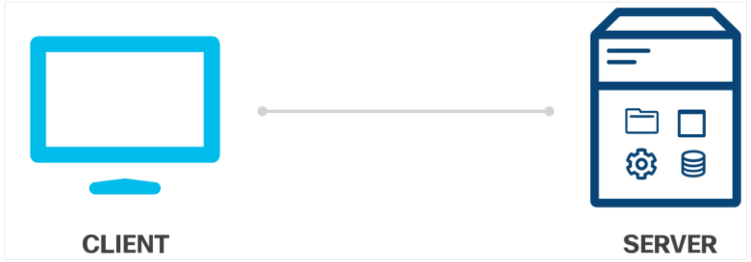
- REpresentational State Transfer (REST) is an architectural style authored by Roy Thomas Fielding.
- Roy has established six constraints that can be applied to any protocol in REST.

REpresentational State Transfer (REST) (Contd.)

Client-server

- The client and server should be independent of each other.
- This will enable the client to be built for multiple platforms which will simplify the server side components.

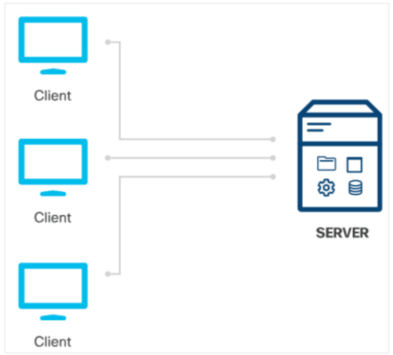
REST client-server model



Stateless

- Requests from the client to the server must contain REST client-server model and all the information which the server needs to make the request.
- The server cannot contain session states.

REST stateless model



API Architectural Styles

REpresentational State Transfer (REST) (Contd.)

Cache model:

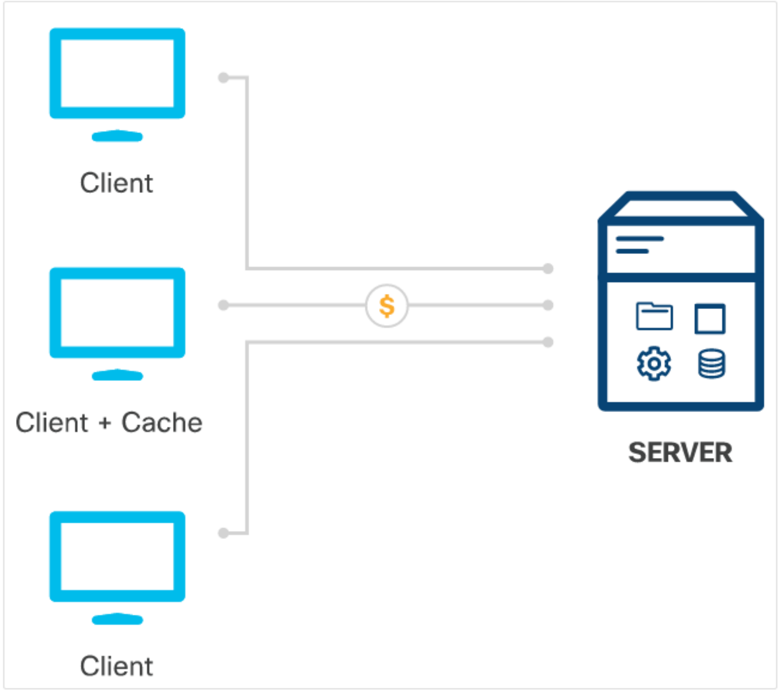
- Responses from the server must state whether the response is cacheable or non-cacheable.
- If it is cacheable, the client can use the data from the response for later requests.

Uniform interface:

The interface between the client and the server adhere to the four principles:

- Identification of resources
- Manipulation of resources through representations
- Self-descriptive messages
- Hypermedia as the engine of application state.

REST cache model

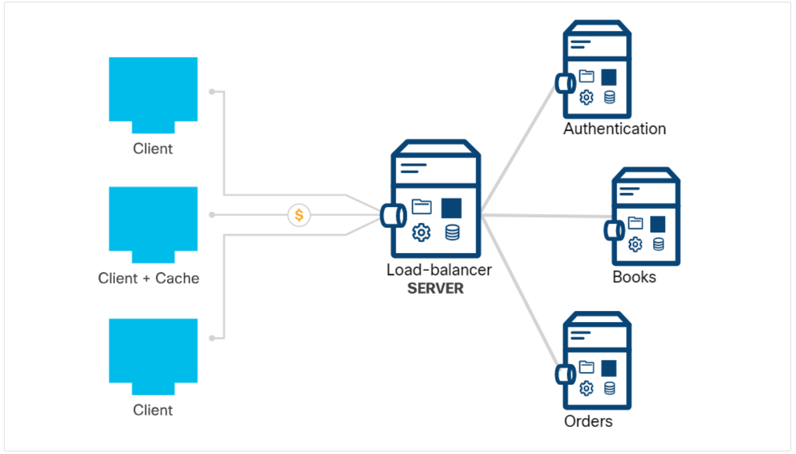


REpresentational State Transfer (REST) (Contd.)

Layered system:

- The Layered system consists of different hierarchical layers in which each layer provides services only to the layer above it.
- As a result, it consumes services from the layer below.

