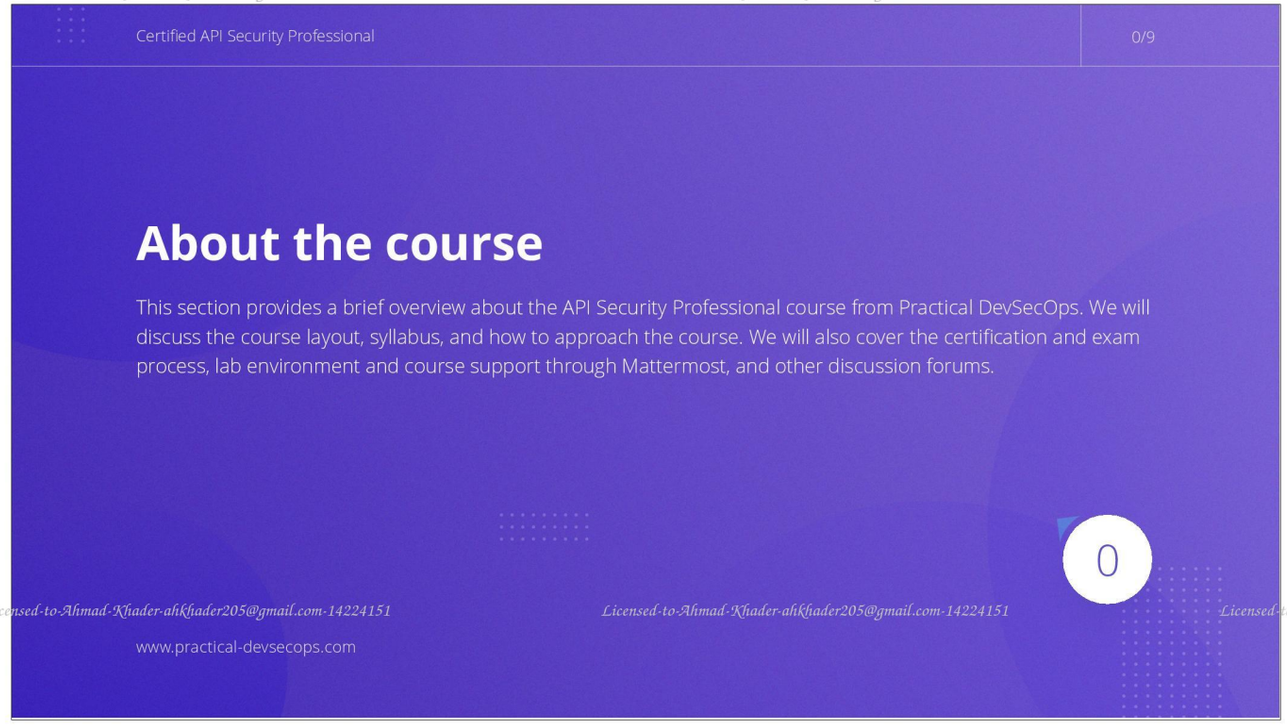
Hi, Welcome to the Certified API Security Professional course, we are very glad to be assisting in your journey of API security.



This section provides a brief overview about the API Security Professional course from Practical DevSecOps. We will discuss the course layout, syllabus, and how to approach the course. We will also cover the certification and exam process, lab environment and course support through Mattermost, and other discussion forums.

API Security Professional (CASP) Course

Welcome to the world's most comprehensive API Security course. By the end of this course you will be able to identify, exploit, and protect against a wide variety of API security vulnerabilities.

You will start the course with API basics, core components of API architecture, and ways to interact with the APIs. Once you learn the fundamentals, you will gain hands-on experience with a series of realistic attack scenarios like Server Side Request Forgery, Broken User Authentication, Broken Access Control issues, Injection attacks, Privilege escalation, and Security misconfigurations. Finally, you will also learn how to fortify your APIs and make them resilient to cyber attacks.

- Introduction to API Security
- API Security tools of the trade
- Authentication Attacks and Defenses
- Authorization Attacks and Defenses
- Input Validation Threats and Defenses
- Other API Security Threats
- Other API Security Defenses
- Implementing Security Defenses
- API Security, the DevSecOps Way

Welcome to the world's most comprehensive API Security course. By the end of this course you will be able to identify, exploit, and protect against a wide variety of API security vulnerabilities.

You will start the course with API basics, core components of API architecture, and ways to interact with the APIs. Once you learn the fundamentals, you will gain hands-on experience with a series of realistic attack scenarios like Server Side Request Forgery, Broken Authentication, Broken Access Control issues, Injection attacks, Privilege escalation, and Security misconfigurations. Finally, you will also learn how to fortify your APIs and make them resilient to cyber attacks.

1. Introduction to API Security
2. API Security tools of the trade
3. Authentication Attacks and Defenses
4. Authorization Attacks and Defenses
5. Input Validation Threats and Defenses
6. Other API Security Threats
7. Other API Security Defenses
8. Implementing Security Defenses
9. API Security, the DevSecOps Way

Course Inclusions

The Certified API Security Professional course includes the following:

1. Course videos
2. Course slide handouts
3. Lab demonstrations
4. 60 days of hands-on lab access
5. Mattermost channel access
6. A single CASP certification attempt

Course Portal and Course Manual

Video Lectures

Demos/Quizes

Checklists/Handouts

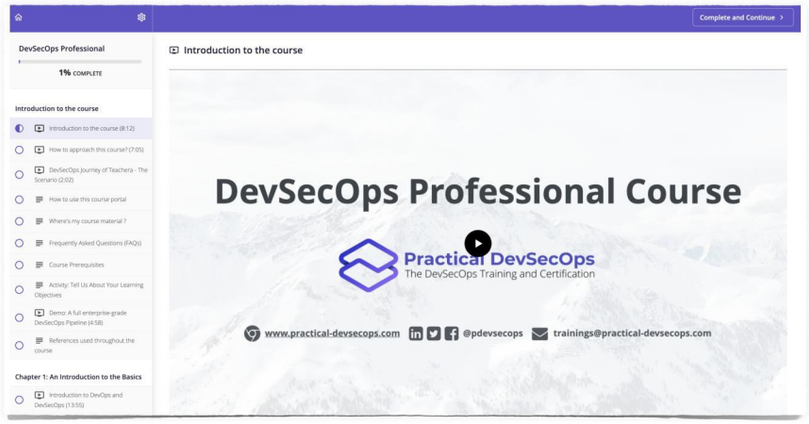
The lectures are delivered in video format on our course portal. The course portal also includes various checklists, infographics and other resources.

The API Security Professional course portal which contains videos, checklists, etc. is your go-to resource for learning API security techniques, tools, and processes. You can access the course videos in the course portal.

The course has 9 modules. Each module covers the necessary lecture, and showcases the demonstrations that are relevant for that module. We also have quizzes that test your skills and help in knowledge retention.

If you prefer to read the content, please access the slide handouts. However, do note that the course portal remains the single source of truth.

Course Portal Setup



A sample course portal image is shown here.

Lab setup and exercises

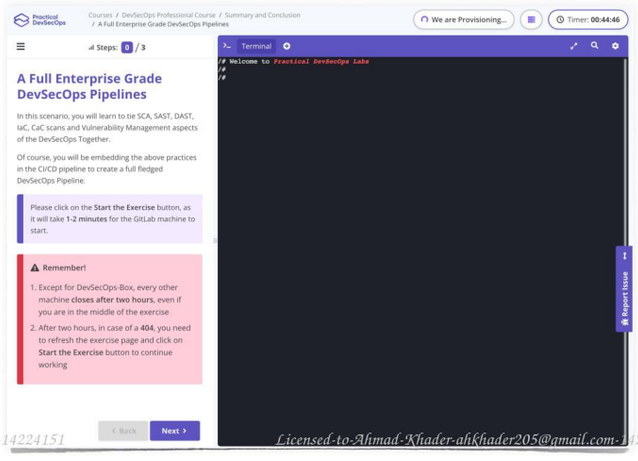
- Labs are how we engage with out clients, labs are real world implementations
- Labs are are taught with a combination of open source and commercial tooling
- Labs focus on teaching techniques that apply to any tool or a domain, or an industry

Labs are how we engage with our clients, these are not simulated exercises but real world implementations.

This class teaches advanced API Security techniques with many open source tools but some of techniques and methods might not apply to you because you are in another domain or industry. Thats okay, as techniques will be the same for other technologies and domains.

We will setup Lab after the introduction.

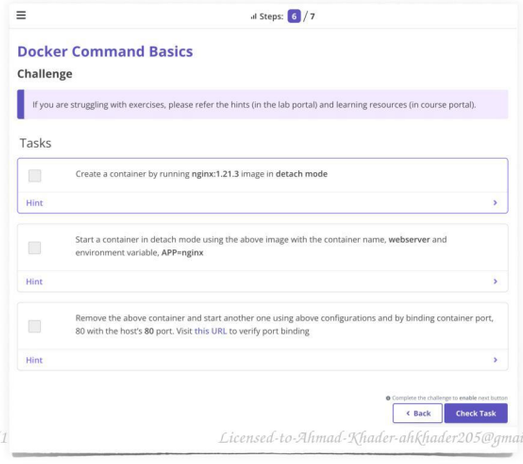
Lab environment



Here is a screenshot of the lab environment. The UI will change from time to time however the core concept remains the same.

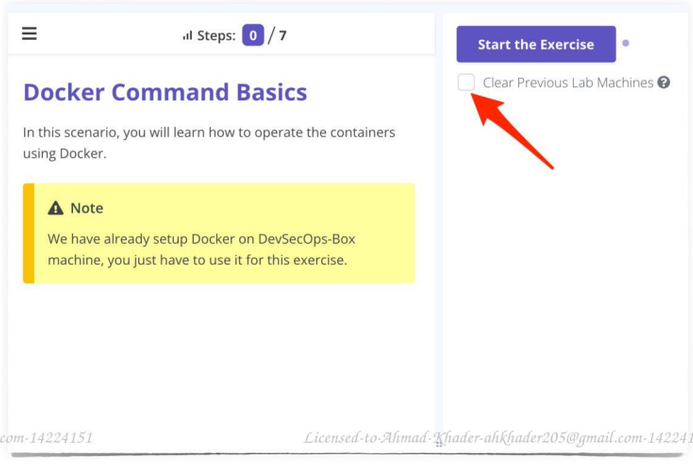
You will have instructions on the left and a terminal on the right.

Lab challenges



After each topic, we have numerous exercises that will help you practice what you learned in the live environment, right in your browser.

Lab challenges



You can access the lab environment anywhere, anytime, and as many times as you want. Since this course is practical and hands-on, you will spend 80% on the lab and 20% on the fundamentals.

Communicating with Instructors

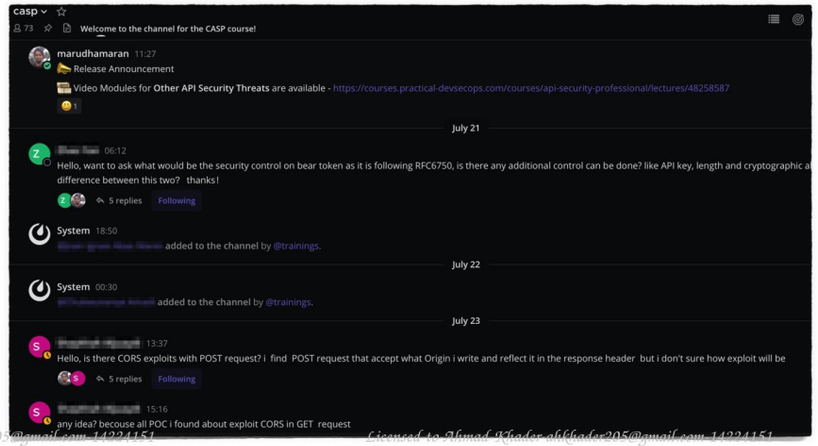
As a humble note, and reminder, please follow the lab portal and re-visit the steps again if you face any issues.

