

API COURSE INFOSYS SPRINGBOARD

Uber application:

How does Uber work?

- When you book a ride with it, you can see the cab's and your relative location
- You get a message when the driver is near you
- After the ride, you also make payments digitally using Paytm, UPI or card
- You get an email on how much you spent on your last trip- the invoice basically

How do you think Uber achieves all this? Do you think they have written code for each of these operations- to track location, to send messages, to make payments and to send emails?

Well, as a startup, Uber has been smart and saved itself thousands of hours of work. How? They reused the codes already written for these functionalities.

And how did they do it? Via APIs, of course! And as a result, Uber has been able to grow far more rapidly than if it had built any or all of these functionalities itself

The focus is on reuse and API enables it!

Uber uses:

- Google Maps API for showing distance and directions
- Twilio API for sending SMSes
- Braintree API for making payments
- SendGrid API for email notifications, and so on...

Why to reinvent the wheel! Why rewrite the code that exists already.

Uber is not the only one, it's the story of all modern applications.

Take Airbnb; it also uses Twilio, Braintree and SendGrid APIs. And for SMS and voice verification, it uses Nexmo API.

Uber and Airbnb have some generic aspects to their businesses – automated notifications, payment processing, user verification, and so on – that it only makes sense to use third-party APIs to take some of their loads off, and focus their energy on their core business.

What we can infer from these use-cases is that organizations are no more interested in building every aspect or functionality of their business in-house. They would rather reuse functionalities, from the ones that are already the best in the market.

Look at this web site called programmableweb.com that houses an API repository containing more than 20k APIs that are popular in their areas of work. Here we get APIs by category like Mapping APIs, Mobile APIs, Cloud APIs, Payment APIs and so on.

To summarize, there is a huge demand to integrate and exchange data or resources. APIs are the innovation being adapted by industry at large.

And to add on, it's not just about integrating two applications, it's about a new way of building applications, or about assembling your application using APIs.

As we see, Uber assembled a significant portion of itself using so many APIs.

Definitely a revolution!

TrustedBank is a large banking organization that provides services such as receiving deposits, providing loan and financial supports in offline mode i.e., at its bank branches.

A team of developers headed by the CIO of the TrustedBank has developed several services, which will increase the bank's value, and help in better customer reach.

The managing director of this bank also wants these services to be exposed to other developers who can leverage these services for the development of other applications.

Hence, the services created by the bank can be exposed to public by using the concept called Open APIs

The following are the advantages of Open APIs:

- Innovation: One of the key business benefits from Open API is that it helps an API provider leverage the creativity of other developers to gain profit opportunities
- Target on specific markets: The Open APIs targets on specific products or specific group of users or market needs and satisfy them. For example, a sports application providing the information about the sports, target the audience who are sports enthusiast
- Ability to integrate multiple systems: These APIs enable the integration of different systems to provide the easily accessible data and to create better solutions. This helps add the new features and services to the application or software
- Reduce delivery time: Open APIs help in reducing the time to deliver the product in the market.

APIs are the platform-neutral, customizable customer facing interfaces that enable different business, with distinct software components and services to effectively communicate with each other. The innovative use of APIs has enabled organizations to grow and let grow, by collaborating with third-party businesses and external developers, creating a channel for revenue generation- called the API Economy.

So far you have learnt about:

- Banks open their data, functionalities and services to third-party developers through Open APIs to better serve their customers and clients, and to shift the paradigm of banking to enable accelerated growth
- Open banking allows banks to collaborate with Fintechs and startups to promote their services, and become more relevant and competitive
- API needs to be viewed as a product, by constantly working on the clients' feedback through the complete API lifecycle

As an employee of InApp, anyone can understand the requirements and how the response will be generated. Since the language RAML, it is easy to generate machine-readable documentation.

```
##%RAML 1.0
title: InApp Employee Directory
/employees:
  get:
    queryParameters:
    name:
      displayName: Employee Name
      type: string
      description: Employee's with the mentioned name
      example: Mary
      required: false
    joiningYear:
      displayName: Joining Year
      type: number
      description: Employees who joined the company in that year
      example: 1994
      required: false
    emailID:
      displayName: Email ID
      type: string
      description: Company's email id for any official communication
      example: mary.r
      required: false
  responses:
    200:
      body:
        application/json:
          example: |
            [{
              "employee": {
                "id": "EM81342",
                "name": "Mary Roch",
                "emailID": "mary.roch@xyz.com",
                "joiningdate": "13-11-1994",
                "contact number": "9999999999",
              }, {
                "id": "EM81342",
                "name": "Mary Ray",
                "emailID": "mary.ray@xyz.com",
                "joiningdate": "11-8-1994",
                "contact number": "9999999999",
              }
            ],
            "success": true,
            "status": 200
          }
        /{employeeID}:
          description: To fetch employee details based on their ID
          get:
```

responses:

200:

body:

application/json:

example: |

```
{
  "employee": {
    "id": "EM81342",
    "name": "Mary Roch",
    "emailID": "mary.roch@xyz.com",
    "joiningdate": "13-11-1994",
    "contact number": "9999999999",
  },
  "success": true,
  "status": 200
}
```

Idempotence, in programming and mathematics, is a property of some operations such that no matter how many times you execute them, you achieve the same result

- Some of the different API design architectural patterns are Uniform Resource Identifier (URI) Design, Even-Driven Architecture (EDA) Design, Hypermedia Architecture (HA) Design, and Tunneling Styles.
- The multiple ways in which API can be designed are REST, REST-like, SOAP, RPC, Falcor, GraphQL, and gRPC.
- JSON-LD is a lightweight linked data format based on JSON format.
- JavaScript Object Notation is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute-value pairs and array data types.
- Push and Pull models are used to restrict unwanted messages and traffic on the internet.
- WebSockets is an advanced technology which makes an interactive communication between browser and server possible.
- Using HTTP/2 Server Push, the web server can "push" the content to the client/user before the client request.
- Server-Sent Event is a standard that allows browser or user to receive any updates from the server over an HTTP connection.
- OData (Open Data Protocol) is an ISO/IEC approved, OASIS standard that defines a set of best practices for building and consuming RESTful APIs.

Here is some of the top API Management tools in the market:

1. MuleSoft
2. Google Apigee
3. Amazon API Gateway
4. Kong
5. Microsoft Azure

API Management is the process of deploying, documenting, monitoring, analyzing and managing APIs. To achieve this, there are API Management platforms available in the market from several vendors like Google. Regardless of the vendor, there are few common features such as:

1. API Analytics
2. Monetization
3. API Security
4. Developer Portal
5. API Gateway

When there are several calls made to an API, security becomes an essential factor. When enterprises open their applications and data to the world, there are new threats, and companies are racing against the clock to fix the issues.

Features of API Management tools are:

1. Basic authentication
2. API keys
3. OAuth (Open Authorization)
4. SAML (Security Assertion Markup Language)
5. Threat protection

API gateways are used to secure and mediate the traffics between clients and back-ends. It acts as an intermediate between a company's APIs and the API consumers with the following features:

1. Accepts API calls from the user and routes them to back-end
2. Verifies API keys, and other credentials
3. Enforces policies on the API
4. Log errors and responses
5. Analyze the traffic
6. Transform response from back-end before sending to client

Web APIs can be created by following architectures including REST, SOAP, RPC etc. However, REST architectures dominate the most widely used APIs available in the market and industry. Hence, it would be a good place to start investigating the different forms of security mechanisms that can be used to protect Web APIs.

HTTPS should be preferred as the protocol instead of the usual HTTP. HTTPS requests and responses are communicated in an encrypted format, that increases the protection from unwanted data theft.

Security

Authentication means proving identity.

Authorization means having access and permissions to perform a set of actions.

Following are the various ways in which Authentication and Authorization mechanisms can be implemented:

1. HTTP Authentication and Authorization
2. Proxy authentication
3. API Key
4. Limit Access
5. Geofencing
6. L7 DOS Protection

API Key

API keys can be used to protect API from unauthorized access, API keys are tokens provided by client while making an API call. They're a secret parameter, only known by the client and the server. API Keys can also be called as Public Keys, Consumer Keys or App Keys.

One can send an API Key as a request header or cookie, or they can be sent via query parameters. To ensure that API Key based authentication is secure, it should be used with other security features like SSL or HTTPS.

API Keys are primarily used because they're simple, short, static and provide an easy way to communicate amongst multiple services. Moreover, they also don't expire, unless they're revoked.

Limit Access

Limiting access to an API endpoint is essentially important if the API is exposed to public.