REST layered system model



Code-on-demand:

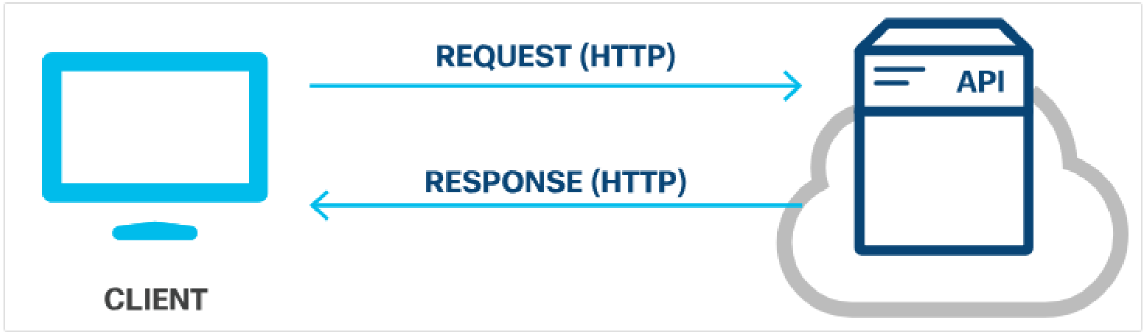
- The information returned by a REST service can include executable code (for example, javascript) or links intended to usefully extend client functionality.
- The constraint is optional because execution of third-party codes introduces potential security risks.

4.4 Introduction to REST APIs

REST Web Service APIs

- A REST Web service API (REST API) is a programming interface that communicates over HTTP.
- REST APIs use the same concepts as the HTTP protocol which are as follows:
  - HTTP requests/responses
  - HTTP verbs
  - HTTP status codes
  - HTTP headers/body

REST API request/response model



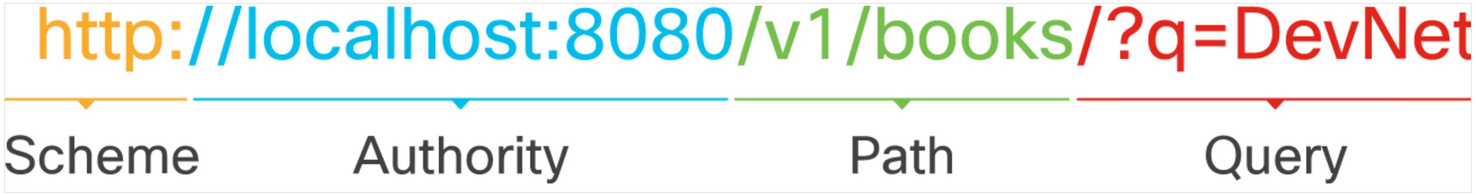
REST API Requests

- REST API requests are HTTP requests that are a way for an application (client) to ask the server to perform a function.
- REST API requests are made up of four major components:
  - Uniform Resource Identifier (URI)
  - HTTP Method
  - Header
  - Body

REST API Requests (Contd.)

The Uniform Resource Identifier (URI), also referred to as Uniform Resource Locator (URL), identifies which resource the client wants to manipulate. The components of a URI are:

- **Scheme:** specifies which HTTP protocol should be used, http or https.
- **Authority:** consists of two parts, namely, host and port.
- **Path:** represents the location of the resource, the data or object, to be manipulated on the server.
- **Query:** provides additional details for scope, filtering, or to clarify a request.



Different components of a URI

REST API Requests (Contd.)

HTTP method:

- REST APIs use the standard HTTP methods to communicate to the web services for which action is being requested for the given resource.
- The suggested mapping of the HTTP Method to the action is as follows:

HTTP Method	Action	Description
POST	Create	Create a new object or resource.
GET	Read	Retrieve resource details from the system.
PUT	Update	Replace or update an existing resource.
PATCH	Partial Update	Update some details from an existing resource.
DELETE	Delete	Remove a resource from the system.

REST API Requests (Contd.)

Header:

- HTTP headers are formatted as name-value pairs that are separated by a colon (:), [name]:[value].

Two types of headers:

- **Request headers** - Include additional information that does not relate to the content of the message.

Key	Example Value	Description
Authorization	Basic dmFncmFudDp2YWdyYW	Provide credentials to authorize the request

- **Entity headers** - Additional information that describes the content of the body of the message.

Key	Example Value	Description
Content-Type	application/ json	Specify the format of the data in the body

REST API Requests (Contd.)

Body:

- The body of the REST API request contains the data pertaining to the resource that the client wants to manipulate.
- REST API requests that use the HTTP method POST, PUT, and PATCH typically include a body.
- The body is optional depending on the HTTP method.
- If the data is provided in the body, then the data type must be specified in the header using the Content-Type key.

Introduction to REST APIs

REST API Responses

- REST API responses are HTTP responses that communicate the results of a client's HTTP request.
- REST API Responses are made up of three major components:
  - HTTP Status
  - Header
  - Body

REST API Responses (Contd.)

HTTP Status

- The HTTP status code help the client determine the reason for the error and can sometimes provide suggestions for fixing the problem.
- HTTP status codes consists of three digits, where the first digit is the response category and the other two digits are assigned in numerical order.
- There are five different categories of HTTP status codes:
  - **1xx – Informational** – for informational purposes, responses do not contain a body
  - **2xx – Success** – the server received and accepted the request
  - **3xx – Redirection** – the client has an additional action to take to get the request completed
  - **4xx -- Client Error** – the request contains an error such as bad syntax or invalid input
  - **5xx -- Server Error** – unable to fulfill the valid requests.

REST API Responses (Contd.)

The common HTTP status codes are as follows:

HTTP Status Code	Status Message	Description
200	Ok	Request was successfully and typically contains a payload (body)
201	Created	Request was fulfilled and the requested resource was created
202	Accepted	Request has been accepted for processing and is in process
400	Bad Request	Request will not be processed due to an error with the request
401	Unauthorized	Request does not have valid authentication credentials to perform the request
403	Forbidden	Request was understood but has been rejected by the server
404	Not Found	Request cannot be fulfilled because the resource path of the request was not found on the server
500	Internal Server Error	Request cannot be fulfilled due to a server error
503	Service Unavailable	Request cannot be fulfilled because currently the server cannot handle the request

REST API Responses (Contd.)