We are committed to help our students. If you face any issues with lab connectivity or just want to bounce your ideas around, please reach out to us via the following modes of the communication:

1. We use Mattermost as our preferred chat platform and someone from our team is usually available on Mattermost to help you with your queries.
2. You can email us on trainings@practical-devsecops.com.

Mattermost is usually a quicker way to get in touch with us.

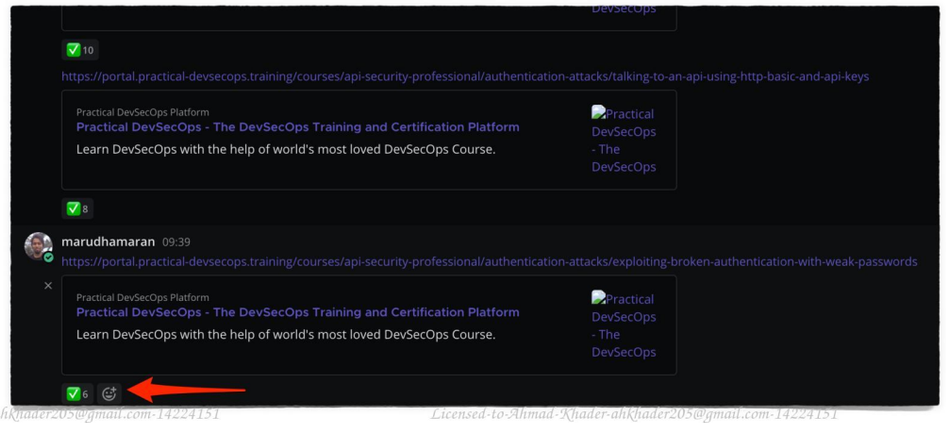
Support on Mattermost



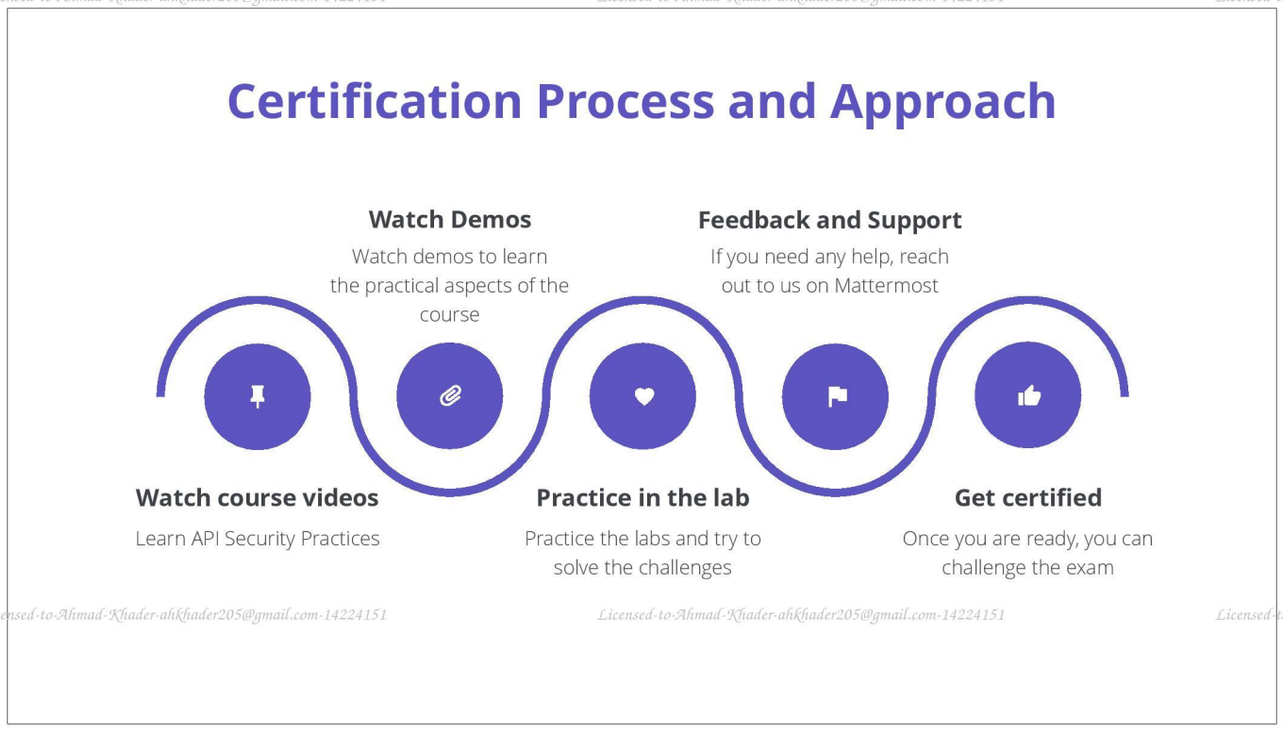
Here is a screenshot a typical Mattermost communication with our staff. Please choose appropriate channel for your course, refer the course portal for more information.

For the API Security professional course, there is a dedicated channel named casp that is intended for support.

Conversation With Instructors



This is a screenshot a typical Mattermost communication with out staff. Please choose appropriate channel for your course, refer the course portal for more information.



The course portal contains necessary links to the Mattermost channel, lab portal, and many other learning references.

The following is the recommended approach to the course:

1. Watch the videos to learn the concepts, principles, and practices.
2. Watch demonstration videos if there are any.
3. Pick the relevant exercises in the lab and complete the mandatory exercises.
4. If you need any help, please reach out to us on Mattermost using your course's dedicated channel.
5. Once you are ready, you can schedule your exam.

Course Pre-requisites

Course participants should have a basic understanding of Linux Commands and OWASP Top 10.

Basic knowledge of application development is preferred but is not necessary.

This course doesn't have any pre-requisites except basic understanding of OWASP Top 10 vulnerabilities.

However, a basic understanding of Linux Command Line Interface would help you work through the lab exercises, and familiarity with software development, or IT systems would help you in understanding API Security better, through it is not a pre-requisite.

You do not need any prior experience with Linux, DevOps tools or APIs to take this course.

Learning Objectives

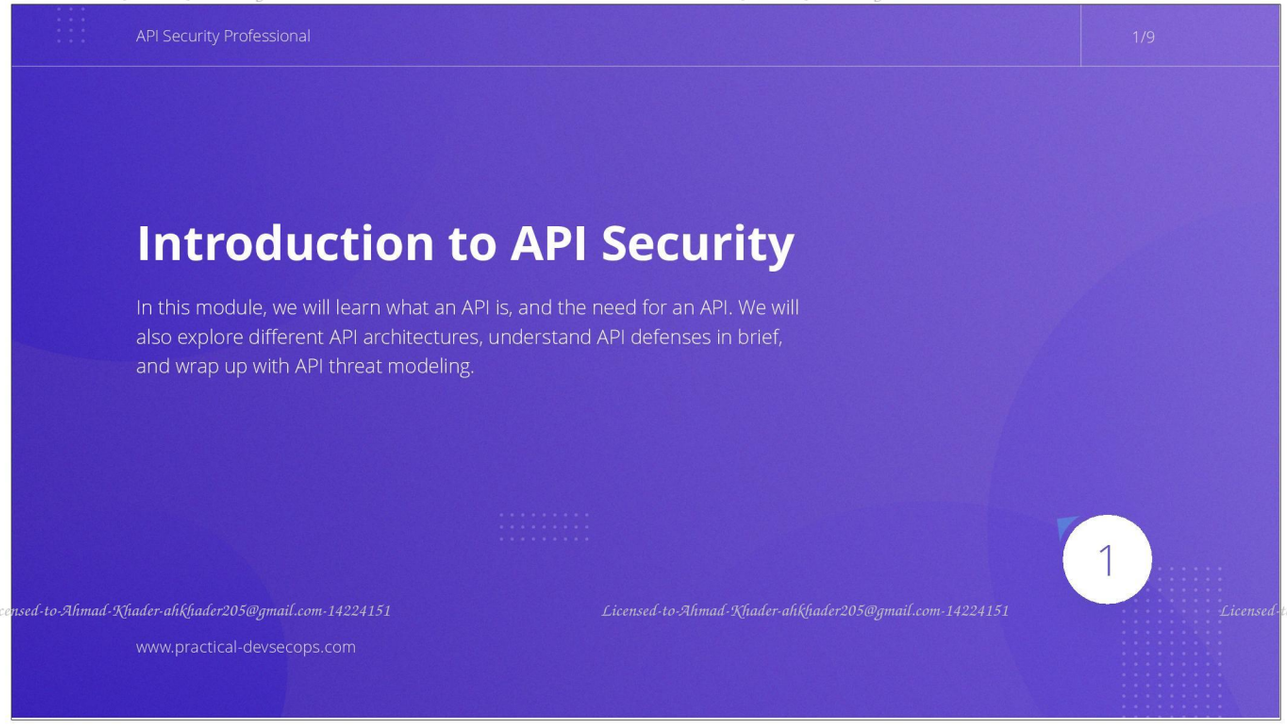
1. Identify, exploit, and protect against a wide variety of API security vulnerabilities.
2. Gain a practical understanding of API Security and the tools to automate it.
3. Build a solid foundation that is required to understand API Security Risks and Mitigations.
6. Understand and implement the modern ways of scaling API Security Testing.
7. Use the DevSecOps best practices to find and fix API security flaws early and often.

By the end of this course, you will learn:

- Identify, exploit, and protect against a wide variety of API security vulnerabilities.
- Gain a practical understanding of API Security and the tools to automate it.
- Build a solid foundation that is required to understand API Security Risks and Mitigations.
- Understand and implement the modern ways of scaling API Security Testing.
- Use the DevSecOps best practices to find and fix API security flaws early and often.

We look forward to assisting you in your API Security journey. See you soon!

Practical DevSecOps
The DevSecOps Training & Certification



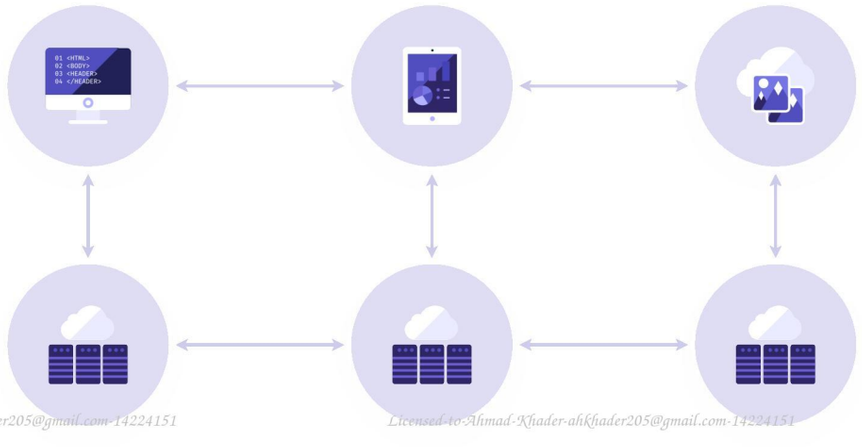
In this module, we will learn what an API is, and the need for an API. We will also explore different API architectures, understand API defenses in brief, and wrap up with API threat modeling.