This would help in limiting the threat from potential DOS attacks (Denial of Service attacks, which render the targeted servers unable to service authentic requests).

Rate limiting option is available in various API Management products in the market, like Google's Apigee, MuleSoft, Microsoft's Azure API Management.

Apart from the API Management, there's another tool known as ThreatX, which identifies the specific threat profile for an application and automatically stops the threats while allowing authenticated requests.

Step 2- Input Validations

A web transaction begins from the point where the user enters some data to be sent to the server. At this position input validation mechanisms can be applied.

Data points like Post Office Area code, Date, etc. should be invalidated if the entered data doesn't conform to the input field's specified data type. This type of input sanitization ensures security at the first level of transaction.

Following types of validations:

1. XML Input Validation
2. Validate Incoming Content-types
3. Validate Response Types

4. Secure Parsing
5. Strong Typing
6. URL Validations

The server should always check for the type of content incoming in the PUT or POST methods, in the header called "Content-Type". Only if the content-type matches the data allowed to be stored on the server, should it be allowed to proceed.

Examples include application/xml and application/json.

A 406 Not Acceptable response should be returned in case of incompatible content-type.

The possibility of attacks coming towards the server can be reduced if the range or type of symbols is limited to be entered into the input fields.

For instance, the legitimate values can be limited to be either true or false, only numeric digits, etc.

Similarly, the content types of incoming requests for the server, the server should only allow the requests which can accept the type of response that servers can provide. This information is kept in the 'Accept' header.

Step 3- Output Data Encoding

Security Headers

- To ensure correct interpretation of a resource's content by the browser, a content-type header with the proper content-type and a charset should be sent by the server
- X-Content-Type-Options: nosniff should also be sent by the server so that the browser does not detect a different content-type, rather than what was sent
- The client should also send X-Frame-Options to deny protection against drag and drop clickjacking attacks in old browsers

JSON Encoding

- JSON Encoders prevent remote code execution of arbitrary JavaScript in the browser, making this a key concern about them. A proper JSON serialize is required for encoding of data supplied by the user in-order to prevent execution of user supplied input on the browser
- Consider using .innerText/.value/.textContent rather than .innerHTML while inserting values in the browser to protect against DOM XSS attacks

XML Encoding

Instead of building XML by string concatenation, use XML Serializer for building it, to ensure that the content sent to the browser does not contain XML injection and is parse-able

Step 4- HTTP Status Codes



The HTTP mechanism provides a long list of status codes that are used to convey the type of response received from the server. Hence, the client system can understand whether its request has been passed or failed just by looking at the response status code.

The most used status codes are 404, 200, 500, etc. However, while designing REST API, the status codes must be used precisely. Using status codes to handle errors can significantly reduce the risks of potential security breaches. Below are some of the status codes:

Status Code	Explanation
200 OK	This status code depicts a successful HTTP action. It can be returned for GET, POST, PUT, PATCH, or DELETE methods.
400 Bad Request	This status code means that the request sent to the server has not been sent in the permitted format and needs to be changed.
401 Unauthorized	This status code means that the client is trying to access a protected resource without providing the necessary credentials.
403 Forbidden	This status code means that even though the client has proven its identity in accessing a protected resource, it doesn't have the required access to read it.
404 Not Found	This status code is used to convey that the resource URL given by the client is either non-existent or the resource has been deleted and no longer available.
405 Method Not Allowed	<p>This status code means that the client has sent an un-permitted combination of HTTP method and resource URL.</p> <p>For instance, sending a POST request to a method which accepts only GET request i.e. a read-only resource. This will produce a 405 status code.</p>

You are familiar with the different API security mechanisms. Below is the table with different types of HTTP authentications:

Type	Explanation	Reference Image
------	-------------	-----------------

<p>Basic Authentication</p>	<ul style="list-style-type: none"> • Authentication is done by the client using user ID and password which are validated using a string for comparison • HTTP Framework utilized is as shown below: <p>challenge = "Basic" realm</p> <p>credentials = "Basic" basic-credentials</p> <ul style="list-style-type: none"> • The client sends the ID and password together, separated by a single colon (":") character to enhance the security. A base64 [7] string is encoded in the credentials • Below is a sample header field for Basic Authentication: <p>Authorization: Basic QWxhZGRpbjpvYVUHNlc2FtZQ==</p>	
<p>Digest Access Authentication</p>	<ul style="list-style-type: none"> • Is based on challenge-response protocol in which the client requests a secure URL and server replies with the realm and a nonce value, which is used by the client to calculate the MD5 checksum value • This scheme applies a hash function to the credentials before sending them onto the network • A different hash value is produced for every authentication because of the nonce value, this is done to avoid replay attacks and also provides some protection against the hackers as the plaintext password is not sent over the wire 	
<p>Bearer Authentication</p>	<ul style="list-style-type: none"> • Bearer or token authentication provides security tokens called bearer tokens, which essentially mean 'giving access to the bearer of this token' • A token is a cryptic string, generated as a response by the server to a login request • A client is required to send this token in the authorization header while accessing protected resources as: <p>Authorization: Bearer <token></p> <ul style="list-style-type: none"> • As in Basic authentication, one must use bearer authentication only over HTTPS (SSL) 	

So, basic authentication and digest authentication schemes provide authentication by using a username and a secret, whereas bearer authentication uses a token for authentication.

Microservice

Let us take an example of an e-commerce application, that takes the purchase order from the customers, provides the payment options, and shipping the order. This application is deployed as a Monolith where all the functionalities provided by the application are tightly coupled. If the application wants to add a new functionality of returning of orders by customers, then the whole application needs to be changed. This creates delay in the deployment of application.

The solution for this problem can be provided by converting the monolith application into microservices. The steps to convert a monolith application to microservices are:

1. Start with a simple and loosely coupled functionality
2. Divide the backend and frontend
3. Identify the modules and extract them

The monolith to microservices conversion can be started by taking the small functionality/part that is not much dependent on the monolith. Implement that part as a microservices and connect that with monolith. While selecting, start with the one that don't require the any changes in the client side implementation.

One method that can be used that is, when developing a new functionality implement them as a microservice and then integrate to monolith. For example, consider an e-commerce application, take the user login functionality implement them using microservice and integrate them to older application.

In general, an enterprise application consists of three layers of components: presentation layer, business layer and data-access layer. Separate the functionalities provided by each layer.

After splitting the layers, each layer connects other remotely using APIs. Divide the backend and frontend the request remotely using the APIs. This helps in faster development of application, easy deployment, and also the independent scaling of the applications. This also reduces the complexity of testing the applications.

After separating the different layers, identify the modules/functions that can be converted to services. First create an interface between the monolith and the microservice. Then start implementing each service as an independent microservice and integrate it to monolith using the interface.

When starting with conversion of module to service, select the service which can be easily extracted and implemented. Without decoupling the data, the architecture of application cannot be said as microservices. Hence decouple the data using the data migration depending on the environment, type of data needed etc.

All the services interact with each other using the APIs. Once all the modules are refactored as service and integrated, each service then can be implemented independently.

So far, you have learnt that:

- Microservices is the architecture of the present, the preferred one ever since Netflix made it popular
- In monolithic architecture, the entire application is a single thing. It is self-contained, and the functionalities of the application are interconnected and interdependent. It is not possible to scale up a specific functionality
- Microservices is an architectural style of developing an application as a suite of small services, each taking care of a business functionality. Each microservice can be deployed independently, and scaled up and down based on business needs
- Unlike Monolithic architecture where all the applications are tightly coupled with each other, in Microservices application it is not the same
- Microservices use APIs to form a network and communicate with each other
- The end user will see no difference in terms of usage if an enterprise architecture is made Microservices, as the functionalities at the application boundaries remain the same
- Microservices has not taken away the use of SOA and ESB from the nature of integration. Depending on the business need, architecture and feasibility, the type of integration is chosen

In the digital world, all the applications are commonly delivered as service, which are called as Web apps or SaaS (Software as a Service). The 12-Factor App Model is a methodology that is used for building software-as-a-service apps that:

- Uses declarative formats for setup automation, to minimize time and cost for new developers joining the project;
- Has a clean contract with the underlying operating system, offering maximum portability between execution environments;
- Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration;
- Minimizes divergence between development and production, enabling continuous deployment for maximum agility;
- Scales up without significant changes to tooling, architecture, or development practices.