- **Header** - The header in the response is to provide additional information between the server and the client in the name-value pair format that is separated by a colon (:), [name]:[value].There are two types of headers: **response headers** and **entity headers.**
  - **Response headers** – It contains additional information that doesn't relate to the content of the message. The typical response headers for a REST API request include:

Key	Example Value	Description
Set-Cookie	JSESSIONID=30A9DN810FQ428P; Path=/	Used to send cookies from the server
Cache-Control	Cache-Control: max-age=3600, public	Specify directives which MUST be obeyed by all caching mechanisms

  - **Entity headers** – They are additional information that describes the content of the body of the message. One common entity header specifies the type of data being returned:

Key	Example Value	Description
Content-Type	application/json	Specify the format of the data in the body

REST API Responses (Contd.)

Response Pagination

- Response Pagination enables data to be broken into chunks.
- Most APIs that implement pagination use the query parameter to specify which page to return in the response.

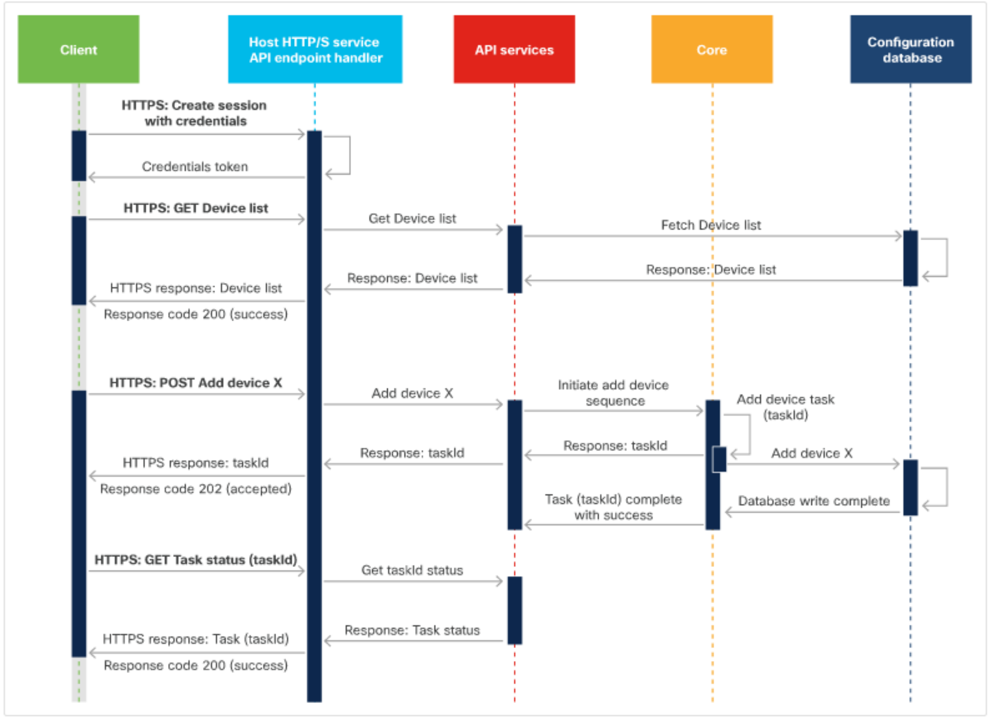
Compressed response data

- Compressed data reduces large amount of data that cannot be paginated
- To request a data compression, the request must add the Accept-Encoding field to the request header. The accepted values are:
  - gzip
  - compress
  - deflate
  - br
  - identity
  - *

Using Sequence Diagrams with REST API

- Sequence diagrams are used to explain a sequence of exchanges or events.
- API request sequence diagram has three separate sequences:
  - **Create session**- the starting request is labeled as HTTPS: Create Session w/credentials.
  - **Get devices** - request a list of devices from the platform.
  - **Create device** – begins with a POST request to create a device.

API Request/Response Sequence Diagram



4.5 Authenticating to a REST API

Authenticating to a REST API

REST API Authentication

- REST APIs require authentication so that random users cannot access, create, update, or delete information incorrectly or maliciously.
- Some APIs that do not require authentication are read-only and they do not contain any critical or confidential information.

Authenticating to a REST API

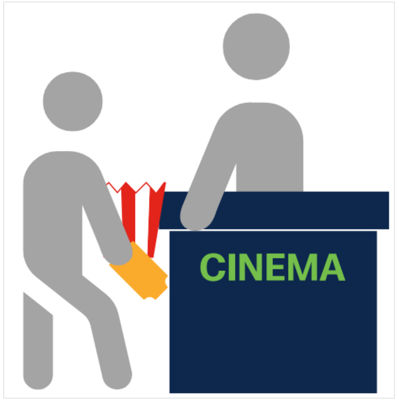
Authentication Vs. Authorization

Authentication:

- Authentication proves the user’s identity.
- For example, when you go to the airport, you have to show your government-issued identification or use biometrics to prove that you are the person you claim to be.

Authorization:

- Authorization defines the user access.
- It is the act where the user is proving to have permissions to perform the requested action on that resource.
- For example, when you go to a concert, all you need to show is your ticket to prove that you are allowed in.



Authenticating to a REST API

Authentication mechanisms

The common types of authentication mechanisms include:

- **Basic authentication**: It transmits credentials as username/password pairs separated with a colon (:) and encoded using Base64.
- **Bearer authentication**: It uses a bearer token, which is a string generated by an authentication server such as an Identity Service (IdS).
- **API Key**: It is a unique alphanumeric string generated by the server and assigned to a user. The two types of API keys are public and private.

Authenticating to a REST API

Authorization Mechanisms

- Open Authorization (Oauth) combines authentication with authorization.
- Oauth was developed as a solution to insecure authentication mechanisms.
- Oauth has increased security as compared to other options.
- There are two versions of Oauth - OAuth 1.0 and OAuth 2.0, where OAuth 2.0 is not backwards compatible.
- OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service.
- OAuth allows the user to provide credentials directly to the authorization server [Identity Provider (IdP) or an Identity Service (IdS)], to obtain an access token to share with the application.
- The process of obtaining the token is called a flow.