Introduction to an API

What is an API, why we need it and why should we secure them

In this section, we will cover, what is an API, why we need it and why should we secure them.

Application Programming Interface (API)

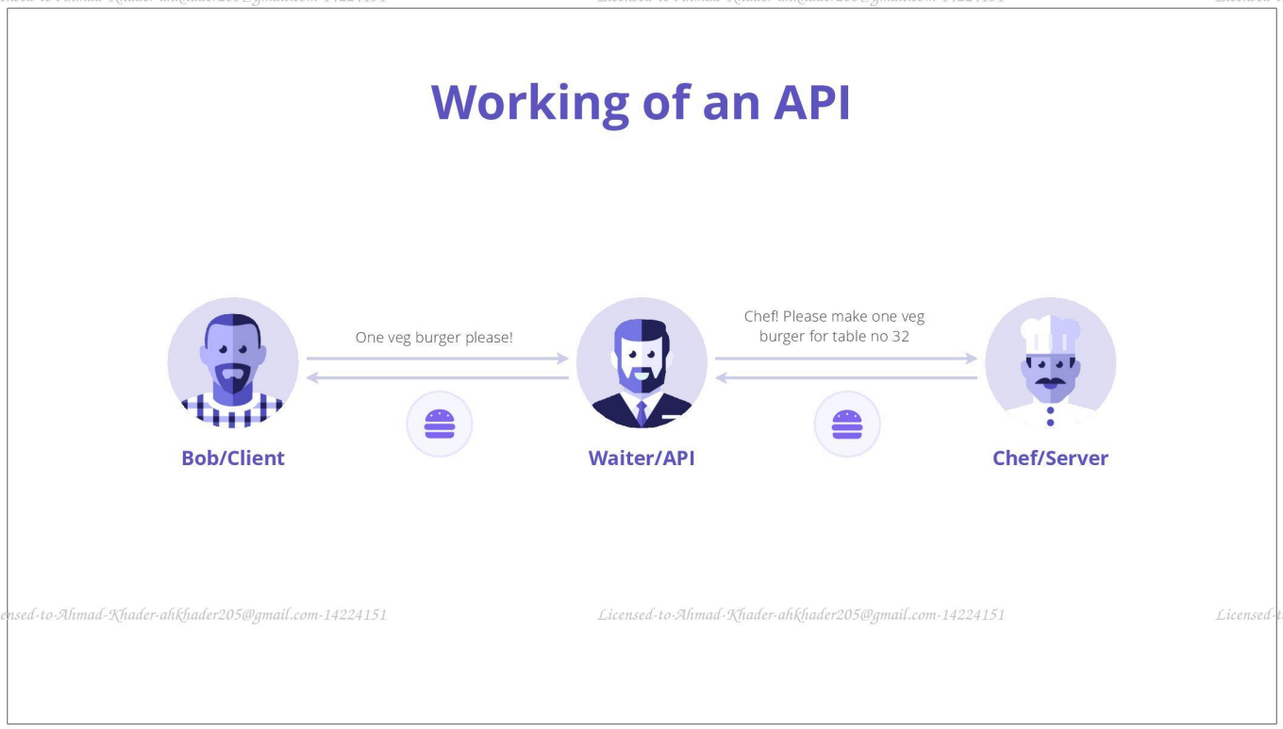


Application Programming Interface or an API is a set of rules or definitions that helps build and integrate applications.

API helps your application communicate with other applications. Applications in your day-to-day life use API for communication without you realizing it because they are so abstracted from the user, and they are really behind the scenes mechanism that provides for today's connected applications.

Let us understand the inner workings of an API through an example.

0



Bob goes to a restaurant and asks the waiter to bring one vegetarian burger.

The waiter goes to the chef and says, “Chef! Please make one vegetarian burger for table no 32”.

The chef understands the request from the waiter, the chef prepares the food and gives it back to the waiter. The waiter carries the food and serves Bob.

Let us use this analogy to understand how an API works.

Well, let us replace Bob with the client.
Let us replace the waiter with an API.
Let us replace the chef with the server.

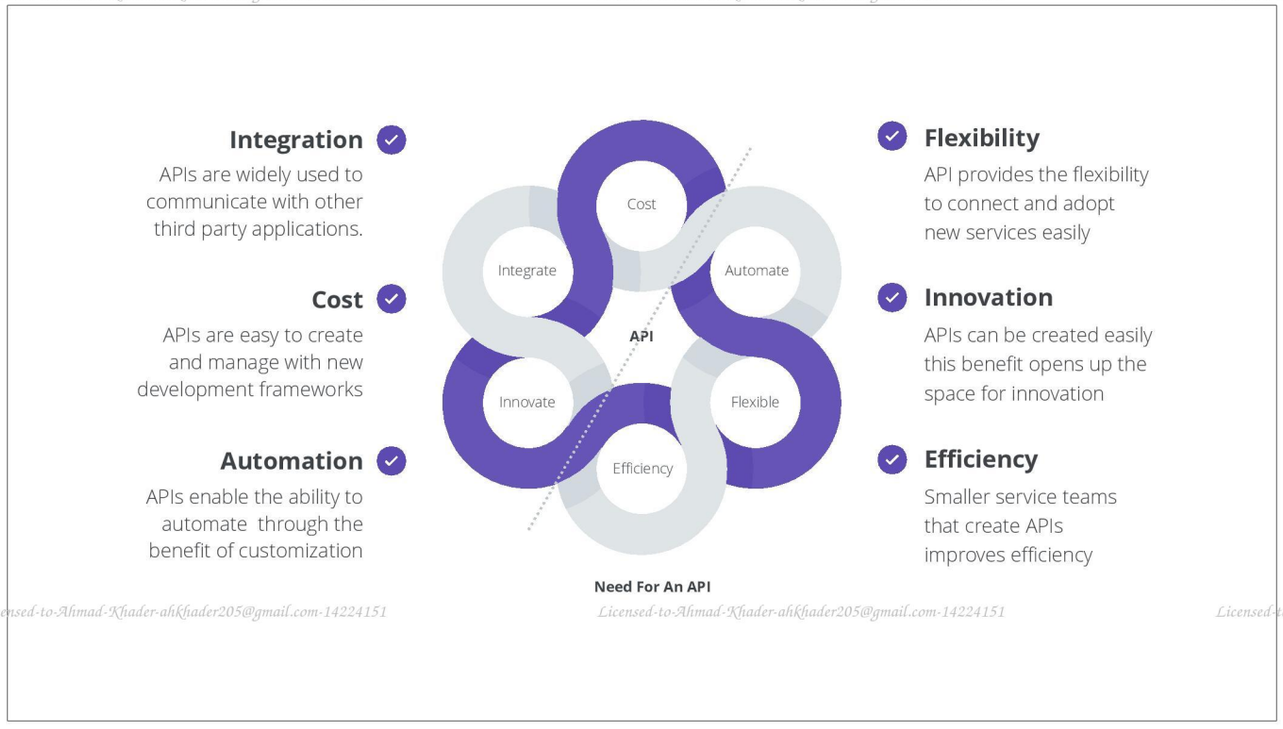
So, a client requests information from the server using the API, just like how Bob, the customer, ordered a vegetarian burger through the water, and how the vegetarian burger was finally prepared by the chef, and served to the customer Bob, through a waiter.

In the world of APIs, the server understands the request from a client and processes the request.

Once the processing is done, the server gives back the response to the client using an API, just like how the chef delivered the food, to Bob, through a waiter.

In real world, there are many other processes involved in the client-server model, like a load balancer, a reverse proxy, Content Delivery Networks or CDNs, API gateways, service busses, and many other components.

However, Do we really need an API?



Need For An API

Customers desire to buy products or services from innovative companies, and APIs behind the scenes fuel innovations to help bring products and services quickly to customers.

As APIs are easy to customize, companies can listen to the voice of the customers and enhance their products. Also, businesses can adopt these APIs into their products to provide better services.

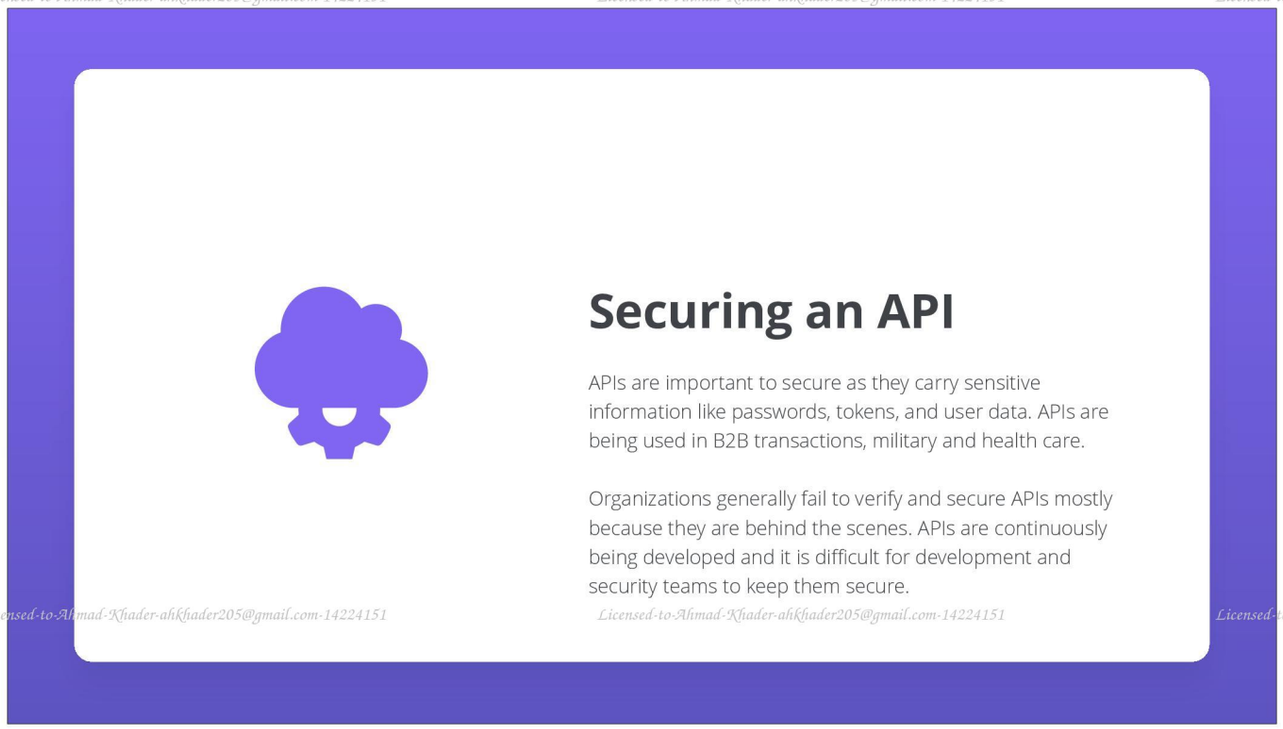
APIs open up the space for easy development and innovation. Developers can invest their time building new products for customers by reusing existing components through their APIs.