The 12-factor app model can be applied on the applications that are built using any kind of programming languages and the applications that use any combination of back-end services like database, queue, memory cache, etc. The 12-factor app model will lead to:

1. Microservice
2. Cloud based applications
3. Team efficiency

The 12-Factors of 12-factor app model are:

1. Codebase: One codebase tracked in revision control, many deploys
2. Dependencies: Explicitly declare and isolate dependencies
3. Config: Store config in the environment
4. Backing services: Treat backing services as attached resources
5. Build, release, run: Strictly separate build and run stages
6. Processes: Execute the app as one or more stateless processes
7. Port binding: Export services via port binding
8. Concurrency: Scale out via the process model
9. Disposability: Maximize robustness with fast startup and graceful shutdown
10. Dev/prod parity: Keep development, staging, and production as similar as possible

11. Logs: Treat logs as event streams
12. Admin processes: Run admin/management tasks as one-off processes

1 codebase

A 12-factor app is usually tracked using the version control systems like Git, Subversion. The database where the copy of the revision is stored is known as code repository or code repo or repo.

- A codebase is a single repository (which is a centralized repo like Subversion) or any sets of root repos which shares the root commits (which is a decentralized version control system like Git). The app and the code repo has one-to-one relation.
- If any application contains multiple codebases, then it's not an app – it will be a distributed system. In a distributed system each component is an app and each app can comply with 12-factor.
- A single codebase cannot be shared by multiple apps. This will be a violation of twelve-factor. This issue can be resolved by sharing code into libraries which can be included through the dependency manager.

2 dependency

- A twelve-factor app doesn't rely on implicit existence of system-wide packages. Here all the dependencies are declared completely and exactly using a dependency declaration manifest.
- For further assurance, that there is no leak of dependencies from the surrounding systems, it uses dependency isolation tool during execution. Also full and explicit dependency specification is applied uniformly to both production and development.
- Twelve-factor apps also do not rely on the implicit existence of any system tools. For example, some shelling out tools like ImageMagick or curl, we cannot guarantee that these tools will be present in all the systems. If the app needs to shell out to a system tool, that tool should be vendored into the app.

3 config

- The configuration of an application is one of the important factors which is more likely to vary between the deployments like staging, production, etc.
- Externalize the configurations which are specific to environments.
- The application occasionally stores them as constants in the codebase, which is the violation of 12-factor. These config files must be strictly separated from the code because the config may vary across the deploys, whereas code remains same.
- An empty `example.properties` file that contains all the properties that needs to be configured like database credentials, key-secret pairs of external systems can be kept. So that all the properties information is in one place and it's easy to discover the configurable properties based on the scenario.

4 Backing services

- A backing services are the services that are consumed by the application over the network for execution. The common backing services are data stores like MySQL, SMTP services, messaging, or queuing services.
- The backing services are generally managed by the system administrators. Backing services can be managed locally or managed by external third-party service providers.
- The code for a twelve-factor app makes no distinction between local and third party services. All the backing services are attached as resources for the app. An app must be able to switch between the locally-managed and third-party services.
- Each distinct backing service is a resource for the app. For example, consider that there are three MySQL databases are associated with app. The app considers each of the database as a distinct resource. This shows that the loose coupling of resources to the app.
- The resources can be attached or detached from the app. If any resource needs to be removed, then that can be detached from the app without the changes in the code.

5 Build release run

The three stages of the transformation of codebase into deploy are:

1. Build stage: It's a transform which converts a codebase or code repo into an executable bundle called build. The build stage fetches the dependencies and compiles binaries and assets based on the last commit specified by the deployment process.
2. Release stage: This stage takes the build from build stage and combines it with the deploys current config. The release stage will contain the build and config that are ready for execution in the execution environment.
3. Run stage: In this stage based on the selected release, it runs the app in the execution environment.

It is impossible to make changes for an app during the runtime. Hence 12-factor app uses the strict separation between the build, release and run stages. Builds are initiated by the app developer when there is a new release of code. But runtime execution can occur because of server restart or process manager restarting the process that is crashed. The separation of these stages will help in automation and easy maintenance of app.

6. Processes: Execute the app as one or more stateless processes

- In this stage the app is executed in the execution environment as one or more different processes.
- The app may be deployed in the local system using the language runtime and the process is executed using the command line. On the other hand, the app may be deployed in the sophisticated system which may use many processes and can be instantiated into many processes.
- Twelve-factor processes are stateless and shares-nothing. If any of the data needs to be persisted, then it must be stored in the stateful backing service like database.

7. Port binding: Export services via port binding

- Web apps are occasionally executed in a webserver container.