Lab - Explore REST APIs with API Simulator and Postman

In this lab, you will complete the following objectives:

- **Part 1**: Launch the DEVASC VM
- **Part 2**: Explore API Documentation Using the API Simulator
- **Part 3**: Use Postman to Make API Calls to the API Simulator
- **Part 4**: Use Python to Add 100 Books to the API Simulator

4.6 API Rate Limits

What are Rate Limits?

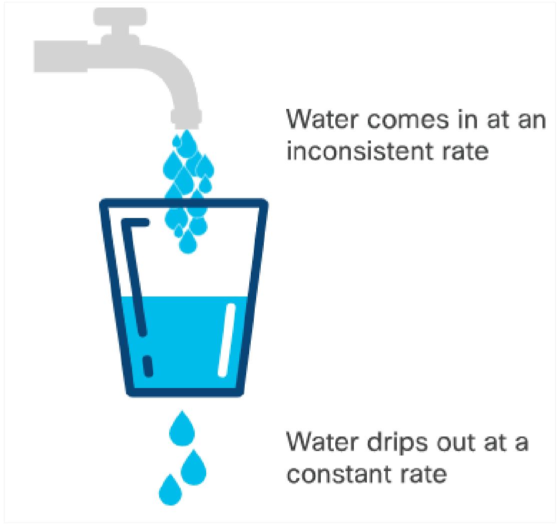
- An API rate limit is a way for a web service to control the number of requests a user or an application can make per defined unit of time.
- Rate limiting helps to :
  - avoid a server overload from too many requests at once
  - provide better service and response time to all users
  - protect against a Denial-of-Service (DoS) attack

Rate Limit Algorithms

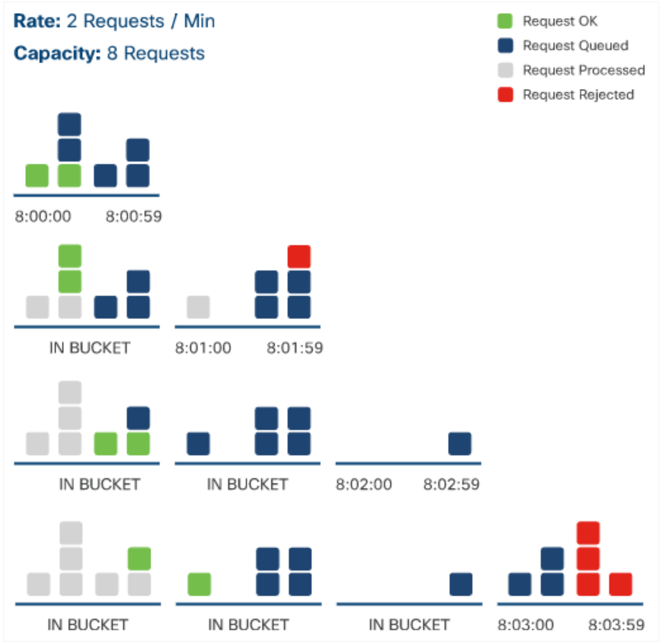
Leaky bucket

- This algorithm puts all incoming requests into a request queue in the order in which they were received.
- The incoming requests can come in at any rate, but the server will process the requests from the queue at a fixed rate.
- If the request queue is full, the request is rejected

Visual representation of leaky bucket algorithm


Water comes in at an inconsistent rate
Water drips out at a constant rate

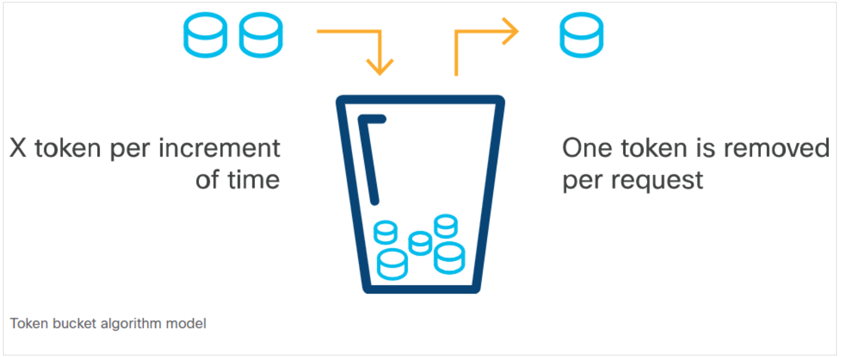
Example of leaky bucket algorithm



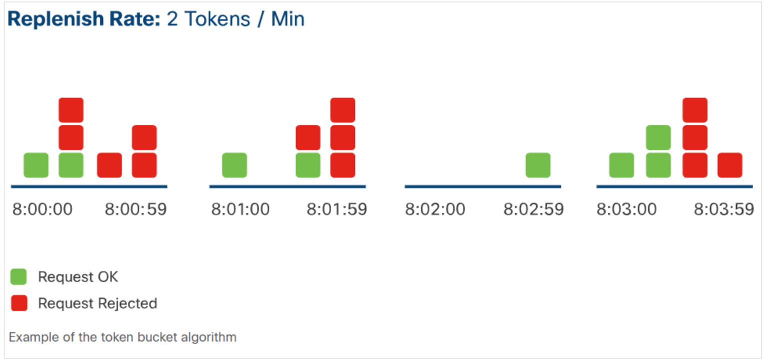
Rate Limit Algorithms (Contd.)

Token bucket

- This algorithm gives each user a defined number of tokens they can use within a certain increment of time.
- When the client makes a request, the server checks the bucket to make sure it contains at least one token. If so, it removes that token and processes the request. If there isn't a token available, it rejects the request.
- The client must calculate the number of tokens he currently has to avoid rejected requests.