According to a survey from a company called Formotus, the average cost of deployment is $270,000. APIs are easy to create and it can prove to be a cost effective solution. One of the important benefits of APIs is they can be used and integrated easily.

APIs can be used for automated testing, deployment, and much more. Organizations have small teams for creating services using APIs, this helps to improve the collaboration within the team and improves work efficiency.

APIs are not only used within the organization but APIs can be made open to the world too. This helps organizations to create new business ideas and to achieve their vision.

If APIs are a vantage point in any business, let’s explore the need to secure an API.



Remember we said, APIs are one of the backbones to signify the importance of an API in today's connected world?

Cybercriminals try to take over all possible technologies from networks to hardware. A huge number of APIs are being created daily which opens up the space for vulnerabilities.

APIs are important to secure as they carry sensitive information like passwords, tokens, and critical user data. APIs are being used in B2B transactions, military, space technology, healthcare, and everywhere else too.

APIs are open to the world and anyone can attack an organization using APIs, among other entry points of attack. Even an improper API can lead to mass data leakage as reported in many data breaches.

Organizations generally fail to verify and secure APIs mostly because they are behind the scenes. APIs are continuously being developed and it is difficult for development and security teams to keep them secure.

Implementing strong security processes will help an organization become secure. This might sound hard for some but it sounds better than an organization losing millions of money in the form of penalties and loosing trust in customers which can be unquantifiable.

Client vs Server

Client

Client is a place from where the request is initiated

Server

A server processes requests from the client and responds

Let’s quickly explore the difference between a client and a server to help us work with APIs.

Client is a place from where the request is initiated. This request can be initiated using scripts, or tools, or GUI based application.

The client sends a request to the server and waits for the response from the server.

Server is a place where the requests from the client are obtained and used for further processes.

The server processes the user input from the request, handles the business logic, works with third-party applications to serve an intended purpose.

Web Application and APIs

Web applications	APIs
• Can be accessed from browser, command line • Human to system communication. • Does not require a computer savvy user. • What a user sees as a face of a product or a company • Does not necessarily need an API behind the scenes to work	• Can be accessed using request initiators. • System to system communication. • Need to have basic network or protocol knowledge. • What the user does not see as a face of a product or a company • APIs need a front end to actually serve their intended purpose

A web application is a software that is running on a remote server and can be interacted with using a web browser through the HTTP protocol.

The web used to be static content, called websites, but now web applications can serve the purposes of storage, administration, analytics, and anything else that was traditionally offered by desktop based systems.

APIs are used by the web application to communicate with the server.

To interact with the web application, the user doesn't need any technical knowledge whereas, in case of an API, the user needs to have a minimum understanding of how the HTTP protocol works.

These APIs are designed for application-to-application communication whereas web application is designed for humans to work with remote computer systems.

In the next section, we will explore API architectures.

Practical DevSecOps
The DevSecOps Training & Certification

Understanding API Architecture

HTTP protocol, stateful and stateless, API architectures

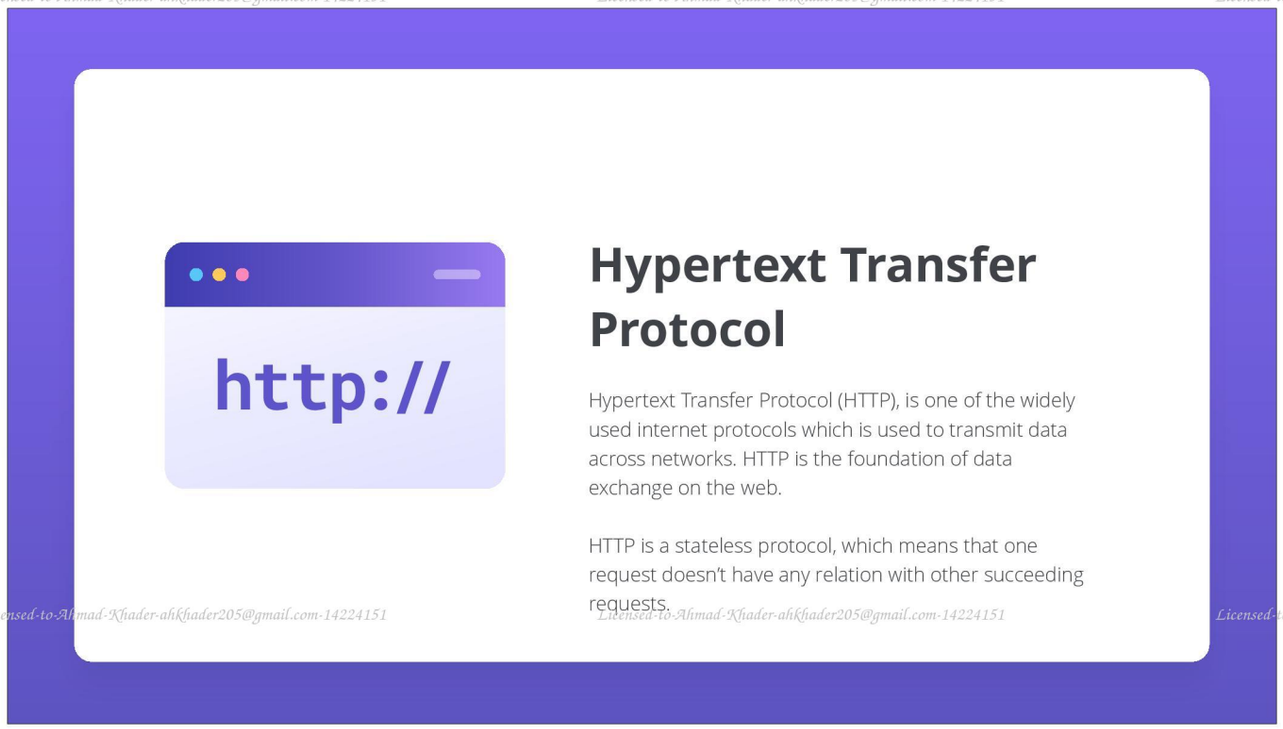
2

In this section, we will explore the HTTP protocol, types of API architectures and understand the differences between stateful and stateless protocol.

Overview of HTTP Protocol

In this sub-section, we will learn what HTTP is, how it works and helps in making a API requests.

In this sub-section, we will learn what HTTP is, how it works and helps in making a API requests.



Hypertext Transfer Protocol (HTTP), is one of the widely used internet protocols which is used to transmit data across networks. HTTP is the foundation of data exchange on the web.

APIs use HTTP for transmitting data. The current version of the internet uses HTTPS which is a secure version of HTTP.

HTTP is a stateless protocol, which means that one request doesn't have any relation with other succeeding requests.

The stateless behavior served well for static web pages in the initial days of HTTP, however to build dynamic applications that can remember an authenticated user, cookies and other mechanisms were introduced to make HTTP remember states.

10,000 Feet View of HTTP Communication

Let’s start with a ten thousand feet view of a HTTP communication.
When the client needs information, it sends a HTTP request to the server and the server process the request to return an HTTP response which then the client interprets for its use.

HTTP request and response have different parts like methods or verbs, headers, URI, and body. Let’s explore the anatomy of a HTTP request and response in the upcoming slides.

Let’s start by exploring the HTTP request first.

In the client-server model, the client sends a request to the server with various information like protocol, methods, headers and body. It might be wise to explicitly call them as a request protocol, request method, request headers, and request body as the HTTP response would also have some similar elements.

Here's a first HTTP Request with no request body.

All the information presented here is important for the server to understand the request and do further processing.

In this example HTTP request, all we see are HTTP request headers, and there is no HTTP request body.

The first line or the Request Line says *GET* (pronounced as get) as the type of HTTP verb, */search.php?q=Taj* as the resource that is being requested, and *HTTP/1.1* as the name and version of the HTTP protocol itself.

Everything after the first line of request are simply key-value pairs.

For example, in the second line, the key *Host* has a value *www.myhotels.com* which means the server that should serve this request is the host identified by *www.myhotels.com*.

The next line is the key *Accept* which has a long list of values indicating the content types that are understandable by the client.

Then there is Accept-Language, Accept-Encoding, and User-Agent as example request headers.

Request headers are used to send information like authentication tokens, length of the body, host, cookies and many other information.

This example request has no request body as pointed out by the Empty line. In a HTTP request the request headers, and the request body is separated through a newline indicated by a carriage-return, and a new line character, that is the \r\n which is also a point of abuse for many HTTP attacks.

The example request here does request a specific set of data. Could you guess what that is?

HTTP Request #1

The Request line requests a specific resource in the form of */search.php?q=Taj*.

search.php is the resource of the web page that is being requested by the GET verb, and more specifically we are querying for a hotel named *Taj* as indicated by *q=Taj*.

In terms of HTTP requests that do not have a request body, data can be passed through URLs.

Here's a second HTTP request that has a noticeable new verb called POST (pronounced as post), and it also has a HTTP Request body.

Looking at the first line of the HTTP request we can see that the HTTP verb is *POST,* and we are posting information to the resource at */hotels/search.php* to be requested at the *Host* identified by the domain name *www.myhotes.com.*

Among all the other familiar HTTP request headers that we have seen in the previous example, a the every end of this example, there is *q=Taj* which is a HTTP request body.

According to the HTTP semantics, if the HTTP request verb is POST, then the HTTP request needs to have a request body.

The request body in this example *q=Taj* is also in a key value pair format where the key is the letter *q* and the value is *Taj*.

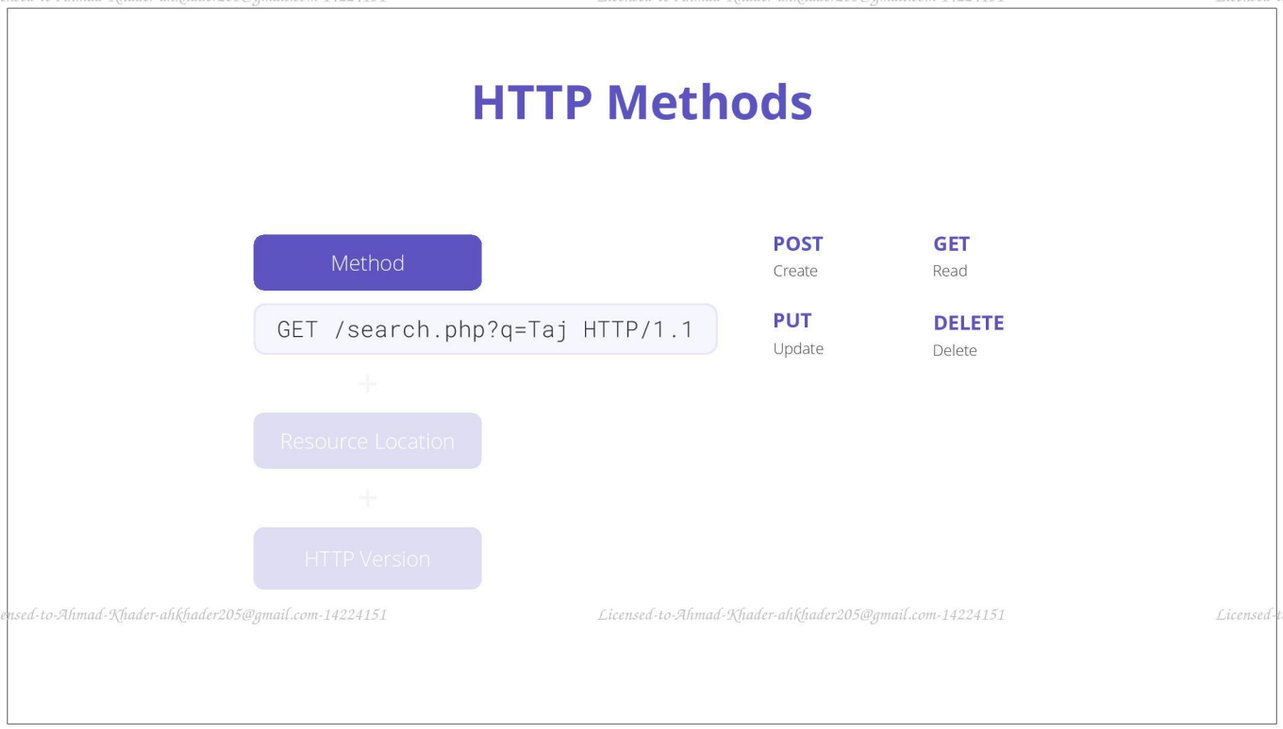
In practice, the request body can be of XML, JSON or any other data exchange formats.