- The twelve-factor app doesn't rely on the runtime injection of webserver into execution environment to create a web-facing service. Hence twelve-factor app is completely self-contained.
- The web app exports HTTP as a service by binding to a port, and listening to requests coming in on that port.
- Port binding approach also means that one app will become the backing service for another app. Therefore, by providing the URL to backing app as a resource for the consuming app.

8. Concurrency: Scale out via the process model

- In the twelve-factor app, processes are a first-class citizen.
- All the processes in the twelve-factor app take signal from the UNIX process model for running service daemons. Using this model, the developer can handle diverse workloads (load balancing) by assigning each type of work to a process type. For example, all the requests from HTTP receiver can be handled by web process and the background tasks can be handled by worker process.
- The nature of 12-factor like share-nothing, horizontally partitionable means that adding more concurrency is a simple and reliable operation. The array of process types and number of processes of each type is known as the process formation.

9. Disposability: Maximize robustness with fast startup and graceful shutdown

The twelve-factor app's processes are disposable, that means they can be started or stopped at any time.

This facilitates in:

- Fast elastic scaling
- Rapid deployment of code or config changes
- Robustness of production deploys

10. Dev/prod parity: Keep development, staging, and production as similar as possible

In the digital generation, the customers expect the gap between the development and production to be small. In the 12-factor model achieves that by using the continuous deployment which keeps the gap between development and production small.

The three gaps are described below:

1. Make the time gap small: The developer can write the code and can deploy it within an hour and even just the minutes later.
2. Make the personnel gap small: The code developers who deploy it take involvement by watching its behavior in production closely.
3. Make the tools gap small: By keeping development and production as similar as possible.

11. Logs: Treat logs as event streams

- Logs always provides us the visibility of the behavior of a running app.
- Logs are the stream of aggregated, events based on time collected from the output streams of all running processes and backing services.
- A twelve-factor app doesn't concern about the routing or storage of its output stream.
- Here, each running process writes its event stream, not buffered, to stdout.

12. Admin processes: Run admin/management tasks as one-off processes

The process formation is the array of processes that are used to manage the app's regular business handling web requests as it executing. Some of the administrator tasks to maintain the app are:

- Running the database migrations
- Running the console
- Running scripts

In this course, you learnt about:

API in today's digital context, which technology they use, how they are used in business, and how API connects to Microservices.

- API:
 - APIs are business oriented and consumer-centric REST based web services, also called modern and/or web APIs
 - There are three types of APIs depending on the type of consumers namely, public, private, and partner APIs
 - Always treat customer-centric APIs as product, and follow the defined lifecycle where you take and work on the constant feedback from clients to generate continuous revenue from loyal customers
- Secure APIs:
 - The 4 steps can be defined to secure APIs
 - An understanding of the different types of security mechanisms is required before choosing one/more for business
- Restful API Design best practices:
 - Design of API is an integral part of the API life-cycle management
 - Richardson maturity model provides different levels of restful compliance for APIs in resource compliance, HTTP method compliance, and HATEOS
 - Response structure covers different best practices in constructing the API response
 - Security features cover the concepts of error handling, security, versioning.
- API Management:
 - API Management is the process of creating, publishing web APIs, and enforcing usage policies

- API Management Platforms facilitate API Deployment, Developer Engagement, API Documentation, API Gateway and Security, Monitoring Performance, Traffic Management, Version Management and API Monetization
- Some of the popular API management platforms are Google Apigee Edge, MuleSoft Anypoint Platform, etc.
- Open Banking API and Open Banking:
 - Open banking API offers banking customers the ability to understand the features offered by various financial institutions and compare them to choose what is best for them.
 - This helps in creating a more convenient and secure experience for customer
- 12 Factor App model:
 - The model serves to create apps which follow the traditional style of software-as-a-service
 - This expedites the process of creating web-apps, makes them scale-able, flexible, available on cloud platforms, and automates the basic tasks to save time and costs of the project.
- About Microservices:
 - Microservices is an architectural style of developing an application as a suite of small services, each taking care of a business functionality. Each microservice can be deployed independently, and scaled up and down based on business needs
 - Microservices use APIs to form a network and communicate with each other.
 - Microservices hasn't taken away the use of SOA and ESB from the nature of integration. Depending on the business need, architecture and feasibility, the type of integration is chosen

Useful links

<https://www.programmableweb.com>

<https://detectlanguage.com/documentation>