X token per increment of time
One token is removed per request
Token bucket algorithm model

Token Bucket Algorithm model



Example of the token bucket algorithm

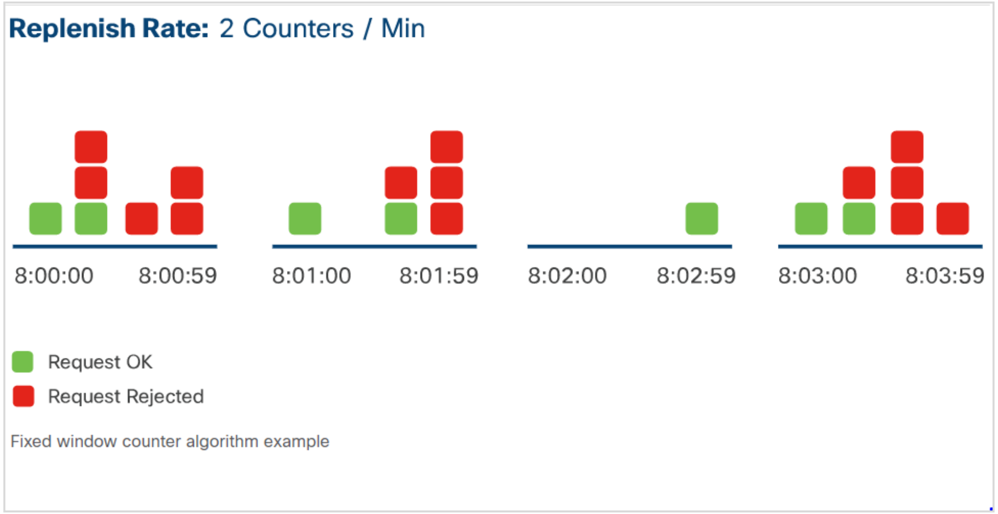
API Rate Limits

Rate Limit Algorithms (Contd.)

Fixed window counter

- In fixed window counter algorithm, a fixed window of time is assigned a counter to represent how many requests can be processed during that period.
- When the server receives a request, the counter for the current window of time must be zero.
- When the request is processed, the counter is deducted. If the limit for that window of time is met, all subsequent requests within that window of time will be rejected.

Fixed Window Counter Algorithm Example



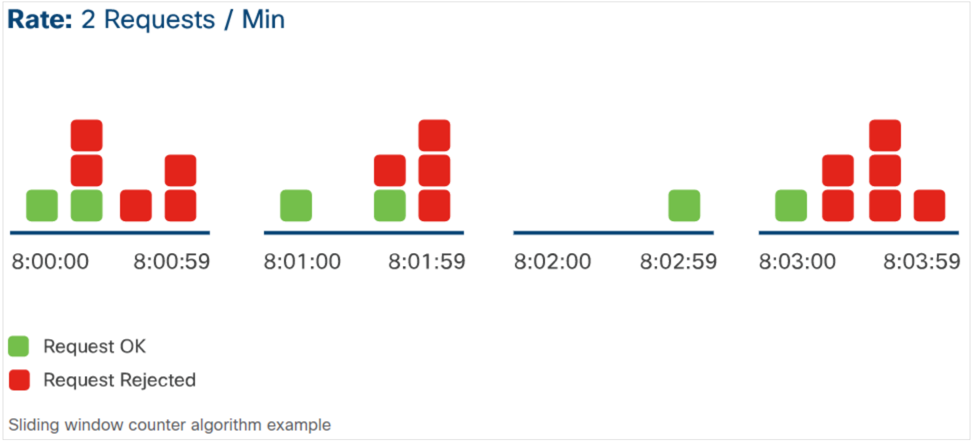
Fixed window counter algorithm example

Rate Limit Algorithms (Contd.)

Sliding window counter

- This algorithm allows a fixed number of requests to be made in a set duration of time.
- When a new request is made, the server counts how many requests have already been made from the beginning of the window to the current time to determine if the request should be processed or rejected.
- The client needs to ensure that the rate limit does not exceed at the time of the request.

Sliding Window Counter Algorithm Example



Sliding window counter algorithm example

Knowing the Rate Limit

- Many rate limiting APIs add details about the rate limit in the response's header.
- The commonly used Rate Limit keys include:
  - **X-Rate Limit-Limit** - The maximum number of requests that can be made in a specified unit of time.
  - **X-Rate Limit-Remaining** - The number of pending requests that the requester can make in the current rate limit window
  - **X-Rate Limit-Reset** - The time when the rate limit window will reset.

Exceeding the Rate Limit

- When the rate limit is exceeded, the server automatically rejects the request and sends back an HTTP response to the user.
- The response containing the 'rate limit exceeded' error also includes a meaningful HTTP status code.
- The most commonly used HTTP status codes are 429: Too Many Requests or 403: Forbidden.

4.7 Working with Webhooks

What is a Webhook?

- A Webhook is an HTTP callback, or an HTTP POST, to a specified URL that notifies the application when a particular activity or event is occurred in the resources.
- With webhooks, applications are more efficient as polling mechanisms are not required.
- Webhooks are also known as reverse APIs, because applications subscribe to a webhook server by registering with the webhook provider.
- Multiple applications can subscribe to a single webhook server.

Examples:

- The Cisco DNA Center platform provides webhooks that enable third-party applications to receive network data when specified events occur.
- You can create a webhook to have Cisco Webex Teams notify you of new messages posted in a particular room.

Consuming a Webhook

- In order to receive a notification from a webhook provider, the application must meet certain requirements:
  - The application must be running at all times to receive HTTP POST requests.
  - The application must register a URI on the webhook provider.
- Also, the application must handle the incoming notifications from the webhook server.
- Use free online tools to ensure the application is receiving notifications from a webhook.