The request also specifies a header called content type which helps the back-end application to identify the type of data, for example, application/JSON, application/XML, and application/x-www-urlencoded depending on the type of body.

All the HTTP request headers and body represented in this example are significant for security.

In the upcoming lessons, we will also learn many other HTTP request headers that help us in securing our APIs, but for now, let’s deep dive to understand HTTP requests and responses.

The HTTP request starts with a request line, which is a combination of a method, resource location, and the HTTP version followed up by a carriage return and a line feed (CRLF). Let us try to understand each of these components.



HTTP method help the server to understand what type of action that the client is trying to perform. HTTP methods are also called HTTP verbs as we have been referring to. There are nearly nine different types of HTTP methods according to the documentation at the Mozilla Developer Network or the MDN.

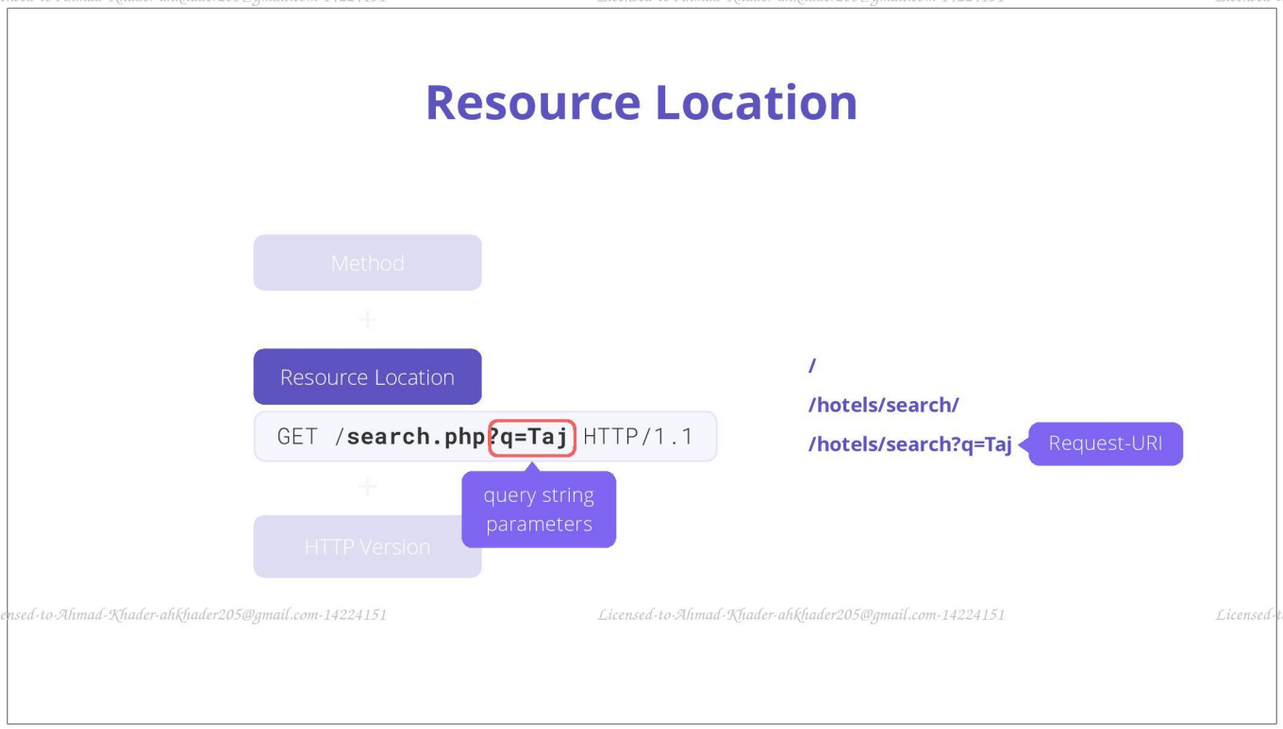
Let’s look at some of the widely used HTTP methods.

POST method is often used to perform write operations, that is when we want to post information to a server. For example, you would use post to save an email address, or perform a transaction, or to save any other data at the server.

GET method is used by the client to fetch data from the server. No write operation will be performed in the server side.

PUT method is used to create new resource at the server or perform an update operation. For example, PUT can be used to create a new file using File Upload.

DELETE method is used to delete a resource at the server.



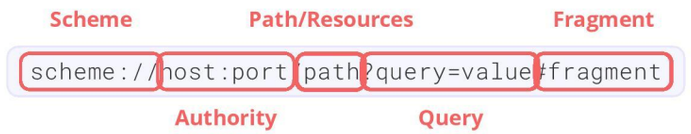
After the HTTP method, followed by a space is the resource location.

Resource location can be just a forward slash which is often used to retrieve the contents of a homepage of a web site.
The resource location can be a path which is used for navigating to different pages or APIs.

The resource location could also specify data or parameters to be passed the server. In the above example, *q=Taj* is the parameter that the client sends to the server. Anything in the resource location after the question mark is called query string parameters.

So, *q=Taj* is a query string parameter that is used to send data to the server as a part of the resource location. The resource location, that is */search.php?q=Taj* is semantically referred to as a Request-URI where URI stands for Uniform Resource Identifier.

URI Components (RFC 3896)



A URI starts with a scheme or protocol followed by authority, port number, query and fragment.

The scheme or protocol is where protocols like HTTP, FTP, or FILE are set.

The host is where the domain name comes with a subdomain. If there is no subdomain, then the subdomain is defaulted to www.
The host can also have a port number to specify to contact at the server at a specific port. If no port is mentioned, the connection is initiated at a default port based on the type of scheme or protocol. By default, port 80 is used for connecting through HTTP, and port 443 is used for connecting through HTTPS.

Next comes the query, where small amounts of data can be passed to the server.

The fragment is used for accessing the DOM. DOM expands as Document Object Model and DOM is a part of the browser clients, of clients that support web based document rendering.
DOM is often used to load pages or switch between content without refreshing the page, and like many other elements DOM is also an object of abuse by adversaries.

One important thing to note, the fragment is not sent when a browser client sends the request. Even if the browser client sends a request with a fragment to the server, the fragment would simply be discarded by the server, as fragment is a data that is used to manipulate the DOM, and DOM is part of a browser client.

URI Components (RFC 3896)

Here is an example of an URI with https as scheme, website.com as host, 8080 as port, /country/person as path, name as query parameter name, and bar as its value with address as fragment.

URI Components - Examples

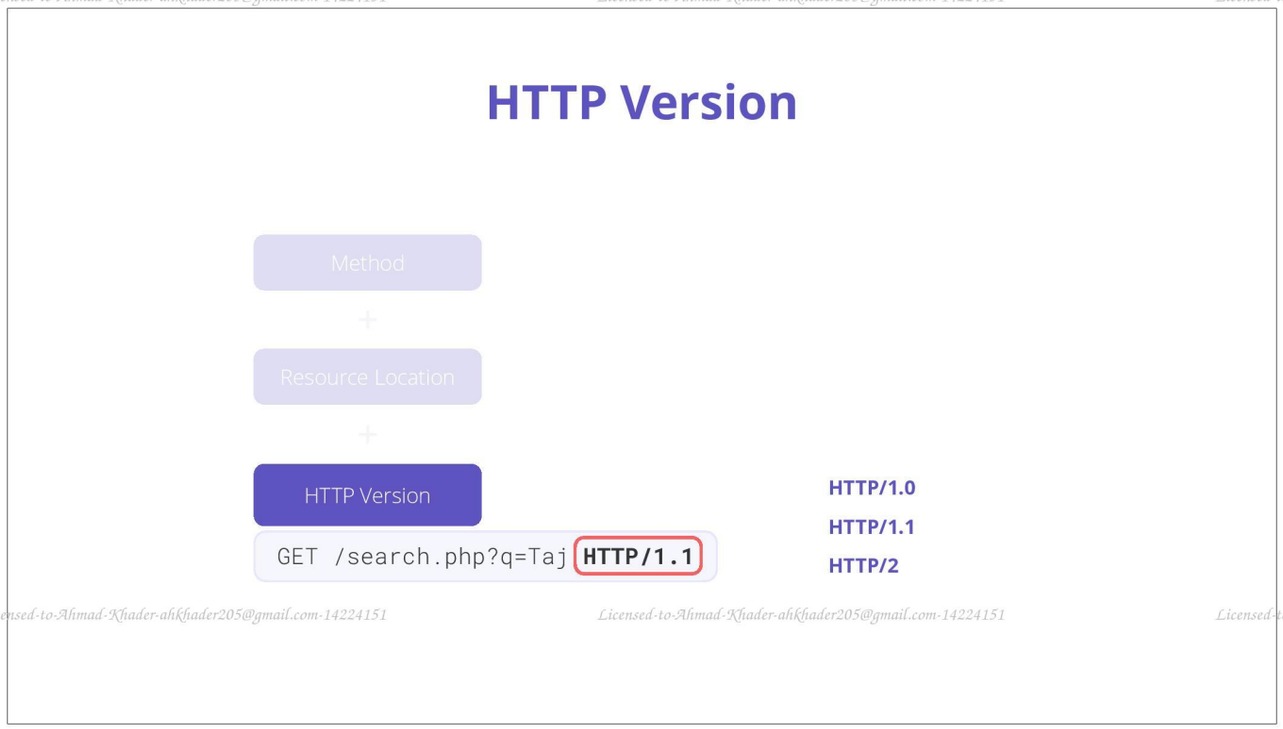
file:///etc/passwd
http://example.org
ftp://192.168.0.23
mailto:trainings@practical-devsecops.com
file://localhost/etc/passwd

Here are some other examples of URI with different schemes or protocols such as *file*, *http*, *ftp*, and *mailto*.

Let us take a look at some of these examples, *file* scheme with URI refers to the local file path. *file* scheme is used to access a file resource in the local machine.

ftp protocol with the FTP server's IP address would try and initiate a connection to the FTP server on a specific IP address, and *http* protocol with the domain name as example.org would initiate a HTTP connection to the server at *example.org*.

Similarly, *mailto* scheme would open mail clients with the information identified in its URI.



HTTP version indicates the HTTP protocol version used by the API.