4.8 Troubleshooting API Calls

Troubleshooting REST API Requests

- There will be instances where you will make an API request but will not get the expected response. Hence, learning to troubleshoot the most common REST API issues is important.
- Always have the API reference guide and API authentication information handy while troubleshooting the REST API issues.

Troubleshooting API Calls

No Response and HTTP Status Code from the API server

- Sometimes API servers cannot be reached or fail to respond. You can identify what went wrong from the error messages received as a result of the request.

Troubleshooting tips for client side error:

- **User error**: Mistyping the URI when using the API for the first time.
- **Invalid URI Example** – To test the invalid URI condition, run a script which makes the request to a URI that is missing the scheme.

```
import requests
uri = "sandboxdnac.cisco.com/dna/intent/api/v1/network-device"
resp = requests.get(uri, verify = False)
```

The traceback will be as follows:

```
....requests.exceptions.MissingSchema: Invalid URL 'sandboxdnac.cisco.com/dna/intent/api/v1/network-
device': No schema supplied. Perhaps you meant http://sandboxdnac.cisco.com/dna/intent/api/v1/network-
device?....
```

No Response and HTTP Status Code from the API server (Contd.)

Wrong Domain name Example – To test the wrong domain name condition, run a script which simply makes the request to a URI that has the wrong domain name.

```
import requests
url = "https://sandboxdnac123.cisco.com/dna/intent/api/v1/network-device"
resp = requests.get(url, verify = False)
```

The traceback will look like this:

```
....requests.exceptions.ConnectionError: HTTPSConnectionPool(host='sandboxdnac123.cisco.com', port=443):
Max retries exceeded with url: /dna/intent/api/v1/network-device (Caused by
NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 0x109541080>: Failed to
establish a new connection: [Errno 8] nodename nor servname provided, or not known'))....
```

No Response and HTTP Status Code from the API server (Contd.)

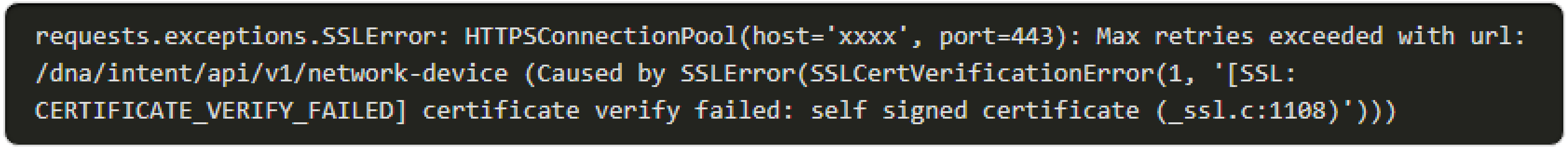
Connectivity Issues

- Are there any Proxy, Firewall or VPN issues?
- Is there an SSL error?

Invalid Certificate Example

- This issue can only occur if the REST API URI uses a secure connection (HTTPS).
- When the scheme of the URI is HTTPS, the connection will perform an the SSL handshake between the client and the server. If it fails, fix the invalid certificate

Traceback:



```
requests.exceptions.SSLError: HTTPSConnectionPool(host='xxxx', port=443): Max retries exceeded with url:
/dna/intent/api/v1/network-device (Caused by SSLError(SSLCertVerificationError(1, '[SSL:
CERTIFICATE_VERIFY_FAILED] certificate verify failed: self signed certificate (_ssl.c:1108)')))
```

No Response and HTTP Status Code from the API server (Contd.)

- But, if you are working in a lab environment where the certificates aren't valid yet, you can turn off the certificate verification setting.
- To turn it off for the requests library in Python, add the verify parameter to the request.

```
resp = requests.get(url, verify = False)
```

Resolution:

- Fix the issue by identifying the root cause.

Troubleshooting API Calls

No Response and HTTP Status Code from the API server (Contd.)

Troubleshooting tips for server side error :

- **API server functioning –** poweroff, cabling issues, domain name change, network down.
- To test if the IP address is accessible, run a script which makes the request to the URL and waits for a response.

```
import requests
url = "https://123.123.123.123/dna/intent/api/v1/network-device"
resp = requests.get(url, verify = False)
```

- If the API server is not functioning, you will get a long silence followed by a traceback that will be as follows:

```
....requests.exceptions.ConnectionError: HTTPSConnectionPool(host='209.165.209.225', port=443): Max
retries exceeded with url: /dna/intent/api/v1/network-device (Caused by
NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 0x10502fe20>: Failed to
establish a new connection: [Errno 60] Operation timed out'))....
```

No Response and HTTP Status Code from the API server (Contd.)

Is there a communication issue between the API server and the client?

- Use a network capturing tool to see if the response from the API server is lost in the communication between the API server and the client..
- If you have access, take a look at the API server logs if the request was received.

Resolution:

- Server side issues cannot be resolved from the API client side.
- Contact the administrator of the API server to resolve this issue.

Interpreting Status Codes

- The status code is a part of HTTP/1.1 standard (RFC 7231), where the first digit defines the class of the response and the last two digits do not have any class or categorization role.
- The five categories of status codes are as follows:
  - **1xx: Informational** - Request received, continuing to process.
  - **2xx: Success** - The action was successfully received, understood, and accepted.
  - **3xx: Redirection** - Further action must be taken in order to complete the request.
  - **4xx: Client Error** - The request contains bad syntax or cannot be fulfilled.
  - **5xx: Server Error** - The server failed to fulfill an apparently valid request.
- **Steps to troubleshoot errors:**
  - **Check the return code** - It can help to output the return code in the script during the development phase.
  - **Check the response body** - Output the response body during development
  - **Use status code reference** – If the issues cannot be resolved by checking the return code and response body.

2xx and 4xx Status Codes

- **2xx – Success error**: Successfully received, understood and accepted
- **4xx – Client side error**: Error is on the client side.

Troubleshooting common 4xx errors:

400 – Bad request

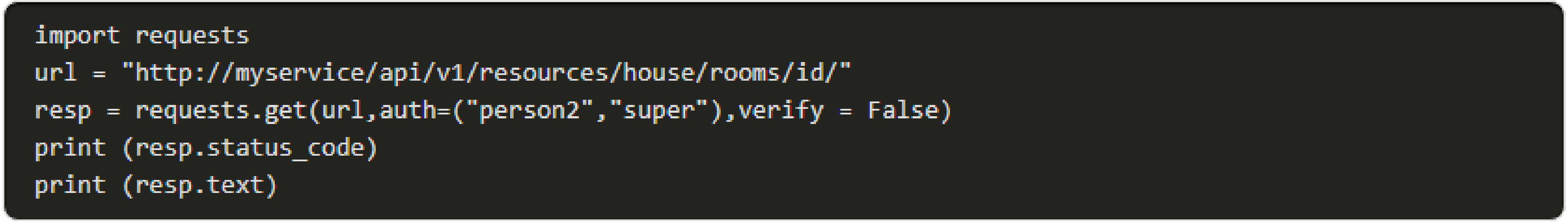
The request could not be understood by the server due to malformed syntax, which is mainly due to:

- Misspelling of resources
- Syntax issue in JSON object.

Troubleshooting API Calls

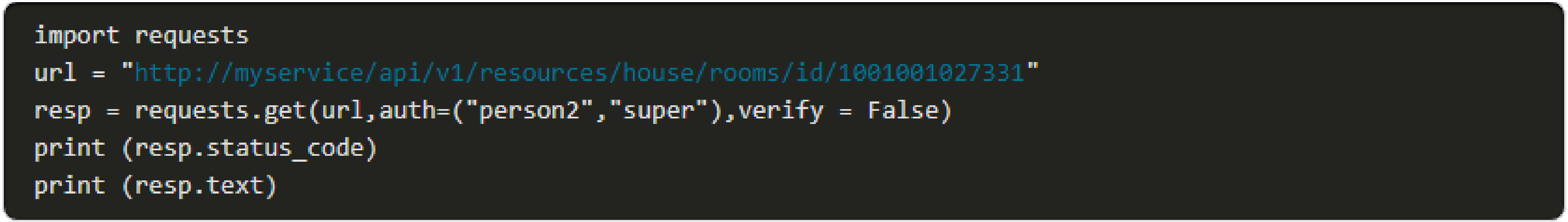
2xx and 4xx Status Codes (Contd.)

Example : This example returns a status code of 400.



```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/"
resp = requests.get(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

The server side also tells you "No id field provided", because the id is mandatory for this API request.



```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1001001027331"
resp = requests.get(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

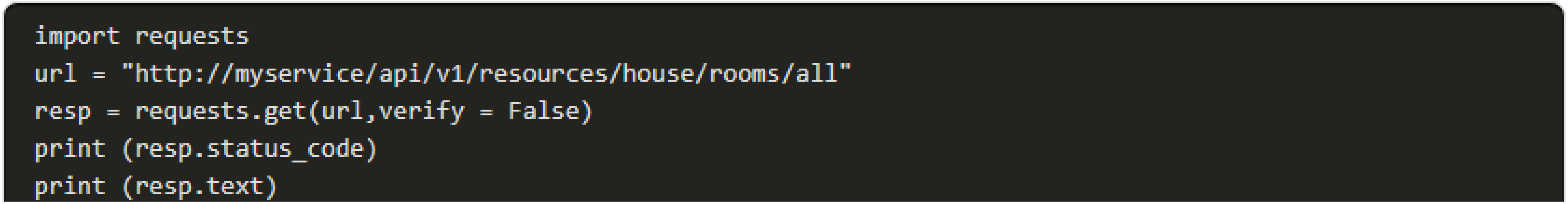
Troubleshooting API Calls

2xx and 4xx Status Codes (Contd.)

401 – Unauthorized:

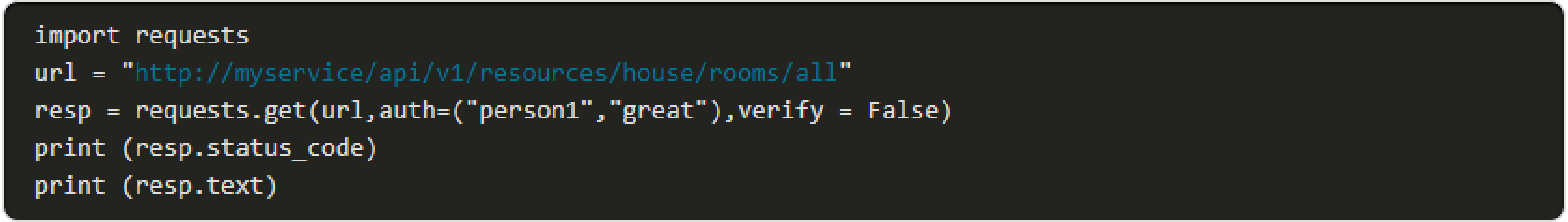
- This error message means the server could not authenticate the request.
- Check your credentials, including username, password, API key, token, request URI

Example



```
import requests
url = "http://myservice/api/v1/resources/house/rooms/all"
resp = requests.get(url,verify = False)
print (resp.status_code)
print (resp.text)
```

The authentication auth=("person1","great") should be added in the code



```
import requests
url = "http://myservice/api/v1/resources/house/rooms/all"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

Troubleshooting API Calls

2xx and 4xx Status Codes (Contd.)

403 – Forbidden

- The server recognizes the authentication credentials, but the client is not authorized to perform the request.
- **Example:** The status code 403 is not an authentication issue; it is just that the user does not have enough privileges to use that particular API.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,auth=("person1","great"),verify = False)
print (resp.status_code)
print (resp.text)
```