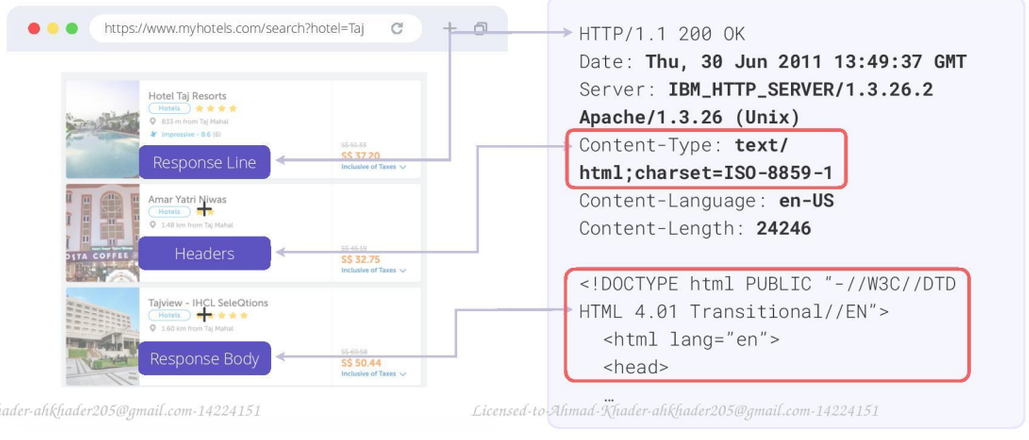
HTTP version is part of the request line of the HTTP header followed by the HTTP method, and the URI.

HTTP has three different versions, 1.0, 1.1, and 2.

1.0 is for HTTP, 1.1 is for HTTPS and 2 is for HTTPS 2.

Now that we have understood HTTP request, let’s move on the HTTP response next.

HTTP Response



Similar to the HTTP request, the HTTP response is sent in a structured way so that the browser or the client can understand and process the response. The HTTP response is very important in building client-side security.

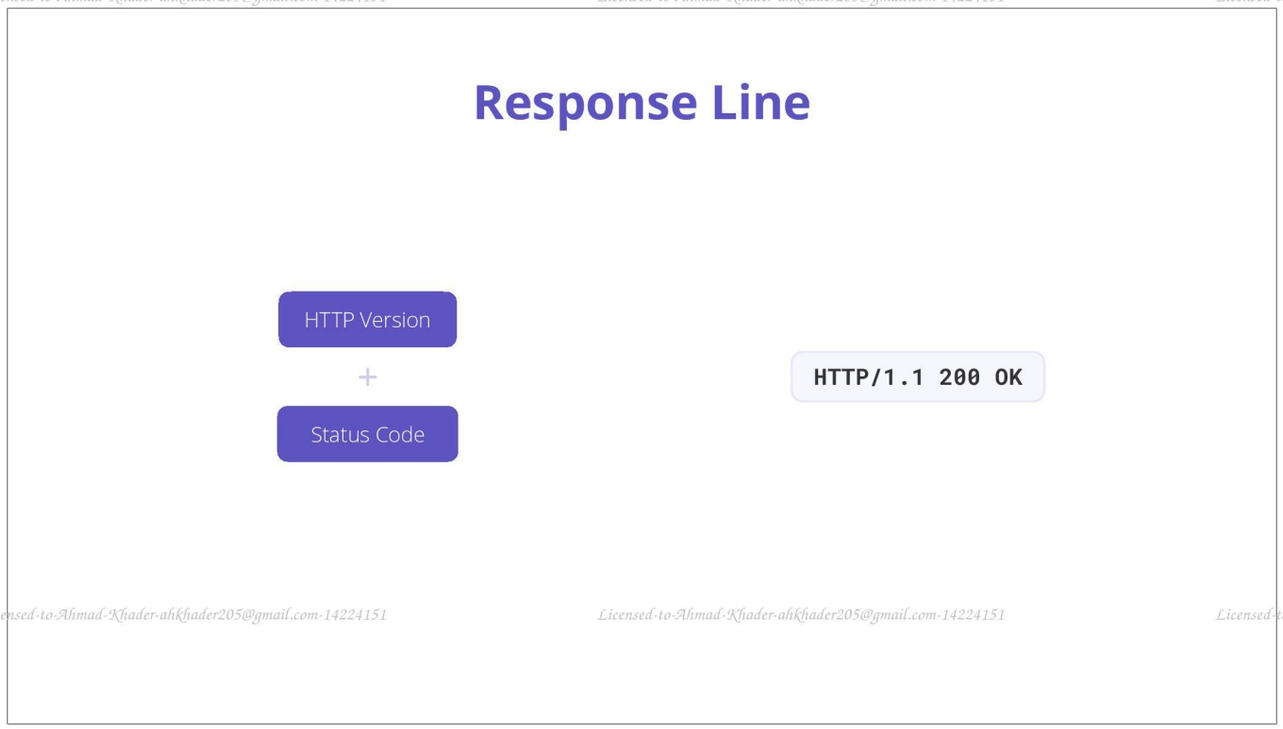
The response starts with a response line, a set of headers and the response body. We will explore the response line in the upcoming slides.

In the response body, do you notice that the the application has sent an HTML response to the client which can be used to render the HTML response as a formatted web page.

Well, how do we know the response body is an HTML response? We can simply look at the HTML tags in the response body, or the response body itself screams HTML at many places. We can do that as humans.

But the clients, that interpret the response body need a better or structured way of identifying the type of data in the HTTP response body. That's where the *Content-Type* header comes in with its value set to *text/html;charset=ISO-8859*-1.

If the *Content-Type* is set to *text/html*, the client expects, treats, and tries to parse the HTTP response body as HTML.
If the *Content-Type* is set to *application/json*, the client expects, treats, and tries to parse the HTTP response body as JSON.



The HTTP response line is a combination of the HTTP version and a response code.

The response code is also known as status code.

Response line helps the client to understand the protocol being used and the status of the HTTP request. The client uses the status code to identify if the HTTP request was interpreted well by the server.

There are many status codes, we will explore the commonly used status codes in the upcoming slides.

Response Code

1xx Informational
2xx Success
3xx Redirection
4xx Client Error
5xx Server Error

HTTP status codes help the client understand whether a HTTP request is successful or not.
Status codes are usually represented in three-digit numbers ranging from 100-599. Each status code has a different meaning.

The status codes 100-199 are used for informational responses.
The status codes 200-299 are used for successful responses.
The status codes 300-399 are used for redirection responses.
The status codes 400-499 are used for client error responses.
Finally, the 500-599 status codes are for server error responses.

Informational Response Codes

Response Code

```
HTTP/1.1 101 Switching Protocols
```

100 Continue
101 Switching Protocols

Informational responses are sent when the server receives the request and continues processing the request.

100 Continue response code is used when the client should proceed with the upcoming requests or ignore the response for the completed request.

101 Switching Protocol response code is used when the request switches from one protocol to another protocol.

Successful Response Codes

Response Code

```
HTTP/1.1 200 OK
```

200 OK
201 Created
202 Accepted

Successful responses are used when the server processes the request and provides the expected information.

200 OK is a standard status code used for successful HTTP requests.

201 Created status code is used to indicate the request has been fulfilled and resulted in a new resource being created.

202 Accepted status code is sent when the request has been accepted for processing, but the processing has not been completed.

Redirection Responses Codes

Response Code

```
HTTP/1.1 301 Found
```

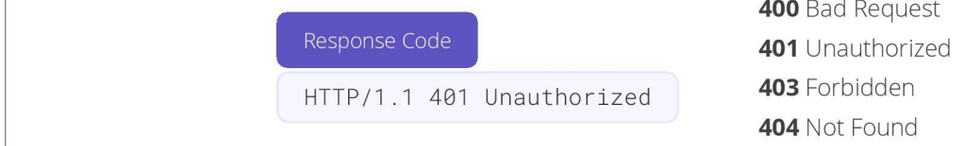
301 Moved Permanently
302 Found

Redirection responses are sent when the HTTP request is redirected to another location for further action.

301 Moved Permanently status code is sent when the current request and upcoming requests should be redirected to the new location.

302 Found status code is sent when the current request is being redirected to a different location temporarily.

Client Error Responses Codes



Client Error response codes are sent to the client when the server doesn’t find expected data in the HTTP request or when the HTTP request can’t be processed further by the server.

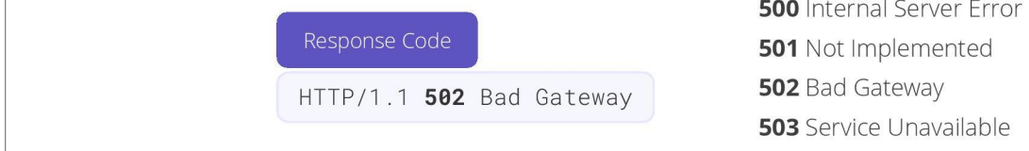
400 Bad Request status code is sent when the request does not have proper syntax.

401 Unauthorized status code is sent when the client doesn't have access to the resource.

403 Forbidden status code is similar to *401 Unauthorized* sent when the client doesn't have proper rights to access the resource or the client is not authenticated.

404 Not Found status code is sent when the requested resource could not be found on the server.

Server Error Responses Codes



Server Error response codes are sent when the server receives a valid HTTP request but could not process the HTTP request.

500 Internal Server Error status code is sent when the server does not know to handle the request.

501 Not Implemented status code is sent when the server does not recognize the request method or lacks the ability to fulfill the request.

502 Bad Gateway status code is sent when the server was acting as a gateway or proxy and received an invalid response from the upstream server.

503 Service Unavailable status code is sent when the server is down or unavailable.

Stateful vs Stateless Protocol

Stateful

One request is related to another succeeding request.

Stateless

A request does not retain any relation with preceding request

HTTP inherently is a stateless protocol, that is the server cannot remember to connect a HTTP requests to the previous HTTP requests.

A stateful protocol is the type of protocol where the initial HTTP request and next HTTP request has a connection with each other.

A stateless protocol is the type of protocol in which the client sends a request to the server and the server responds back depending on the data provided in the request.
If an HTTP request fails, the client has to send the same HTTP request again to the server for processing.
One request will not have any connection to the other request.

The File Transfer Protocol or FTP is an example of a stateful protocol.

The Hyper-Text Transfer Protocol or HTTP is an example of a stateless protocol.

Practical DevSecOps
The DevSecOps Training & Certification

Overview of API Architecture

In this sub-section, we will learn about API protocols, data formats, and different types of APIs.

In this sub-section, we will learn about API protocols, data formats, and different types of APIs.