The authentication should be modified to use person2/super instead of person1/great.

```
import requests
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

Troubleshooting API Calls

2xx and 4xx Status Codes (Contd.)

407 - Proxy Authentication Required

- This code is similar to 401 (Unauthorized), but it indicates that the client must first authenticate itself with the proxy.
- In this scenario, there is a proxy server between the client and the server, and the 407 response code indicates that the client needs to authenticate with the proxy server first.

409 - The request could not be completed due to a conflict with the current state of the target resource.

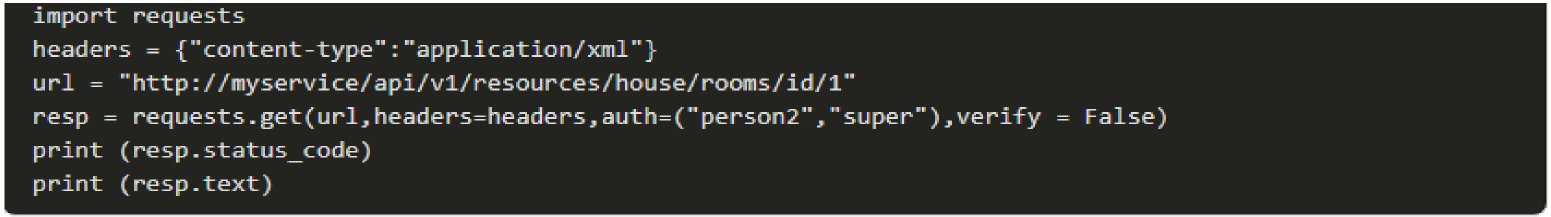
- For example, an edit conflict where a resource is being edited by multiple users would cause a 409 error.
- Retrying the request later might succeed, as long as the conflict is resolved by the server.

Troubleshooting API Calls

2xx and 4xx Status Codes (Contd.)

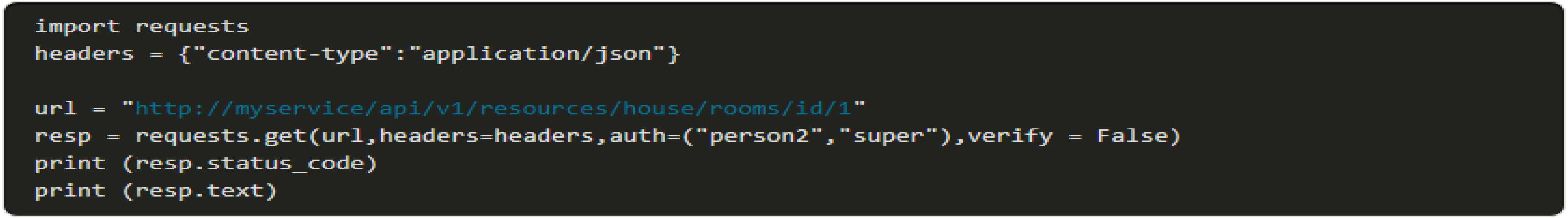
415 - Unsupported Media Type

- In this case, the client sent a request body in a format that the server does not support.
- **Example:** If the client sends XML to a server that only accepts JSON, the server would return a 415 error.



```
import requests
headers = {"content-type":"application/xml"}
url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,headers=headers,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

Omitting the header or adding a header **{"content-type":"application/ json"}** will work.



```
import requests
headers = {"content-type":"application/json"}

url = "http://myservice/api/v1/resources/house/rooms/id/1"
resp = requests.get(url,headers=headers,auth=("person2","super"),verify = False)
print (resp.status_code)
print (resp.text)
```

5xx Status Codes

- **500 - Internal Server Error**
  - The server encountered an unexpected condition that prevented it from fulfilling the request.
- **501 - Not Implemented**
  - The server does not support the functionality required to fulfill this request
- **502 - Bad Gateway**
  - The server (acting as gateway or proxy) received an invalid response from an inbound server.
- **503 - Service Unavailable**
  - The server is currently unable to handle the request due to overload or scheduled maintenance.
- **504 - Gateway Timeout**
  - The server (acting as a gateway or proxy) did not receive timely response from an upstream server.

4.9 Understanding and Using APIs Summary

What Did I Learn in this Module?

- API defines the ways users, developers, and other applications can interact with an application's components.
- An API can use common web-based interactions or communication protocols and its own proprietary standards.
- APIs can be delivered synchronously (or) asynchronously.
- The three most popular types of API architectural styles are RPC, SOAP, and REST.
- A REST web service API (REST API) is a programming interface that communicates over HTTP while adhering to the principles of the REST architectural style.
- Authentication is the act of verifying the user's identity. Common types of authentication mechanisms include Basic, Bearer, and API Key.
- Authorization is the act where the user is proving to have permissions to perform the requested action on that resource.

What Did I Learn in this Module? (Contd.)

- An API Rate limit is a way for a web service to control the number of requests a user or an application can make per defined unit of time.
- A webhook is an HTTP callback, or an HTTP POST, to a specified URL that notifies your application in case of an activity or 'event' in one of your resources on the platform.
- The API reference guide and API authentication information must be handy before troubleshooting.
- Client side errors include user error, wrong URI, wrong domain, a connectivity issue, and an invalid certificate.
- Server side error includes communication problems between the server and the client.
- 4xx codes are Client side errors and 5xx codes are Server side errors.

Lab – Integrating a REST API with Python

In this lab, you will complete the following objectives:

- **Part 1**: Launch the DEVASC VM
- **Part 2**: Demonstrate the MapQuest Directions Application
- **Part 3**: Get a MapQuest API Key
- **Part 4**: Build the Basic MapQuest Direction Application
- **Part 5**: Upgrade the MapQuest Directions Application with More Features
- **Part 6**: Test Full Application Functionality