API Protocols

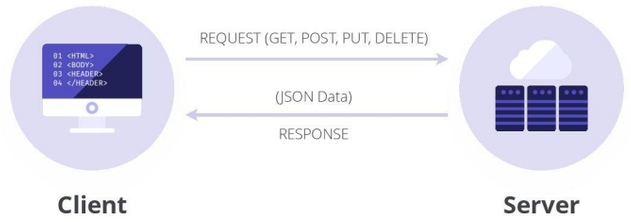
REST, SOAP and RPC

1

www.practical-devsecops.com

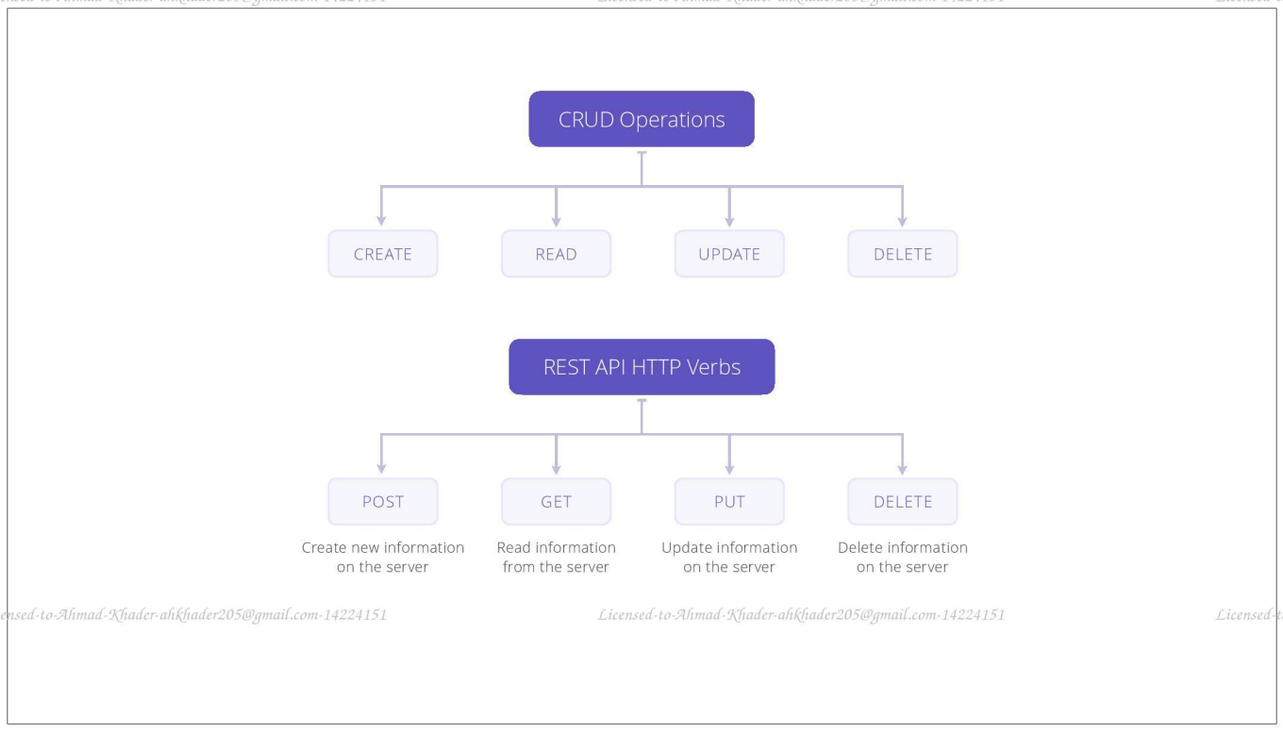
In this short lecture, we will explore the commonly used API protocols such as REST, SOAP, RPC.

REpresentational State Transfer



REST (REpresentational State Transfer) is one of the widely used API protocols. Well, REST is not a protocol in itself, as REST uses HTTP natively as it was meant to be.

REST makes use of HTTP for implementation like HTTP methods or verbs, URI, headers, body, and status codes. We will understand REST in the upcoming section.



REST used to be more of a convention than a set of rules. Arguably REST was using HTTP verbs as they were meant to be used.

Imagine there is some customer related information on the server. In database terminologies the acronym CRUD is used to represent the operation of Create, Read, Update, and Delete certain data.

Similarly, REST APIs emerged as a technique to Create, Read, Update, and Delete data maintained by a server through APIs.

HTTP method POST is used to *Create* new information on the server.
HTTP method GET is used to *Read* new information from the server.
HTTP method PUT is used to *Update* information on the server.
HTTP method DELETE is used to *Delete* information on the server.

REST URI Examples

URI	Methods	Intention
/api/users	GET	Gets list of users
/api/users/1	GET	Gets user with id =1
/api/users/	POST	Creates new user
/api/users/1	PUT	Update the user information
/api/users/1	DELETE	Deletes the user

Imagine there is an API that is used to manage customers or users.

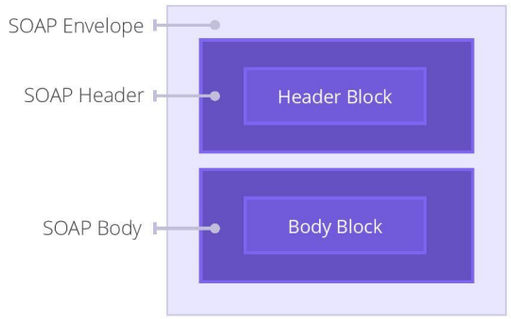
Sending a *GET* request to the API URI */api/users* would get a list of users on the system.

However, sending a *GET* request to the API URI */api/users/1* would actually retrieve information for a particular user with the ID 1.

Similarly, if new user needs to be created, then a *CREATE* request is sent to the API URI */api/users/1*, usually along with additional information such as a users's name and other details as a part of the HTTP request body in JSON format.

The HTTP verb *PUT* is used to update a user's information with the information that needs to be updated as a part of the HTTP request body in JSON format.

The HTTP verb *DELETE* with the URI */api/users/1* deletes the user with the ID 1.



Simple Object Access Protocol

SOAP (Simple Object Access Protocol) is a flexible and language independent protocol which helps to create Web APIs.

SOAP was the de-facto standard to create web API, or in SOAP's parlance as web services, until REST came along.

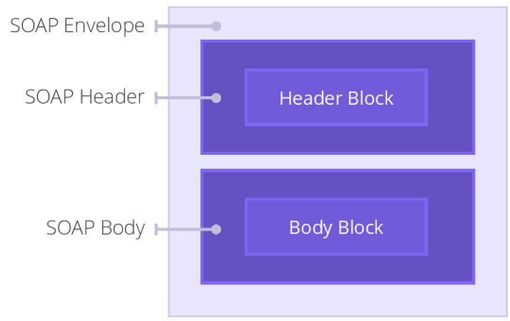
What is SOAP?

Simple Object Access Protocol (SOAP) is XML based protocol which is used to transfer data. SOAP is flexible and language-independent, which helps the developer to write APIs in their native language such as Java, C#, or Python, but the communications would later happen in XML format.

SOAP was the de-facto standard to create web API, or in SOAP's parlance as web services, until REST came along.

SOAP is still widely used to communicate between enterprise services.

Simple Object Access Protocol



SOAP is more of a message exchange format rather than a protocol itself. SOAP as a message exchange format can be integrated with other protocols such as HTTP, TCP, or SMTP.

The popularity of XML in the early 2000s lead to the popularity of SOAP implementations.

In its basics, SOAP has three elements. An envelope, a header, and a body. Compare that to a correspondence letter that is sent through a post office.

There is an envelope, the envelope has the receiver's address and other details on top of it, and then inside the envelope is the actual correspondence letter that is addressed to the receiver.

Similarly, SOAP has an envelope. Inside the envelope is a header, and a body. Header is optional and usually containers manifest information for a receiver or an intermediate transmitter. Body contains the actual message, and everything in SOAP is XML.

SOAP Message

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope>
<env:Header>
</env:Header>
  <SOAP-ENV:Body>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Let us look at a sample SOAP message.

As in the example snippet above, SOAP message starts with an envelope which indicates the start and end of the message. This helps the receiver process the message. And then there is a SOAP header, and a SOAP body.

SOAP Message

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
   SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">
<env:Header>
<m:keyVerification xmlns:m="http://travelcompany.example.org/reservation"
         env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
<m:key>dhf923nskAdnfk39lnsfk00amBdfML<m:/key></env:Header>
   <SOAP-ENV:Body xmlns:m = "http://www.practical-devsecops.com/products">
      <m:GetProductName>
         <m:ProductName>Practical DevSecOps</m:ProductName>
      </m:GetProductName>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

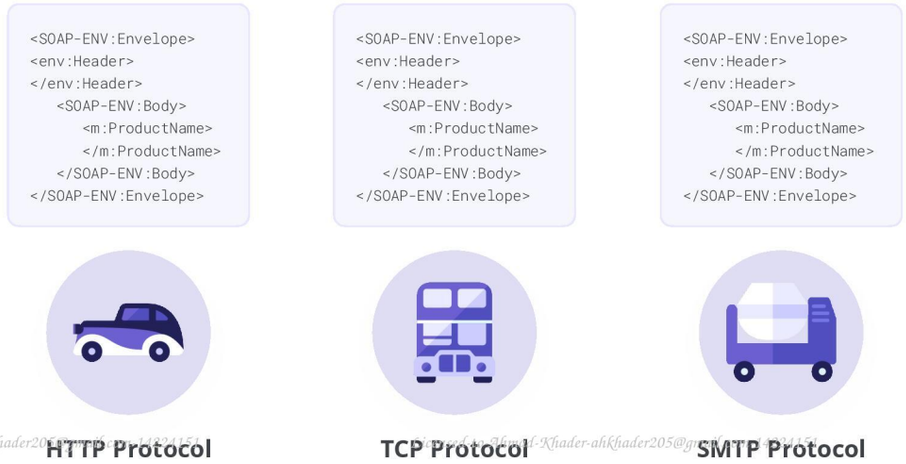
Envelope is a mandatory part of the SOAP message and this envelope should have at least one body element.
Every SOAP message has a root envelope element. SOAP envelope is specified using envelope element and prefaced with SOAP-ENV. The encoding specification, as well as the encodingStyle element, are optional.

Next, we have the SOAP header, which is an optional header which helps to specify application-related information. For example, adding some nonce, or sharing credentials, or public key information, or message digests to improve authenticity, and security of the SOAP message itself. Multiple SOAP headers can be defined in the SOAP message.

Next, we have the SOAP body, SOAP body is a mandatory element that contains the ultimate information that is intended for the server to process a business function. The body must be defined inside the envelope element.

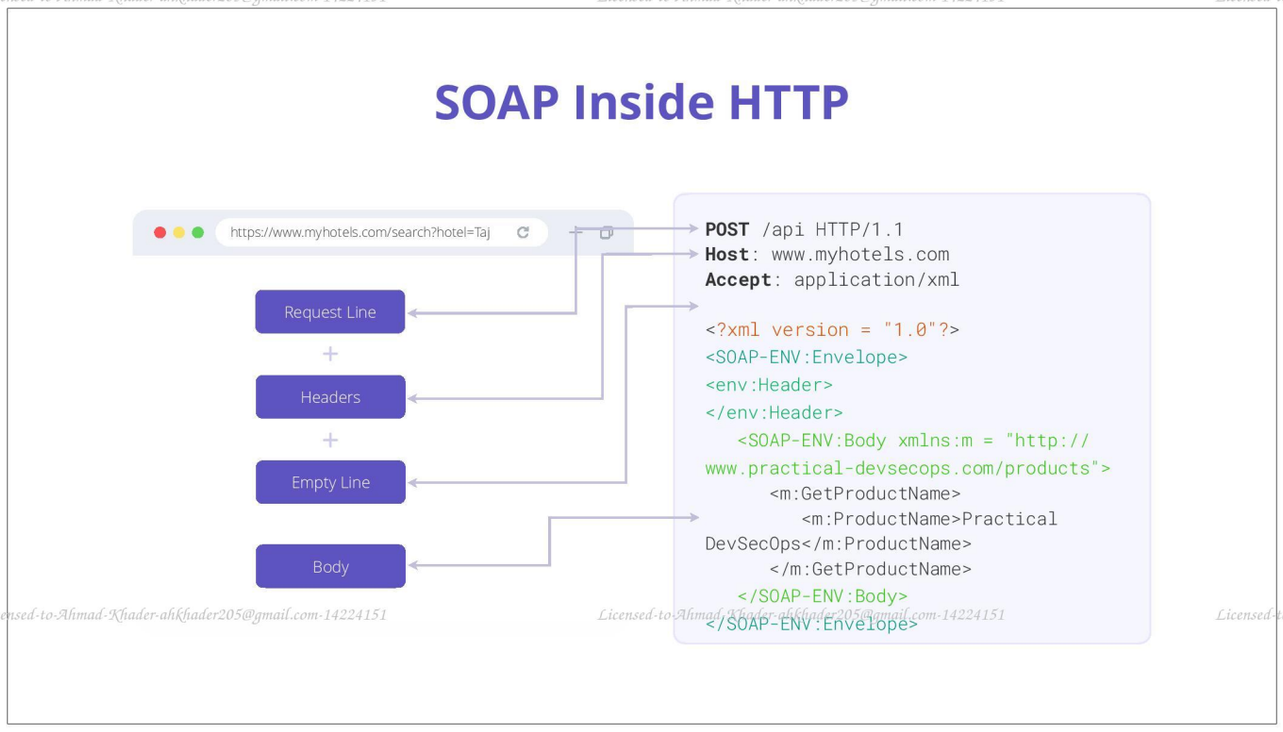
Now that we have seen how a SOAP message needs to be constructed let’s take a look at how SOAP works.

SOAP and Other Carrier Protocols



Since SOAP is a message exchange format, it needs carries just like how a corresponder letter from a bank is carried by a mailman to the recipient.

As an example we represent here that SOAP messages can be carried by the HTTP protocol, or directly the TCP protocol itself, or even in an SMTP protocol.



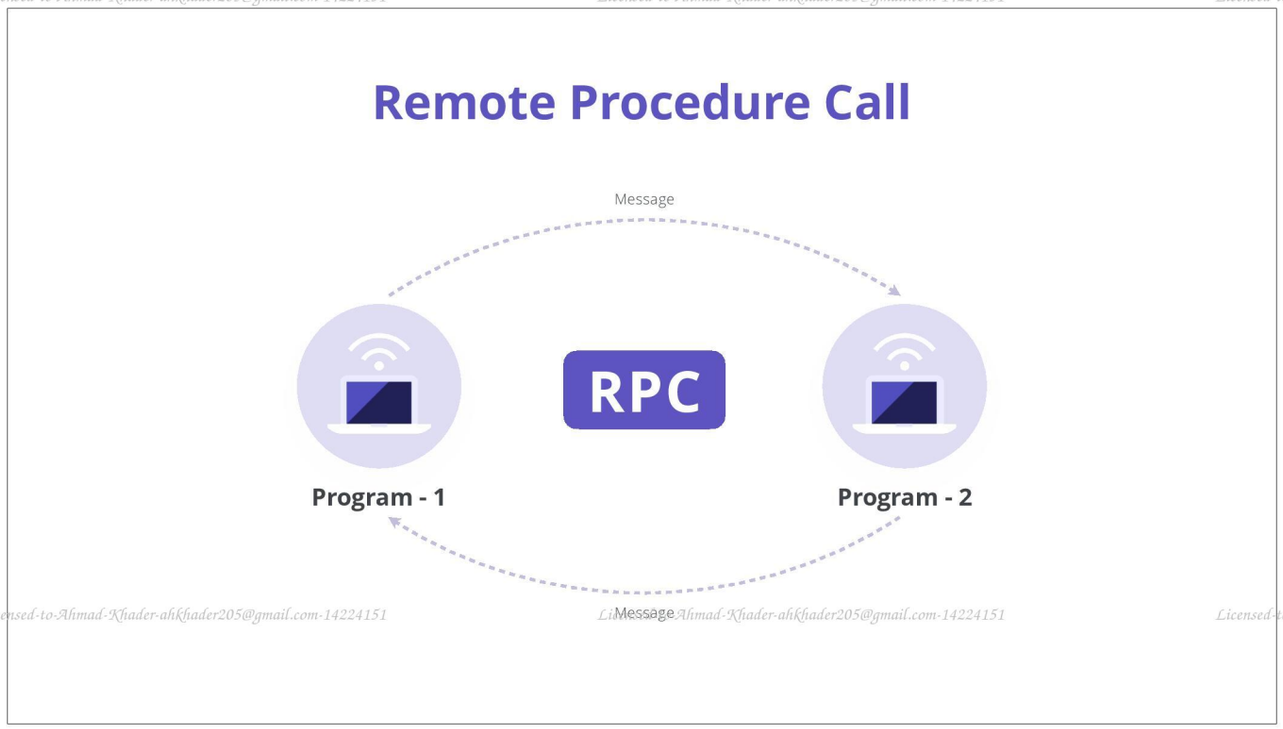
Let’s take an example of how SOAP messages are used in HTTP.

Here’s a sample HTTP request that we saw before.

The HTTP headers are stripped to the bare minimum. But, the most import of part is the HTTP request body.

The entire HTTP request body now has a structure, and that structure is the SOAP message format. Similarly, the HTTP response would also contain SOAP messages in XML.

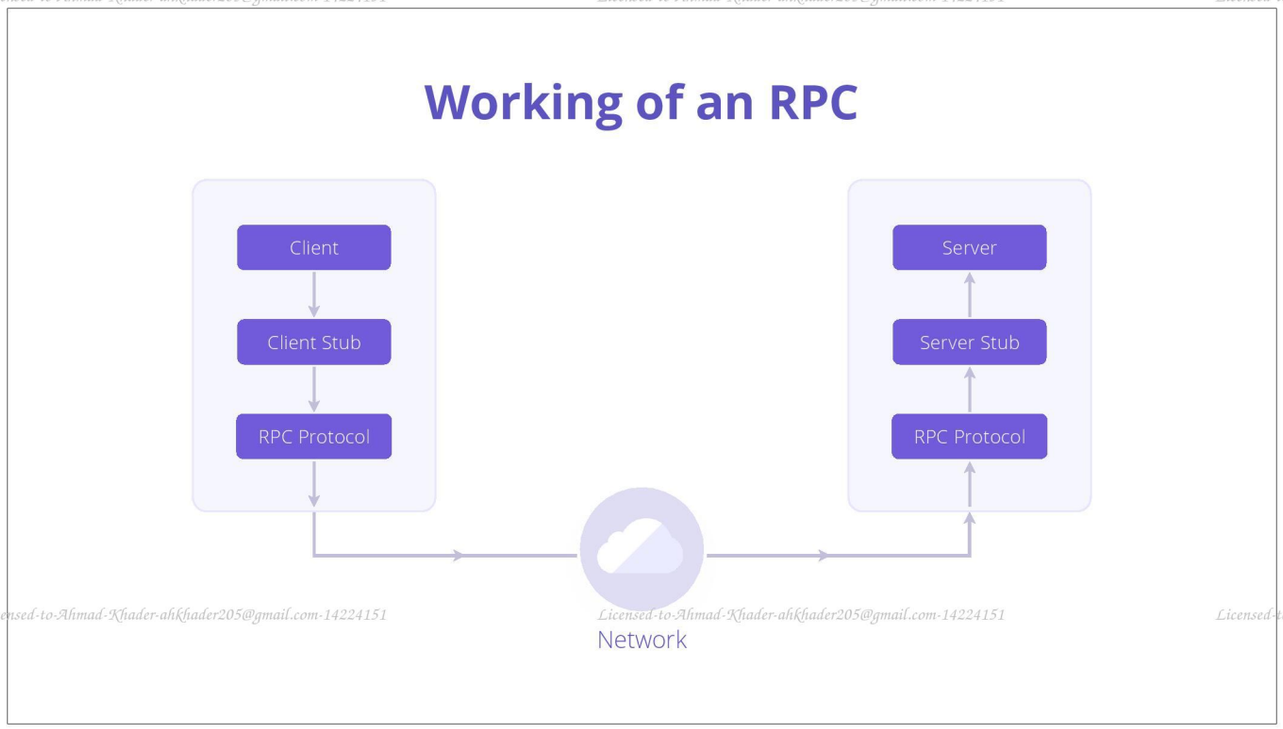
That’s just an example of how SOAP can be used with HTTP as a carrier.



Remote Procedure Call or RPC is an effective technique to call functions or procedures in the remote server. This mechanism helps to build distributed client-server-based systems. RPC mechanisms are typically used within an enterprise network.

RPC uses the client-server model. The program that calls the function is the client and the program that receives and provides a response is the server.

Like usual function calls, the arguments can be passed from one system to another and get processed. Let us look at this working of RPC through an example.



The client initiates the request and sends it to the client stub. The client stub which is a piece of code that converts the parameters into messages.

The client stub sends the message to the server stub using the RPC protocol.

The server stub receives the message and converts the message into arguments to then be passed on the server.

The server processes the request and sends back the response to the server stub. The server stub then sends back the response to the client stub via RPC protocol.

The client stub converts the message into parameters and sends it to the client.

Remote Methods Invocation in Java, COM, COM+ in Windows are some examples of RPC.

Practical DevSecOps
The DevSecOps Training & Certification

API Data Formats

JSON and XML

In this short lecture, we will explore the two most commonly used API data exchange formats, that is XML, and JSON.

JavaScript Object Notation (JSON)

```
[
  {
    "username": "John",
    "email": "johnloveshack@gmail.com",
    "fullname": "John Doe Smith",
    "phonenumber": "9873524278"
  },
  {
    "username": "Joseph",
    "email": "biden@gmail.com",
    "fullname": "Joseph Robinette Biden",
    "phonenumber": "900537283"
  }
]
```

JSON or the JavaScript Object Notation is a lightweight data exchange format which is easy for humans to read and write. JSON is developer friendly as it is easy to write and process the information using the code.

Virtually all modern programming languages support JSON in one form or the another.

JSON in API is used to transfer the information to the server and the client. JSON is sent over the request body or parameter as an encoded value.

At the current state of affairs, most APIs use JSON as a preferred data exchange format. Compared to XML, JSON data is shorter, and hence network load, latency, and processing time are significantly reduced in client server communications.

Extensible Markup Language

```
<?xml version="1.0"?>
<catalog>
    <book id="bk101">
        <author>John</author>
        <title>Java Developer</title>
        <genre>Systems</genre>
        <description>Create Java based web applications<description>
    </book>
<![CDATA[
Reserved XML characters can be used here< > & "
]]>
</catalog>
```

Extensible Markup Language (XML) is one of the widely used markup languages with the benefit of data specification and error handling.

Similar to HTML, angle brackets are used to construct XML. XML is also widely used in APIs particularly in SOAP kind of set up. With XML, you can define your own set of tags, and most importantly you can define the expectation of a XML data in the form of an XML schema.

XML can also be queried through X-PATH navigations. Binary data or literal data could be used inside XML through the CDATA sections.

Because of the huge set of features that XML offers, there are also attacks such as XML Bomb, Denial of Service that are possible through XML Entity Expansions.

Compared to a JSON message, the same XML message can be huge in size, however XML has other advantages like mentioned above.

Practical DevSecOps
The DevSecOps Training & Certification